**Теория Тестирования**

**? Качество программного обеспечения**

Это совокупность характеристик ПО, относящихся к способности **удовлетворять установленные и предполагаемые** потребности (Способность **удовлетворять установленные и предполагаемые** потребности).



**? QA / QC / Testing**

**Обеспечение качества** – совокупность мероприятий, охватывающая все **технологические процессы** разработки, внедрения, эксплуатации ПО на всех этапах жизненного цикла ПО, направленных на **обеспечения** требуемого уровня качества ПО

**Обеспечение качества** – подготовка и установка стандартов, анализ качества, выбор инструментов, предотвращение ошибок, совершенствование программы

**Контроль качества** – совокупность **действий,** производимых над ПО в процессе разработки, для получения инфы об актуальном состоянии ПО в аспектах **готовности** ПО к выпуску, **соответствию** заявленным **требованиям**, **соответствию** заявленному **уровню качества**.

**Контроль качества** – анализ результатов тестирования, поиск и устранение ошибок; анализ кода, технические обзоры и проверка программы

**Тестирование** – **проверка** продукта на соответствия **заявленным требованиям** и **нахождению дефектов**

**Портрет тестировщика:**

* Коммуникабельность
* Внимательность
* Небезразличие к судьбе продукта
* Уметь поставить себя на место пользователя
* Логическая мышление
* Изобретательность
* Усидчивость
* Умение концентрироваться

? Ошибка / Дефект / Поломка (Error/Defect-Bug/Failure)

**Error** – непреднамеренное действие человека

**Defect/Bug** – результат ошибки, кот приводит к невозможности выполнять функции

**Failure** – отклонение, сбой в системе

Error 🡪 Defect-Bug 🡪 Failure

**? Валидация / Верификация**

* Валидация (validation)— доказанное объективными результатами исследования **подтверждение** того, что **для конкретного определенного использования** приложения требования были выполнены. [ISO 9000]
* Верификация (verification)— доказанное объективными результатами исследования **подтверждение** того, что **определенные** требования были выполнены. [ISO 9000]

**Validation** — ’is this the right specification?’.  
**Verification** — ’is the system correct to specification?’.

**? Цели тестирования (test objective, test target)** — это причина или цель разработки и выполнения теста**:**

* Собрать и предоставить информацию об актуальном состоянии ПО и показать, что ПО соответствует предъявленным требованиям = убедить, что ПО **отвечает** оригинальным **требованиям и спецификации**;
* Показать, что ПО удовлетворяет потребности пользователей = обеспечить **уверенность в ПО** (пользователям, заказчикам и т.д.)
* Находить дефекты = обеспечить **очищения** ПО **от ошибок** (Вы не можете предоставить 100% покрытие, но Вы должны сделать все возможное, и гарантировать, что очевидные ошибки исправлены);

**? Принципы тестирования**

1) Тестирование демонстрирует наличие ошибок, но не их отсутствие – не нашли ошибок, не значит что их нет

2) Полное тестирование невозможно – 100% протестировать просто не реально

3) Раннее тестирование – легче и дешевле проводить тестирование на ранних этапах

4) Кластеризация дефектов – 20% функционала содержит 80% ошибок

5) Парадокс пестицидов – одинаковые тест кейсы перестают работать

6) Тестирование зависит от контента – сайт на 45 страниц будет тестироваться иначе, чем сайт на 45 000 стр.

7) Заблуждение об отсутствии ошибок – протестировали, всё пофиксили, не нашли багов, но ПО какое-то не то… 🡪 Валидация (после всех этих фиксов и изменений ПО стало сильно громоздким)

**? Требования** — это спецификация (описание) того, что должно быть реализовано.  
Требования описывают то, что необходимо реализовать, без детализации технической стороны решения. Что, а не как.

С точки зрения пользователя: Требование — некое св-во ПО, необходимое пользователю, для решения **проблемы** при достижении поставленной цели.

С точки зрения заказчика: Требование — некое св-во ПО, кот должна обладать система или её компонент, чтобы удовлетворять требования контракта, стандарта, спецификации либо иной формальной документации.

**? Откуда берутся требования**

* Бизнес аналитик
* Продакт менеджер
* Заказчик

**? Пользовательские требования**

**Use case** – Описание поведения системы, когда она взаимодействует с кем-то (пользователь) или чем-то (модуль, компонент) из внешней среды. Система может отвечать на внешние запросы или сама выступать инициатором взаимодействия.

Пример:

1) Пользователь захотел разместить объявление.

2) Пользователь зашёл в систему.

3) Пользователь авторизовался в системе.

4) Пользователь создал объявление.

5) Система отобразила сообщение об успешном создании объявления.

**User Story** – Способ описания требований, к разрабатываемой системе, сформулированный как одно или несколько предложений на повседневном или деловом языке. Цель в том, чтобы оперативно и без накладных затрат реагировать на быстро меняющиеся требования реального мира.

Пример:

**Как** <Роль/персона пользователя> **я** <Хочу что-то получить>**,** <С такой-то целью>

**Как** <пользователь>**,** **я** <Хочу управлять рекламными объявлениями>**,** <Чтобы удалять устаревшие или ошибочные объявления>

**User Scenario** – Терминал удостоверяется, что пополнение возможно, запрашивает у пользователя номер телефона и, если нужно, код оператора. Пользователь сообщает Терминалу запрошенные данные. Терминал удостоверяется, что данные введены корректно.

**? Требования бывают:**

* Прямые – прописаны в документах, юзер-сторях (банк-приложение: формат номера телефона по образцу)
* Косвенные – происходящие из прямых, и соответствующих здравому смыслу (банк-приложение: формат ввода суммы перевода не может быть буквенным – никто не будет вводить сумму прописью)

**? Критерии хороших требований**

* Выполнимость
* Корректность
* Недвусмысленность
* Полнота набора требований
* Непротиворечивость набора требований
* Упорядоченность по важности и стабильности
* Проверяемость (тестопригодность)
* Модифицируемость;
* Трассируемость
* Понимаемость

**? Требования к требованиям:**

* Единичность – Одно требование описывает одну и только одну вещь
* Завершённость – требование полностью определено в одном месте и вся необходимая инфа присутствует
* Последовательность – Требование не противоречит другим требованиям и полностью соответствует внешней документации (одна и та же кнопка «печать» приводит к разной цепочки действий в разных местах сайта + непротиворечивость).
* Атомарность – Требование должно быть «атомарным». То есть оно не может быть разбито на ряд более детальных требований без потери завершённости (схоже с первым пунктом).
* Отслеживаемость – Требование полностью или частично соответствует деловым нуждам как заявлено заинтересованными лицами и документировано (требование больше для Product Owner: можно ли проверить исполнение требования).
* Актуальность – Требование не стало устаревшим с течением времени (ПО обновлено, а требования нет (не до того) - устарело).
* Выполнимость – Требование может быть реализовано в пределах проекта.
* Недвусмысленность – Требование кратко определено без обращения к техническому жаргону, акронимам и другим скрытым формулировкам. Оно выражает объективные факты, а не субъективное мнение.
* Обязательность – Требование представляет определённую заинтересованным лицом х-ку, отсутствие которой приводит к неполноценности решения, которая не может быть проигнорирована (надо чтобы было реализовано так, как прописано в требовании, и не иначе; даже если иначе будет лучше, но по-другому).
* (Необязательное требование — противоречие самому понятию требования («хотелось бы», сделайте кнопку синей ИЛИ фиолетовой)).
* Проверяемость – Реализованность требования может быть определена через один из 4 возможных методов: осмотр, демонстрация, тест или анализ (релиз возможно подвергнуть машинной проверке)

**? Виды требований**

* Функциональные
* Нефункциональные
  + Требования к дизайну и юзабилити
  + Требования к безопасности и надёжности
  + Требования к производительности
  + Требования к локализации
  + Географические требования

**? Проверка требований**

* Тестирование
* Анализ
* Демонстрация

**? Ур-ни требований:**

* бизнес требования
* пользовательские требования
* функциональные (системные) требования

Как реагирует **система** на **действия пользователя,** который достигает **бизнес-целей**

**? Тест дизайн** — это **этап** процесса тестирования ПО, на котором **проектируются и создаются** тест кейсы, в соответствии с определёнными ранее **критериями качества** и **целями тестирования**.  
Роли, ответственные за тест дизайн:  
• Тест аналитик — определяет «ЧТО тестировать?»  
• Тест дизайнер — определяет «КАК тестировать?»

**? Для чего нужны Техники тест дизайна**

Техники тест дизайна – **рекомендации** для создания теста – чтобы найти баланс и выявить максимальное количество ошибок при необходимом минимуме тестовых сценариев. При этом **нужно проверить все наиболее важные кейсы**, поскольку время тестирования ограничено.

**? Техники тест-дизайна**

* Статическое – не требует запускать программу
  + Code Review (Вычитка исходного кода)
    - Неформальное Ревью (Informal Review)
    - Техническое ревью (Technical Review)
    - Прохождение (Workthrough)
    - Инспекция (Inspection)
  + Static Analysis (Проверка требований)
    - Поток данных (Data Flow Testing)
    - Поток управления (Control Flow Testing)
    - Путь (Path)
    - Стандарты (Standarts)
* Динамическое – надо запускать ПО
  + Чёрного ящика (Specification)
    - Эквивалентное разбиение (Equivalence Partitioning — EP)
    - Анализ Граничных значений (Boundary Value Analysis — BVA)
    - Таблица принятия решений (Decision Table)
    - Комбинации:
      * Все комбинации (All combi)
      * Попарное тестирование (Pairwise Testing)
      * Тестирование Каждого выбора (Each Choice testing)
      * Тестирование Базового выбора (Base Choice testing)
    - Метод Дерева Классификации (Classification Tree Method)
    - Причина-Следствие (Cause/Effect — CE)
    - Состояние переходов – модель перехода состояний (State Transition)
      * Состояния, которые может принимать программное обеспечение (открытые / закрытые или финансируемые / недостаточные фонды);
      * Переходы из одного состояния в другое (разрешены не все переходы);
      * Действия, которые вызывают переход (вывод денег, закрытие файла);
      * Действия, которые возникают в результате перехода (сообщение об ошибке или получение денежных средств).
    - Предугадывание ошибок (Error Guessing — EG)
    - Исчерпывающее тестирование (Exhaustive Testing — ET)
    - Тестирование Синтаксиса (Syntax testing) форма Бэкуса-Наура (BNF)
      * \b[\w.%+–]+@[\w.–]+\.[a–zA–Z]{2,6}\b
    - Случайное тестирование (Random testing) – генерация случайных входных данных
    - Пользовательские сценарии (Scenario Testing including Use Case Technique)
    - Доменное тестирование (Domain testing) – несколько переменных должны быть протестированы одновременно; предоставить стратегию минимального набора показательных тестов. Микс из Тестирования разделением, Анализа граничных значений, Классов эквивалентности
  + На опыте (Experience)
    - Исследовательское (Exploratory)
    - Ad-hoc
  + Белого ящика (Structure)
    - Покрытие выражений / Покрытие операторов (Statement coverage) (if, for, switch)
    - Покрытие решений (Decision coverage)
    - Покрытие ветвей (Branch coverage)
    - Покрытие условий (Condition coverage) – только логические операторы (and, or, xor)
    - Покрытие пути (Path coverage)
    - Мутационное тестирование
    - Тестирование Базового Пути

**Статическое и динамическое тестирование**  
Статическое тестирование отличается от динамического тем, что производится без запуска программного кода продукта. Тестирование осуществляется путем анализа программного кода (code review) или скомпилированного кода. Анализ может производиться как вручную, так и с помощью специальных инструментальных средств. Целью анализа является раннее выявление ошибок и потенциальных проблем в продукте. Также к статическому тестированию относится тестирования спецификации и прочей документации.

**Исследовательское и ad-hoc тестирование**  
Простейшее определение исследовательского тестирования — это разработка и выполнения тестов в одно и то же время. Что является противоположностью сценарного подхода (с его предопределенными процедурами тестирования, неважно ручными или автоматизированными). Исследовательские тесты, в отличие от сценарных тестов, не определены заранее и не выполняются в точном соответствии с планом.

Разница между ad hoc и exploratory testing в том, что теоретически, ad hoc может провести кто угодно, а для проведения exploratory необходимо мастерство и владение определенными техниками. Обратите внимание, что определенные техники это не только техники тестирования.

**Мутационное тестирование** – белого ящика – делаются небольшие изменения кода, если тест-кейсы не находят эти изменения и проходят успешно – то эти тест-кейсы уже не годятся

**Тестирование** базового пути является одним из методов «белого ящика» и гарантирует выполнение хотя бы одного оператора во время тестирования. Он проверяет каждый линейно независимый путь в программе, что означает, что число тестовых примеров будет эквивалентно цикломатической сложности программы.

**? Цикломатическая сложность**

Это метрика ПО, используемая для измерения сложности программы. Это **количественная** мера **независимых путей** в исходном коде программы. Независимый путь определяется как путь, имеющий хотя бы одно ребро, которое ранее не проходило ни в одном другом пути. Цикломатическая сложность может быть рассчитана относительно функций, модулей, методов или классов в программе. Эта метрика основана на представлении программы как control flow. Поток управления изображает программу в виде графа, который состоит из вершин и ребер. На графе вершины представляют обработку задачи, а ребра представляют поток управления между вершинами.

Тестирование базового пути является одним из методов «белого ящика» и гарантирует выполнение хотя бы одного оператора во время тестирования. Он проверяет каждый линейно независимый путь в программе, что означает, что число тестовых примеров будет эквивалентно цикломатической сложности программы.

M = E – N + 2 x P (компонент связности – некоторое множество вершин графа такое, что для любых двух вершин из этого множества существует путь из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества)

M = 5 – 5 + 2 x 1 = 2 OR

M = ◊ (if) + 1 = 1 + 1 = 2

Сложность 1-10 говорит о хорошо написанном коде, легкой тестируемости и экономичности.

Сложность 10-20 и 20-30 говорят об усложненном коде и трудностях с тестированием такого кода вкупе с высокими затратами.

Сложность больше 40 практически не поддается проверке и стоит очень дорого.

**? В чем отличие build от release**

Билд это номер, даваемый ПО при передаче от разработчиков тестировщикам. Релиз — это номер, даваемый ПО при передаче конечному пользователю.

**? Что такое бизнес-логика (domain)**

Бизнес-логика (domain) это то, что конкретная программа по задумке должна сделать. Например, в складской программе проверка на возможность отправить товар (вдруг его нет в наличии). Это правила, которые должны соблюдаться в данной конкретной программе, определенные бизнес-клиентом. Слои приложения – слой пользовательского интерфейса, слой бизнес логики, слой сохранения данных.

**Уровни и Виды Тестирования**

? Виды тестирования:

1. функциональное
2. не функциональное
3. связанное с изменениями

? Регрессии требуют

* Переход на новую версию
* Введение какой-то фичи
* Когда пофиксили ч.л.
* И т.д.

? Ур-ни тестирования

* Модульное / Компонентное / Юнит
* Интеграционное
  + Тестирование интеграции модулей / компонентов
    - Bottom-Up Approach – Подход Сверху-вниз – (драйвер – вызывает тестируемый модуль)
    - Top-Down Approach – Подход Снизу-вверх (заглушка – вызывается тестируемым модулем)
    - Sandwich Approach – Подход с двух сторон
    - Big-Bang Approach – Подход Большого Взрыва
  + Системное Интеграционное тестирование
    - API – Application Program Interface
    - CLI – Command Line Interface
    - GUI – Graphic User Interface
* Системное
* Приёмочное
  + UAT Пользовательское приёмочное тестирование
  + Эксплуатационное
  + Тестирование на соответствие контракту (спеке или к.л. ГОСТу, акту, GDPR – защита перс данных)
  + Альфа (силами штатных работников)
    - Pre-Alpha: - ПО является прототипом. Пользовательский интерфейс завершен. Но не все функции завершены. На данном этапе ПО не публикуется.
    - Alpha: является ранней версией программного продукта.
  + Бета (привлечение сторонних)
    - Beta: ПО стабильно и выпускается для ограниченной пользовательской базы. Цель состоит в том, чтобы получить отзывы клиентов о продукте и внести соответствующие изменения в ПО.
      * Закрытое Beta: группа бета-тестировщиков из сторонней компании
      * Открытое Beta: когда все желающие могут стать бета-тестировщиками.
  + Release Candidate (RC): основываясь на отзывах Beta Test, вы вносите изменения в ПО и хотите проверить исправления ошибок. На этом этапе вы не хотите вносить радикальные изменения в функциональность, а просто проверяете наличие ошибок. RC также выпущен для общественности
  + Release: Все работает, ПО выпущено для общественности.

? Виды тестирования **по степени важности**

* Дымовое – приложение выполняет свои основные ф-ции
* Тест критического пути – значимые эл-ты и ф-ции проверяются на предмет правильности при стандартном использовании – выполнение типичных задач, выполняемые пользователям
* Расширенный тест – проверка нестандартного использования продукта

Виды тестирования **по цели тестирования**

* New Feature Test
* Regression Testing
  + В каждом новом билде
  + Исправление багов
  + Проверка связанных с функциональностью
* Re-test – проверка исправности дефекта

Виды тестирования **по степени автоматизации**

* Ручное
* Автоматизированное 🡪 основные претенденты: Смоук и Регрессия

Виды тестирования **по знанию кода / Методы тестирования:**

* Чёрного ящика (Тестирование поведения / Тестирование, основанное на документации) 🡪 GUI
* Серого ящика 🡪 API, DB, HTML/CSS
* Белого ящика 🡪 Source Code (Исходный код)

? Виды/Типы тестирования

**Функциональные виды тестирования («Что?»)**

• Функциональное тестирование (Functional testing)  
• Тестирование взаимодействия (Interoperability Testing)

**Нефункциональные виды тестирования («Как?»)**

• Все виды тестирования производительности:  
 ◦ нагрузочное тестирование (Performance and Load Testing)  
 ◦ стрессовое тестирование (Stress Testing)  
 ◦ тестирование стабильности или надежности (Stability / Reliability Testing)  
 ◦ объемное тестирование (Volume Testing)  
• Установки (Installation testing)

• Отказ и восстановление (Failover and Recovery Testing)

• Конфигурационное (Configuration Testing)

◦ Кросс-платформенное

◦ Кросс-браузерное

• UI Пользовательского интерфейса (GUI Testing)

• UX Удобства пользования (Usability Testing)

• Безопасности (Security and Access Control Testing)

• Локализации/Интернационализации

**Связанные с изменениями виды тестирования**

• Дымовое тестирование (Smoke Testing)  
• Регрессионное тестирование (Regression Testing)  
• Повторное тестирование (Re-testing)  
• Тестирование сборки (Build Verification Test)  
• Санитарное тестирование или проверка согласованности/исправности (Sanity Testing)

**ФУНКЦИОНАЛЬНЫЕ ВИДЫ ТЕСТИРОВАНИЯ («ЧТО?»)**

**? Функциональное тестирование** рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

**? Тестирование взаимодействия (Interoperability Testing)** — это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости (compatibility testing) и интеграционное тестирование

**НЕФУНКЦИОНАЛЬНЫЕ ВИДЫ ТЕСТИРОВАНИЯ («КАК?»)**

**? Нагрузочное тестирование** — это автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

**? Стрессовое тестирование (Stress Testing)** позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

**? Объемное тестирование (Volume Testing).** Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения

**? Тестирование стабильности или надежности (Stability / Reliability Testing).** Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.

**? Тестирование установки** направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения.

**? Тестирование пользовательского интерфейса (GUI Testing)** — функциональная проверка интерфейса на соответствие требованиям — размер, шрифт, цвет, consistent behavior.

**? Тестирование удобства пользования** — это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. Сюда также входит:  
User eXperience (UX) — ощущение, испытываемое пользователем во время использования цифрового продукта, в то время как User interface — это инструмент, позволяющий осуществлять интеракцию «пользователь — веб-ресурс».

**? Тестирование на отказ и восстановление (Failover and Recovery Testing)** проверяет тестируемый продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи (например, отказ сети). Целью данного вида тестирования является проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

**? Конфигурационное тестирование (Configuration Testing)** — специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.)

**? Тестирование совместимости** (Compatibility testing)

вид нефункционального тестирования, основной целью которого является проверка корректной работы продукта в определенном окружении. Окружение может включать в себя следующие элементы:  
Аппаратная платформа;  
Сетевые устройства;  
Периферия (принтеры, CD/DVD-приводы, веб-камеры и пр.);  
Операционная система (Unix, Windows, MacOS, ...)  
Базы данных (Oracle, MS SQL, MySQL, ...)  
Системное программное обеспечение (веб-сервер, файрволл, антивирус, ...)  
Браузеры (Internet Explorer, Firefox, Opera, Chrome, Safari)

**? Тестирование безопасности** — это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

**СВЯЗАННЫЕ С ИЗМЕНЕНИЯМИ ВИДЫ ТЕСТИРОВАНИЯ**

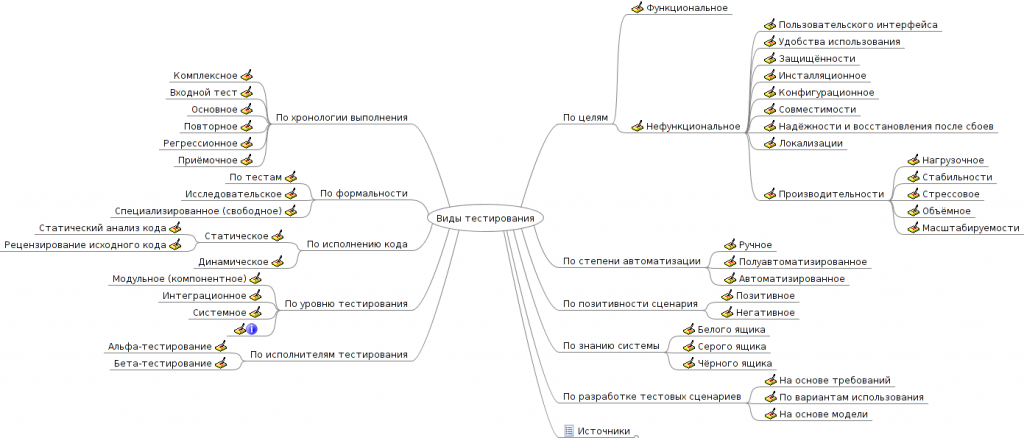
**? Дымовое (Smoke) тестирование** рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции.

**? Тестирование сборки или Build Verification Test** — тестирование направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования. По своим целям является аналогом Дымового Тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

**? Санитарное тестирование** — это узконаправленное тестирование достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям. Является подмножеством регрессионного тестирования. Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде. Обычно выполняется вручную.

**? Повторное тестирование** — тестирование, во время которого исполняются тестовые сценарии, выявившие ошибки во время последнего запуска, для подтверждения успешности исправления этих ошибок.

**? Регрессионное тестирование** — это вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде. Регрессионными могут быть как функциональные, так и нефункциональные тесты.



**? В чем разница между regression testing и re-testing**  
Re-testing — проверяется исправление багов  
Regression testing — проверяется то, что исправление багов, а также любые изменения в коде приложения, не повлияли на другие модули ПО и не вызвало новых багов.

**? Critical path test** **(Тестирование критического пути)**

Основной тип тестовых испытаний, во время которого значимые элементы и функции приложения проверяются на предмет правильности работы при стандартном их использовании.

Критикал-пасс – как расширенный смоук + изменение + удаление товара

? Повторное тестирование = ре-тест

? Тестирование, связанное с изменениями:

* Смоук
* Санити – какая-то определённая фича
* Ре-тест
* Регрессия

**? SDLC – Жизненный цикл разработки ПО**

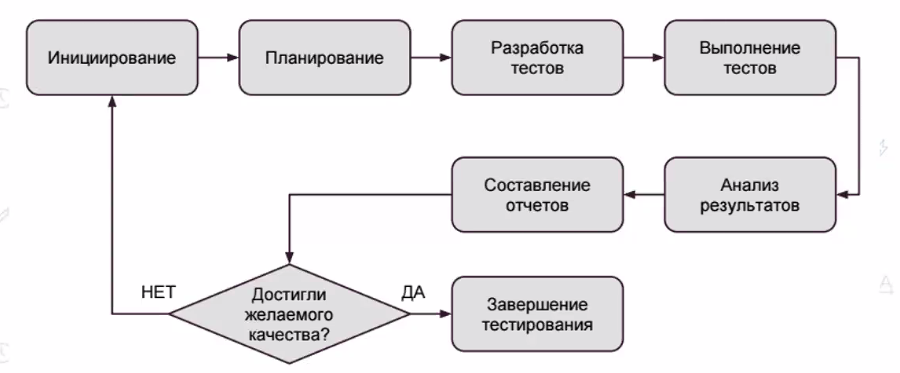
* *(Idea)*
* Planning / Requirement / Сбор и анализ требований
* Analysis / Анализ – док-т Спецификация требований к программному обеспечению (SRS - Software Requirement Specification).
* Design / Проектирование
* Development / Разработка – пишем код
* Testing / Тестирование
* Deployment & Maintenance / Развёртывание и обслуживания

**? STLC - Жизненный цикл тестирования ПО**

* Requirement analysis
* Test Planning
* Test case Development
* Environment setup
* Test execution
* Test cycle closure

**? Этапы тестирования:**  
Модель STLC устанавливает следующие этапы:

* Инициация (Новая версия ПО **ИЛИ** Запрос на тестирование от заказчика **ИЛИ** запрос от менеджера)
* Выявление и анализ требований прямых и косвенных
* Планирование испытаний
* Генерация Test case,
* + Отбор показательных Test case,
* Настройка среды
* Проведение проверок,
* + Фиксация результатов,
* + Анализ результатов
* Передача информации о соответствии проверенного продукта требованиям.
* Подготовка - На этом этапе QA-инженер читает проектную документацию, выясняет требования к продукту, прорабатывает план, продумывает стратегию, расставляет задачи по приоритетности и анализирует возможные риски.
* Тестирование - Предварительно специалисты анализируют собранную ранее информацию, составляют список тестируемых функций, знакомятся с уже известными багами, если они есть, пишут тест-кейсы.
* Анализ результатов и составление отчётов - При работе над созданием тестов QA-специалист ориентируется не только на документацию, но и на устные сведения от других QA, аналитиков, разработчиков, менеджеров проекта.



**? Эвристики окончания тестирования**

1. Эвристика «Время вышло!».

2. Эвристика [пиньяты](http://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BD%D1%8C%D1%8F%D1%82%D0%B0) (The Piñata Heuristic).

3. Эвристика «мертвой лошади» (The Dead Horse Heuristic).

4. Эвристика «Задание выполнено» (The Mission Accomplished Heuristic).

5. Эвристика «Отмена задания» (The Mission Revoked Heuristic).

6. Эвристика «Я зашел в тупик!» (The I Feel Stuck! Heuristic).

7. Эвристика «освежающей паузы» (The Pause That Refreshes Heuristic).

8. Эвристика «Отсутствие продвижения» (The Flatline Heuristic).

9. Эвристика Привычного завершения (The Customary Conclusion Heuristic).

10. Больше нет интересных вопросов (No more interesting questions).

11. Эвристика уклонения/безразличия (The Avoidance/Indifference Heuristic).

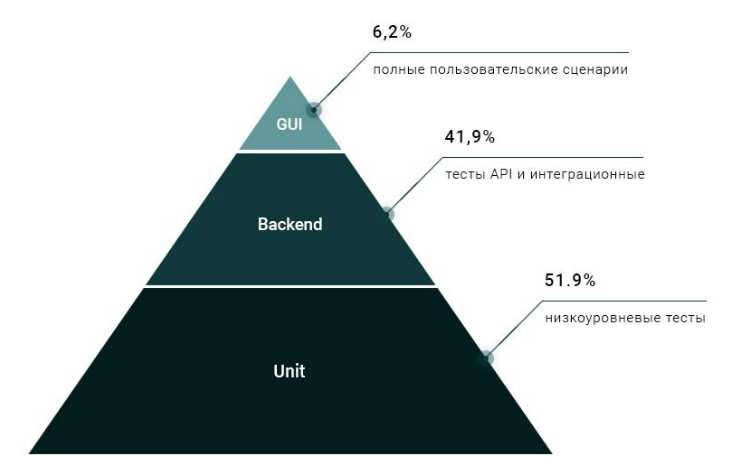
? **Тестирование на основе рисков (RBT)** — **это** тип **тестирования**, основанный на вероятности **риска**. Она включает в себя оценку **риска**, в зависимости от сложности, бизнес критичности, частоты использования, видимых областях, Дефектные районах.

? **UAT** – **User Acceptance Testing** **(Приемочное пользовательское тестирование)** – тестирование, которое проводится конечными пользователями системы с целью принятия решения о внедрении. Любая разработка или доработка программного обеспечения проходит заключительную стадию UAT-тестирования.

? Интернационализация – то, как на разных языках будет выглядеть вёрстка, как сервис будет работать с разными форматами даты

? Локализация – культурное влияние

? пирамида автоматизации



**PICT**

? **Попарное тестирование** (**Pairwise Testing**) — это разработка тестов методом черного ящика, в которой тестовые сценарии разрабатываются таким образом, чтобы выполнить все возможные отдельные комбинации каждой пары входных параметров.

**Pairwise testing** — техника тест-дизайна, а именно метод обнаружения дефектов с использованием комбинационного метода из двух тестовых случаев. Он основан на наблюдениях о том, что большинство дефектов вызвано взаимодействием не более двух факторов (дефекты, которые возникают при взаимодействии трех и более факторов, как правило менее критичны). Следовательно, выбирается пара двух тестовых параметров, и все возможные пары этих двух параметров отправляются в качестве входных параметров для тестирования. Pairwise testing сокращает общее количество тест-кейсов, тем самым уменьшая время и расходы, затраченные на тестирование.

**? Тестирование с помощью алгоритма All-Pairs**

Аll-pairs testing — комбинаторный метод тестирование программного обеспечения, который проверяет все возможные дискретные комбинации параметров для каждой пары входных параметров системы. Исходя из этого, мы получим меньшее число комбинаций, чем при использовании ортогональных матриц.

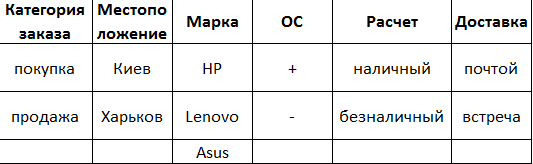
**? Пример**

Предположим, нам необходимо протестировать приложение для покупки/продажи б/у ноутбуков, мы имеем следующие переменные:

* категория заказа: покупка, продажа;
* местоположение: Киев, Харьков;
* марка ноутбука: HP, Lenovo, Asus;
* ОС: доступна, недоступна;
* тип расчета: наличный, безналичный;
* тип доставки: почтой, встреча.

Если мы захотим протестировать все возможные комбинации, то мы должны составить 2 х 2 х 3 х 2 х 2 х 2 = 96 тест-кейса. Не многовато ли работы для тестирования формы?

Далее нам необходимо организовать переменные и значения.



Чтобы начать заполнять таблицу, необходимо организовать столбцы таким образом, чтобы первый имел наиболее большое количество переменных, а последний - наименее. Таким образом, первый столбец в нашей таблице — марка ноутбука. Первым делом записываем три значения Марки (т.к. это столбец с наибольшим числом значений) по два раза( два — это количество переменных следующего столбца, например, категория заказа). Это имеет такой вид:



Т.е. для каждого набора в столбце 1 мы помещаем оба значения столбца 2. То же самое мы повторяем с 3 столбцом.



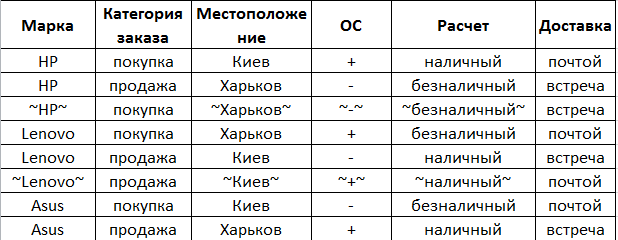
У нас есть комбинация покупка&Киев и продажа&Харьков, но нету комбинации продажа&Киев и покупка&Харьков. Исправим это, поменяв местами значения во втором наборе третьего столбца.



Повторяем такие же манипуляции для колонок 4 и 5.



Колонка Доставка является более проблематичной, ведь нам не хватает комбинаций на покупка&встреча и продажа&почтой чтобы не нарушать отсортированные данные, нужно ввести еще 2 тестовых случая для этих комбинаций. Значком тильды “~” мы маркируем переменные, которые выступают произвольными. Таким образом мы получаем следующую таблицу.



Таким образом, мы получили готовые 8 тест-кейсов вместо 96.

**? Утилиты для автоматизации pairwise testing**

* PICT — ‘Pairwise Independent Combinatorial Testing’, provided by Microsoft Corp.
* IBM FoCuS — ‘Functional Coverage Unified Solution’, provided by IBM.
* ACTS — ‘Advanced Combinatorial Testing System’, provided by NIST, an agency of the US Government.
* Hexawise
* Jenny
* Pairwise by Inductive AS
* VPTag free All-Pair Testing Tool.

**? PICT**

* (Создать таблицу в Exсel с параметрами и значениями)
* Создать текстовый файл File.txt в формате:

Parameter1: aaaaaaa, bbbbbb, cccccccccc, ddddddd

Parameter2: mmmmm, nnnnnn, oooooo

Parameter3: xxxxxxx, yyyyyyy, zzzzzzzzz

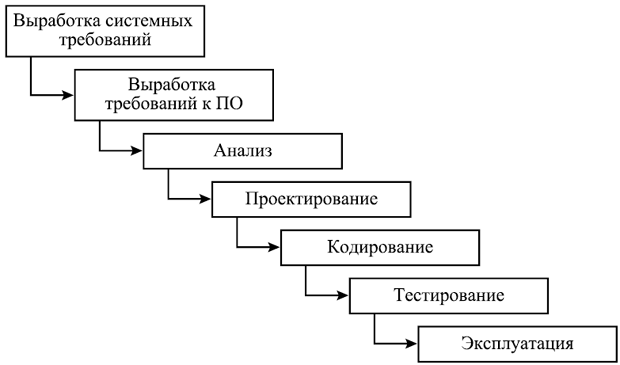
* Копировать файл File.txt на диск C:/ в папку с PICT
* Запустить CLI
* C:\Program Files\PICT> pict "C:\Program Files\PICT\File.txt" > File.xls
* В этой же папке PICT создастся файл File.xls с результатами Pairwise

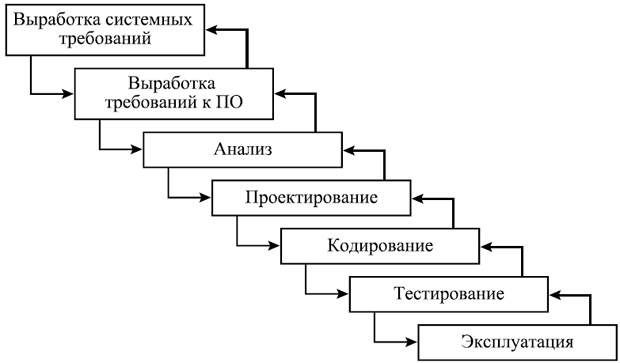
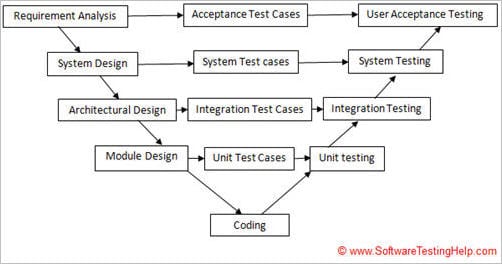
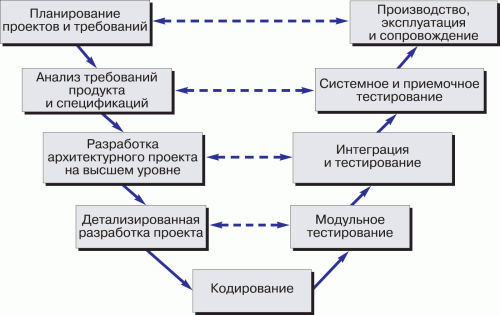
**Методологии в разработке ПО**

? Методологии в разработке ПО:

1. Классические

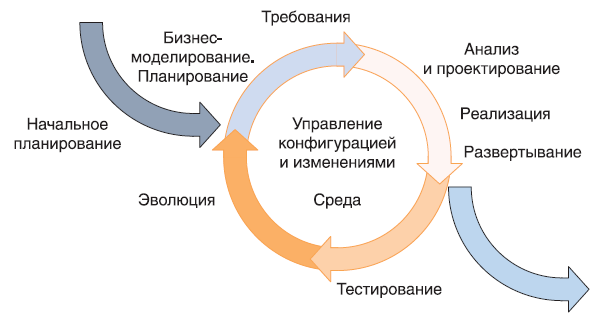
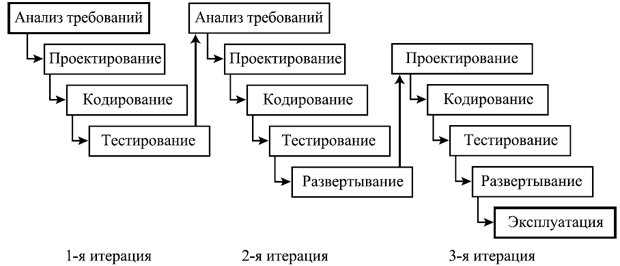
* **«Водопадная» /Каскадная модель –** В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая. Процесс создания программного обеспечения представляет собой поток, последовательно проходящий все фазы. Так обычно строится работа над крупными проектами с длительным сроком внедрения.  
  — высокий уровень формализации процессов;  
  — большое количество документации;  
  — жесткая последовательность этапов жизненного цикла без возможности возврата на предыдущий этап.
  + Watrefall проект должен постоянно иметь актуальную документацию. Обязательная актуализация проектной документации. Избыточная документация.
  + Очень не гибкая методологии
  + Простои команды
  + Может создать ошибочное впечатление о работе над проектом (например фраза «45% выполнено» не несёт за собой никакой полезной информации, а является всего лишь инструментов для менеджера проекта)
  + У Заказчика нет возможности ознакомиться с системой заранее и даже с «Пилотом» системы
  + У Пользователя нет возможности привыкать к продукту постепенно
  + Все требования должны быть известны в начале жизненного цикла проекта
  + Возникает необходимость в жёстком управлении и регулярном контроле, иначе проект быстро выйдет из графиков
  + Отсутствует возможность учесть переделку, весь проект делается за один раз
  + Высокая прозрачность разработки и фаз проекта
  + Чёткая последовательность
  + Стабильность требований
  + Строгий контроль менеджмента проекта
  + Облегчает работу по составлению плана проекта и сбора команды проекта
  + Хорошо определяет процедуру по контролю качества
  + Точная регламентация желаемого функционала на входе позволяет сразу рассчитать стоимость воплощения
  + Если всё делать правильно, «водопад» будет наиболее быстрой и простой моделью.



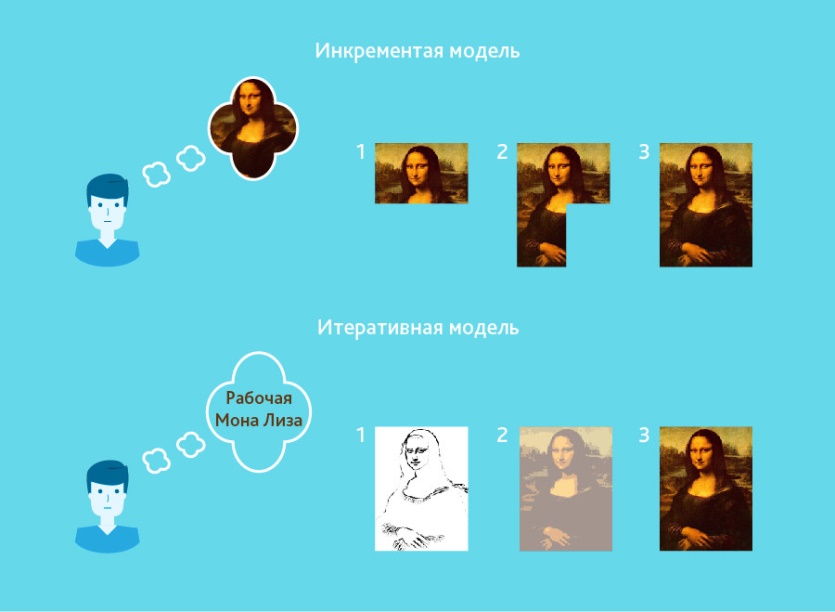
* + **«Водоворотная» модель или Каскадная модель с промежуточным контролем.** В этой модели предусмотрен промежуточный контроль за счет обратных связей. Но это достоинство порождает и недостатки. Затраты на реализацию проекта при таком подходе возрастают практически в 10 раз. Эта модель, как вы уже поняли, является незначительной модификацией предыдущей и относится к первой группе.  
      
    - При реальной работе в соответствии с моделью, допускающей движение только в одну сторону, обычно возникают проблемы при обнаружении недоработок и ошибок, сделанных на ранних этапах.
    - Но еще более тяжело иметь дело с изменениями окружения, в котором разрабатывается ПО (это могут быть изменения требований, смена подрядчиков, изменения политик разрабатывающей или эксплуатирующей организации, изменения отраслевых стандартов, появление конкурирующих продуктов и пр.).
  + **V-model -** Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе.  
    Модель предназначена для проектирования систем, которым критично важно бесперебойное функционирование. Повышенное внимание и выделение ресурсов на тестирование на всех этапах проекта. Крайне жручая к квалифицированным тестировщикам и так же обладает малой гибкостью и большой стоимостью внесения изменений т.к. работы ведутся по четкому ТЗ, составляемому в начале проекта  
    Данная модель имеет более приближенный к современным методам алгоритм, однако все еще имеет ряд недостатков. Является одной из основных практик экстремального программирования и предполагает регулярное тестирование продукта во время разработки.  
      
    

V-модель обеспечивает поддержку в планировании и реализации проекта. В ходе проекта ставятся следующие задачи:

* + - Минимизация рисков: V-образная модель делает проект более прозрачным и повышает качество контроля проекта путём стандартизации промежуточных целей и описания соответствующих им результатов и ответственных лиц. Это позволяет выявлять отклонения в проекте и риски на ранних стадиях и улучшает качество управления проектов, уменьшая риски.
    - Повышение и гарантии качества: V-Model — стандартизованная модель разработки, что позволяет добиться от проекта результатов желаемого качества. Промежуточные результаты могут быть проверены на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.
    - Уменьшение общей стоимости проекта: Ресурсы на разработку, производство, управление и поддержку могут быть заранее просчитаны и проконтролированы. Получаемые результаты также универсальны и легко прогнозируются. Это уменьшает затраты на последующие стадии и проекты.
    - Повышение качества коммуникации между участниками проекта: Универсальное описание всех элементов и условий облегчает взаимопонимание всех участников проекта. Таким образом, уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком.
* **Incremental model (Инкрементальная модель)** - Это модель разработки по частям - Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание.  
  Модель нацелена на быстрое представление прототипа и постепенное наращивание функционала на его "костяке". Цель каждой итерации - получение конечной версии программы, вобравшую функционал предыдущих и текущего отрезка. Такая "эволюция" прототипа продолжается до финальной итерации, после которой и появится готовый продукт. Модель предполагает возможность перехода между этапами с незавершенными задачами, которые будут исправлены при следующей итерации

**Итеративная или инкрементная (эволюционная) модель** приращения продукта позволяет параллельно выполнять ряд задач с непрерывным анализом результатов и корректировкой предыдущих этапов работы. Это более «скоростная» разработка для большого штата квалифицированных программистов.  
**Итеративные или инкрементальные модели (известно несколько таких моделей)** предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части.  
Каскадная модель с возможностью возвращения на предшествующий шаг при необходимости пересмотреть его результаты, становится итеративной.  
Итеративный процесс предполагает, что разные виды деятельности не привязаны намертво к определенным этапам разработки, а выполняются по мере необходимости, иногда повторяются, до тех пор, пока не будет получен нужный результат.  
Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.  


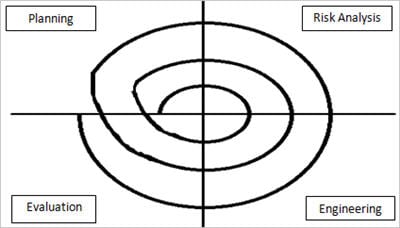
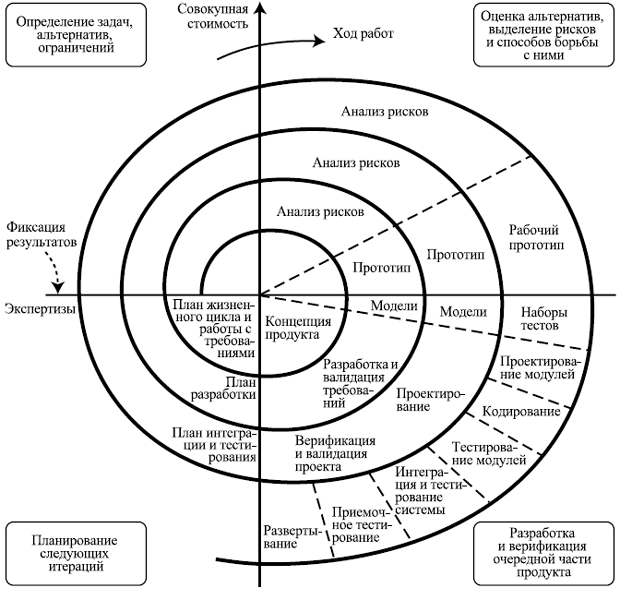
**Iterative model (Итеративная модель)** -

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.  
  
На диаграмме показана итерационная «разработка» Мона Лизы. Как видно, в первой итерации есть лишь набросок Джоконды, во второй — появляются цвета, а третья итерация добавляет деталей, насыщенности и завершает процесс. В инкрементной же модели функционал продукта наращивается по кусочкам, продукт составляется из частей. В отличие от итерационной модели, каждый кусочек представляет собой целостный элемент.

? MVP – minimum value product – в инкрементальной модели

«Минимально жизнеспособный продукт — это версия нового продукта, которую используют, чтобы собрать максимальное количество подтвержденных гипотез о клиентах с минимальными усилиями»

* + **Spiral model (Спиральная модель)** — Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект.   
    **Спиральная методика** характеризуется прохождением проектом повторяющегося цикла в каждой фазе развития: планирование — реализация — проверка — оценка (plan-do-check-act cycle). Так обычно создаются проекты, с окончательно не сформированным видением результата, либо требующие ультрасрочного внедрения по этапам.



* + Результат достигается в кратчайшие сроки.
  + Конкурентоспособность достаточно высокая.
  + При изменении требований, не придется начинать все с «нуля».
  + Невозможность регламентирования стадий выполнения.
* **RAD (быстрая модель разработки)** – Вариация итеративной модели. Цель - как можно быстрее выпустить готовый продукт. На этапе начального планирования предполагается оценка сложности продукта в ***функциональных элементах***(экраны, сообщения, отчеты, файлы и т.п.) После, общий пул функциональных элементов, разбивается на задачи, выполнять которые предполагается одновременно несколькими небольшими командами специалистов. В финале компоненты от каждой команды собираются в требуемое решение. Для реализации подобной модели требуется большое количество высококвалифицированных спецов и людей, способных обеспечивать коммуникацию между командами. Соответственно увеличивается и бюджет.  
    
  **Модель быстрой разработки приложений включает следующие фазы:**
* Бизнес-моделирование: определение списка информационных потоков между различными подразделениями.
* Моделирование данных: информация, собранная на предыдущем этапе, используется для определения объектов и иных сущностей, необходимых для циркуляции информации.
* Моделирование процесса: информационные потоки связывают объекты для достижения целей разработки.
* Сборка приложения: используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.
* Тестирование: тестируются новые компоненты и интерфейсы.

◦ Может использоваться только при наличии высококвалифицированных и узкоспециализированных архитекторов.

◦ Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки.

◦ RAD-модель может быть выбрана, при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2-3 месяцев

1. Гибкие – Agile – Включает в себя практики, подходы и методологии, которые помогают создавать продукт более эффективно:
   * Agile Manifesto:
     + **Люди и взаимодействие** важнее процессов и инструментов.
     + **Работающий продукт** важнее исчерпывающей документации.
     + **Сотрудничество с заказчиком** важнее согласования условий контракта.
     + **Готовность к изменениям** важнее следования первоначальному плану.
   * **Lean** – **Бережливую разработку программного обеспечения (Lean);**  
     Lean – концепция, основными элементами которой являются value (как ценность, в первую очередь в глазах клиента), waste (как потери, что-то, что не приносит ценности или тратит излишние ресурсы) и flow (как процесс предоставления продукта от получения заказа до доставки клиенту). Основные постулаты:
     + Смотри на весь процесс в целом,
     + Привноси ценность,
     + Отрезай потери
   * **KANBAN** – **Метод управления разработкой KANBAN** – Команда ведёт работу с помощью виртуальной доски, которая разбита на этапы проекта. Каждый участник видит, какие задачи находятся в работе, какие — застряли на одном из этапов, а какие уже дошли до его столбца и требуют внимания.   
     Весь КАНБАН можно описать одной простой фразой — «Уменьшение выполняющейся в данный момент работы (work in progress)»
   * **SCRUM** – **Фреймворк для управления проектами SCRUM** – это методология управления проектами, которая построена на принципах тайм-менеджмента. Основной ее особенностью является вовлеченность в процесс всех участников, причем у каждого участника есть своя определенная роль. Суть в том, что не только команда работает над решением задачи, но все те, кому интересно решение задачи, не просто поставили ее и расслабились, а постоянно «работают» с командой, и эта работа не означает только постоянный контроль.

* **XP** – **Экстремальное программирование (Extreme Programming, XP);**  
  XP (Экстремальное программирование) – гибкая методология разработки программного обеспечения. В основе лежит 13 практик, которые подразумевают общее владение кодом (исправлять может каждый участник команды и любой участок), жесткую стандартизацию кода (если все могут исправлять, необходимо сохранить читаемость), постоянное получение обратной связи от заказчика и забетонированный финальный срок. В случае, если проект не укладывается в срок, то дату релиза не трогают, но режут функционал. А еще есть забавная практика "парного программирования". Сколько нужно программистов, чтобы закодить 1 функционал? Если серьезно, то такая методика должна позволять сразу выбирать наилучшее решение и проводить код-ревью одновременно с написанием.
* **RUP** – **Методология разработки программного обеспечения, созданная компанией Rational Software (RUP)** Использование методологии RUP направлено на итеративную модель разработки. Особенность методологии состоит в том, что степень формализации может меняться в зависимости от потребностей проекта. Можно по окончании каждого этапа и каждой итерации создавать все требуемые документы и достигнуть максимального уровня формализации, а можно создавать только необходимые для работы документы, вплоть до полного их отсутствия. За счет такого подхода к формализации процессов методология является достаточно гибкой и широко популярной. Данная методология применима как в небольших и быстрых проектах, где за счет отсутствия формализации требуется сократить время выполнения проекта и расходы, так и в больших и сложных проектах, где требуется высокий уровень формализма, например, с целью дальнейшей сертификации продукта. Это преимущество дает возможность использовать одну и ту же команду разработчиков для реализации различных по объему и требованиям. В основе методологии лежат 6 основных принципов:
  + компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта;
  + работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам;
  + ранняя идентификация и непрерывное устранение возможных рисков;
  + концентрация на выполнении требований заказчиков к исполняемой программе;
  + ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки;
  + постоянное обеспечение качества на всех этапах разработки проекта.
* разработку, управляемую функциональностью (Feature-driven development, FDD);
* разработку через тестирование (Test-driven development, TDD);
* методологию «чистой комнаты» (Cleanroom Software Engineering);
* итеративно-инкрементальный метод разработки (OpenUP);
* **MSF** – **Методологию разработки Microsoft Solutions Framework (MSF)**;  
  — методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.  
  Базовые концепции и принципы модели процессов MSF:
  + - единое видение проекта — все заинтересованные лица и просто участники проекта должны чётко представлять конечный результат, всем должна быть понятна цель проекта;
    - управление компромиссами — поиск компромиссов между ресурсами проекта, календарным графиком и реализуемыми возможностями;
    - гибкость — готовность к изменяющимся проектным условиям;
    - концентрация на бизнес-приоритетах — сосредоточенность на той отдаче и выгоде, которую ожидает получить потребитель решения;
    - поощрение свободного общения внутри проекта;
    - создание базовых версии — фиксация состояния любого проектного артефакта, в том числе программного кода, плана проекта, руководства пользователя, настройки серверов и последующее эффективное управление изменениями, аналитика проекта.  
      MSF предлагает проверенные методики для планирования, проектирования, разработки и внедрения успешных IT-решений. Благодаря своей гибкости, масштабируемости и отсутствию жестких инструкций MSF способен удовлетворить нужды организации или проектной группы любого размера. Методология MSF состоит из принципов, моделей и дисциплин по управлению персоналом, процессами, технологическими элементами и связанными со всеми этими факторами вопросами, характерными для большинства проектов.
* метод разработки динамических систем (Dynamic Systems Development Method, DSDM);

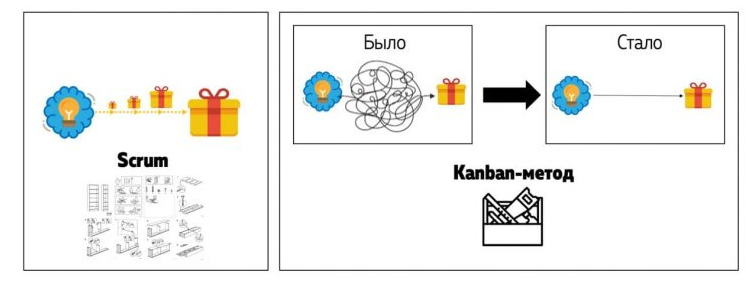
**? Разница** [**SCRUM**](https://www.atlassian.com/ru/agile/scrum) **и** [**KANBAN**](https://www.atlassian.com/ru/agile/kanban) –

* в КАНБАН нет тайм боксов ни на что (ни на задачи, ни на спринты)
* в КАНБАН задачи больше и их меньше
* в КАНБАН **оценки** сроков на задачу опциональные или вообще их нет
* в КАНБАН «скорость работы команды» отсутствует и считается только среднее время на полную реализацию задачи.

[**KANBAN**](https://www.atlassian.com/ru/agile/kanban) разработка отличается от [**SCRUM**](https://www.atlassian.com/ru/agile/scrum) в первую очередь ориентацией на задачи. Если в SCRUM основная ориентация команды — это успешное выполнение спринтов (надо признать, что это так), то в KANBAN на первом месте задачи.

Суть [**KANBAN**](https://www.atlassian.com/ru/agile/kanban) в **визуализации** работы, **ограничении объема** [незавершенной работы](https://www.atlassian.com/ru/agile/kanban/wip-limits) и достижении **максимальной эффективности** (или скорости). KANBAN-команды стремятся максимально сократить время, которое уходит на выполнение проекта (или [пользовательской истории](https://www.atlassian.com/ru/agile/project-management/user-stories)) от начала до конца. Для этого они используют [доску KANBAN](https://www.atlassian.com/ru/agile/kanban/boards) и непрерывно совершенствуют свой рабочий процесс.  
  
Задача команд [**SCRUM**](https://www.atlassian.com/ru/agile/scrum) — поставить **работающее ПО** за ряд **промежутков времени**, которые называются спринтами. Они стремятся **создавать циклы обучения** для быстрого сбора и учета отзывов клиентов. SCRUM-команды используют особые роли, создают специальные артефакты и проводят регулярные собрания, чтобы работа шла в нужном русле. Лучше всего методика SCRUM описана в [руководстве по SCRUM](https://www.scrumguides.org/scrum-guide.html).

|  |  |  |
| --- | --- | --- |
|  | **SCRUM** | **KANBAN** |
| График | Регулярные спринты с фиксированной продолжительностью (н-р, 2 недели) | Непрерывный процесс |
| Подходы к релизу | В конце каждого спринта | Непрерывная поставка |
| Роли | Владелец продукта, scrum-мастер, команда разработчиков | Обязательных ролей нет |
| Ключевые показатели | Скорость | Время выполнения, время цикла, объем незавершенной работы (WIP) |
| Отношение к изменениям | В ходе спринта команды не должны вносить изменения. | Изменение может произойти в любой момент |



[**SCRUM**](https://scrumtrek.ru/blog/agile-scrum/scrum-glossary/3652/scrum/) — это готовое руководство к тому, как организовать итеративно-инкрементальную разработку нового продукта. Есть даже инструкция, которая называется Scrum Guide. Все элементы Scrum взаимосвязаны друг с другом, и поэтому при реализации Scrum нельзя выбросить ничего из того, что указано в Scrum Guide.

[**KANBAN**](https://scrumtrek.ru/blog/kanban/1360/chto-takoe-kanban-metod-maksimalno-korotko/) похож на ящик с инструментами. Вы можете брать только что-то одно, или всё сразу — решать вам. Каждый инструмент приносит свою пользу. Выбор инструмента зависит лишь от вашей готовности его применять. Есть разные степени зрелости применения Канбан-метода, и на каждом уровне организация использует те или иные его элементы.

**? Приоритеты задач в** **SCRUM** – определяет Product Owner

**? SCRUM Команда**

* Product Owner
  + Управление бэклогом продукта
  + Описание бэклога
* SCRUM Master
  + Функционирование SCRUM
  + Обучение и понимание SCRUM
* Development team
  + Самоорганизация
  + Кроссфункциональность
  + Единственная роль – разработчик
  + Коллективная ответственность

**? SCRUM События**

* Спринт
  + Sprint Planning – Планирование Спринта
  + Daily SCRUM – Ежедневный Скрам
    - Что ты делал вчера?
    - Что ты будешь делать сегодня?
    - Какие были препятствия?
  + Sprint Review – Обзор спринта (тех часть) – задачи, кот. выполнили, демо инкремента
  + Sprint Retrospective – Ретроспектива спринта (перс часть членов команды)
    - Что хорошо шло во время спринта?
    - Какие были проблемы в спринте?
    - Как можно улучшить работу?
    - Идеи по ходу ретроспективы

**? Артефакты SCRUM:**

* Product Backlog – упорядоченный по приоритету список задач, которые планируются
  + Уточнения бэклога
  + Definition of Ready (DoR) (Фокус на уровне бэклога) – Критерии подготовленности – помогает заказчику сделать хорошие юзер-истории
  + Definition of Done (DoD) (Фокус на уровне спринта) – Критерии готовности – помогает проверить работу со всеми требованиями проекта, а не только продемонстрировать, что функциональности работают
  + User Story – Пользовательские истории – Формулировка намерения того, что система должна делать для пользователя («**я могу делать что-то** (функциональность) **для чего-то** (ценность для бизнеса)»)
  + Planning Poker – Покер планирования – для оценки User Story в Story Point – 0, ½ , 1, 2, 3, 5, 8… - это оценка сложности (но не привязка ко времени)
* Sprint Backlog – цель спринта – набор эл-тов Product Backlog на данный спринт
* Инкремент – шаг на пути к цели продукта – инкремент объединяет реализацию эл-тов Product Backlog за текущий Sprint

**? Метрики SCRUM:**

* Velocity – Скорость – среднее арифметическое завершённых Story Point в предыдущих спринтах
* Capacity – Ёмкость – кол-во доступного времени команды – рабочие часы в рабочие дни × кол-во участн
* Burndown Chart – Диаграмма сгорания задач
* Cumulative Flow Diagram – Накопительная диаграмма потока

**? Разница между Sprint Backlog и Product Backlog**

В Product Backlog содержится список всех задач на проект

В Sprint Backlog список задач, выбранных из Product Backlog на данный спринт

**? SCRUM-master**

Не принимает участие в разработке

Следит, чтобы всё шло по SCRUM

**Тестовая Документация**

**? Тестовая документация:**

* Внешняя («публичная» документация, которую будут читать не только члены команды разработки)
  + Замечания
  + Баг-репорт
  + Запрос на изменение (улучшение) – feature request
  + Отчет о тестировании
* Внутренняя
  + Тест план
  + Тестовый сценарий
  + Тест-кейс (более детальный, чем тестовый сценарий)
  + Чек-Лист

**Test Strategy**

**? Тестовая Стратегия** – документ, который описывает, как производится тестирование, конкретно в данной компании.

**? Отличие Тест-Стратегии от Тест-Плана** – то, что стратегия указывает на то как все проекты тестируются в нашей компании, а тест план указывает, как будет тестироваться определённый проект.

**Test Plan**

**? Тест план (Test Plan)** — это документ, описывающий **весь объём работ** по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

**? На какие вопросы отвечает Тест план:**

* Что надо тестировать?
* Что будете тестировать?
* Что НЕ будете тестировать?
* Как будете тестировать?
* Когда будете тестировать?
* Критерии начала тестирования.
* Критерии окончания тестирования.

**? Основные пункты тест плана**  
В стандарте IEEE 829 перечислены пункты, из которых должен (пусть — может) состоять тест-план:  
a) Test plan identifier;  
b) Introduction;  
c) Test items;  
d) Features to be tested;  
e) Features not to be tested;  
f) Approach;  
g) Item pass/fail criteria;  
h) Suspension criteria and resumption requirements;  
i) Test deliverables; (практические результаты)  
j) Testing tasks;  
k) Environmental needs;  
l) Responsibilities;  
m) Staffing and training needs;  
n) Schedule;  
o) Risks and contingencies;  
p) Approvals.

**? Кто пишет Тест План** – пишет Лидер или самый старший из команды тестировщиков, или Менеджер.

**? Критерии начала тестирования**

* Готовность тестовой платформы (тестового стенда)
* Законченность разработки требуемого функционала
* Наличие всей необходимой документации

**? Критерии окончания тестирования**

* Результаты тестирования удовлетворяют критериям качества
* Требования к кол-ву открытых багов выполнены
* Выдержка определённого периода без изменений исходного кода приложения – Code Freeze (CF)
* Выдержка определённого периода без открытия новых багов – Zero Bug Bounce (ZBB)
* Отсутствие средств у заказчика и др.

**Test Case**

**? Тестовый случай (Test Case)** — это артефакт (документ), описывающий **совокупность** **шагов, конкретных условий и параметров**, **необходимых для проверки** реализации тестируемой функции или её части.

**? Структура Тест-Кейса:**

* Pre-Condition
* Test Case Description
* Post-Conditions

**? Тест-Кейс состоит из:**

* Уникальный идентификатор тестового случая
* Название
* Окружение (браузер-версия, окружение-версия, платформа)
* Предусловия (опционально)
* Шаги
* Ожидаемый результат
* История редактирования (опционально)

**? Чего НЕ должно быть в Тест-Кейсе:**

* Зависимостей от других Тест-Кейсов («…смотри Test Case ID3 step #12…»)
* Нечёткой формулировки шагов или ожидаемого результата
* Отсутствия необходимой для прохождения Тест-Кейса информации («…введите логин и пароль..» какие?)
* Излишней детализации

**? Виды Тест-Кейсов:**

По ожидаемому результату разделяются на:

* Позитивные – использует только корректные данные и проверяет, что приложение правильно выполнила вызываемую функцию
* Негативные – оперирует как корректными, так и некорректными данными (минимум 1 некорректный параметр) и ставит проверку исключительных ситуаций (срабатывание валидаторов), а так же проверяет, что вызываемая приложением ф-ция не срабатывает при вызове валидатора

**? Зачем нужны Тест-Кейсы:**

* Планирование – а только потом выполнение
* Тест-кейсы дают структурированный системный подход, что снижает вероятность пропуска ошибки.
* Хороший способ хранения части проектной документации.
* Способ протестировать документацию ещё до выхода билда
* Наличие тест-кейсов значительно ускоряет регрессионное тестирование
* Можно доверить выполнение новичку или коллеге из другого отдела
* Имея тест-кейсы можно в любой момент «вспомнить» что делалось месяц, пол-года, год назад.
* Тест-кейсы позволяют легко отслеживать прогресс (X% тестов выполнено, Y% тестов прошло/завалилось Z% требований покрыто тестами)

**? Тест-Кейс – Целесообразность написания для конкретного ПО**

* Жизненно важные системы – ошибка в которых может привести к гибели (самолётостроение, медицина, ПО для атомных станций)
* При тестировании сложных систем (интуитивно не понятных) или сложных частей системы, чтобы не запутаться в чек-листе

**? Тест-Кейс – Нецелесообразность написания для конкретного ПО**

* Простые системы – веб-сайты, мобильные приложения и т.п.
* Ситуации, когда в команде всего 1 или 2 тестировщика, знающие свой продукт – время, потраченное на создание и поддержку тест-кейсов никогда не окупится

**? «+» Достоинства Тест-Кейсов**

* Более быстрое введение в проект новых людей или подключение коллег из других проектов для проведения тестирования
* Напоминание о конфигурировании и настройке системы
* Незаменимы при работе на «тяжёлых проектах»
* Понимание информации одинаково всеми участниками проекта
* Напоминание о старой функциональности, которую всё ещё нужно тестировать

**? «–» Недостатки Тест-Кейсов**

* Разные Тест-кейсы для одного функционала очень похожи
* Сложность поддержки
* Неактуальное состояние
* Следуя сценарию можно упустить важные проблемы
* Валидация небольшого кусочка функциональности
* Тестировщик проверяет продукт, а не тестирует его
* Тестировщики «выключают мозг» проходя тест-кейс
* Любой может выполнить их, они не заменяют опытных тестировщиков, которые могут тестировать.

**? Тестовые данные (Test Data)** — это данные, которые нужны для выполнения Test case.

**? Тест-Кейс – хороший пример**

|  |
| --- |
| **Тест-Кейс №1:** Авторизация. Проверка валидации пароля на некорректный ввод.  **Предусловие:** Пользователь зарегистрирован в Приват24  **Шаги:**   1. Зайти на сайт https://www.privat24.ua 2. Ввести логин +380112223344 3. Ввести некорректный пароль test123   **Ожидаемый результат:**  Появляется сообщение об ошибке «Пароль введён неверно», пользователь не авторизирован. |

**? Тест-Кейс – плохой пример**

|  |
| --- |
| **Тест-Кейс №1:** Авторизация. Восстановление пароля.  **Предусловие:** Пользователь зарегистрирован в Приват24  **Шаги:**   1. Зайти на сайт – какой? 2. Ввести логин – какой? 3. Ввести некорректный пароль – сколько раз? 4. Заблокировать пароль – как? Ввести 3 раза пароль или жать восстановить пароль? 5. Восстановить пароль – каким способом? Шаги и действия?   **Ожидаемый результат:**  Пользователь авторизирован. |

**Check List**

**? Чек-лист (Check List)** — это документ, описывающий, что должно быть протестировано.

Это список, содержащий ряд необходимых проверок для какой-либо работы. Отмечая пункты списка команда или специалист может узнать о состоянии или корректности выполнения этой проверки.

**? Правила оформления Чек-листа**

* Один пункт – одна операция
* Пункты написаны в утвердительной форме
* Оптимальное количество пунктов – 20

**? Пример Чек-листа**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Проверка** | **Результат** | | |  |
| **Win XP** | **XP SP4** | **Win Vista** |  |
| **Операции с файлами** | ok | ok | ok |  |
| Создание файла | ok | ok | ok |  |
| Открытие файла | ok | ok | ok |  |
| Сохранение документа | ok | ok | ok |  |
| Печать | ok | ok | ok |  |
| **Редактирование файлов** | Bugs | Bugs | Bugs |  |
| Отмена | ok | ok | ok |  |
| Вырезание | Bug #146 | Bug#146 | Bug#146 |  |
| Копирование | ok | ok | ok |  |
| Вставка | ok | ok | ok |  |
| Удаление | ok | ok | ok |  |
| Поиск | Bug #123 | Bug #133 | ok |  |
| Поиск с заменой | Bug #126 | ok | ok |  |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Описание** | **Пример** | **Результат** |
| Корректный email (популярный домен) | olga@mail.ru | Регистрация прошла успешно, на почту отправлено письмо-приветствие. |
| Корректный email (корпоративная сеть) | olga@company.ru |
| Точка внутри email | ok.molechka@gmail.com |
| Кириллический email | олечка@мусики.рф | На кириллический email ушло письмо |
| Пустая почта |  | Ошибка «Введите email» |
| Одно слово вместо домена | olgak@fdgfdg | Ошибка «Введите адрес в формате mail@site.com» |
|  |  |  |

**? «+» Преимущества Чек-листов**

* Структурирование информации
* Повышение скорости обучения новых сотрудников

**Test Report**

**? Тест репорт (отчёт)** – документ предоставляющий сведения о соответствии/несоответствии ПО требованиям

* технические пользователи (участники команды разработки)
* менеджеры продукта (представители заказчика)
* бизнес-пользователи (те кто заказал доработку функционала – из бизнес-отдела)

**? Тест-репорт (отчёт) по времени**:

* дневной, недельный, месячный
* промежуточный
* финальный

**Requirements Traceability Matrix**

**? Матрица покрытия требований (RTM — Requirements Traceability Matrix)** - это документ, который связывает требования с тест-кейсами.

**Bug Report**

**? Баг Репорт (Bug Report)** — это документ, описывающий **ситуацию** или **последовательность действий** приведшую к **некорректной работе объекта** тестирования, с указанием причин и ожидаемого результата.

**? Основные поля баг-репорта** – Summary, STR (steps to reproduce), ER+AR, attachments

**? Приоритет (Priority) бага** – атрибут очередности выполнения задачи или устранения ошибки.

Приоритет проставляется исключительно менеджером проекта или руководителем компании.

* **P1 Высокий (High)**  
  Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.
* **P2 Средний (Medium)**  
  Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.
* **P3 Низкий (Low)**  
  Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

**? Серьёзность (Severity)** – атрибут влияние дефекта на работоспособность (на систему).

* **S1 Блокирующая (Blocker)**  
  Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.
* **S2 Критическая (Critical)**  
  Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.
* **S3 Значительная (Major)**  
  Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.
* **S4 Незначительная (Minor)**  
  Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.
* **S5 Тривиальная (Trivial)**  
  Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

## ? Что такое утечка дефектов и релиз бага? (Bug Leakage & Bug Release)

Релиз бага – это когда программное обеспечение или приложение передается группе тестирования, зная, что дефект присутствует в выпуске. При этом приоритет и серьезность ошибки низки, поскольку ошибка может быть удалена до окончательной передачи обслуживания.

Утечка бага - когда баг обнаруживается конечными пользователями или заказчиком, а не обнаруживается группой тестирования во время тестирования программного обеспечения.

## ? Что означает плотность дефектов при тестировании ПО (KLOC)?

Плотность дефектов – это количество дефектов, подтвержденных в ПО/модуле в течение определенного периода эксплуатации или разработки, деленное на размер ПО/модуля. Это позволяет решить, готова ли часть ПО к выпуску. Плотность дефектов рассчитывается на 1000 строк кода, и обозначается KLOC.

Однако не существует фиксированного стандарта для плотности ошибок, исследования показывают, что один дефект на тысячу строк кода обычно считается признаком хорошего качества проекта.

## ? Что означает процент обнаружения дефектов при тестировании ПО? (DDP)

Процент обнаружения дефектов (DDP) – это тип метрики тестирования. Он показывает **эффективность процесса тестирования** путем измерения соотношения дефектов, обнаруженных до выпуска и сообщенных после выпуска клиентами. Например, скажем, QA зарегистрировало 70 дефектов во время цикла тестирования, а клиент сообщил еще 20 после выпуска. DDP составит 72,1% после расчета 70 / (70 + 20) = 72,1%.

## ? Что означает эффективность устранения дефектов при тестировании ПО? (DRP)

Эффективность устранения дефектов (DRP) - это тип метрики тестирования. Это показатель **эффективности** **команды разработчиков** для **устранения проблем** перед выпуском. Он измеряется как отношение зафиксированных дефектов к общему количеству обнаруженных проблем. Например, допустим, что во время цикла тестирования было обнаружено 75 дефектов, в то время как 62 из них были устранены командой разработчиков во время измерения. DRE достигнет 82,6% после расчета 62/75 = 82,6%.

## ? Что означает эффективность Test case в тестировании ПО? (TCE)

Эффективность тестирования (TCE) - это тип метрики тестирования. Это четкий показатель **эффективности выполнения Test case** на этапе выполнения теста в выпуске. Это помогает в обеспечении и измерении качества Test case. Эффективность тестового набора (TCE) => (Количество обнаруженных дефектов / Количество выполненных Test case) \* 100

## ? Возраст дефекта в тестировании ПО?

Возраст дефекта - это время, прошедшее между днём ​​обнаружения тестировщиком и днём, когда разработчик исправил его.

**JIRA**

? JIRA – это программный инструмент для управления проектами, разработанный компанией Atlassian. Jira часто используется в IT-компаниях для формирования списка задач, отслеживания общего прогресса команды и решения возникающих по ходу разработки продукта проблем.

? По каким принципам построена JIRA – по принципам канбан- и скрам-досок + массой вспомогательных механизмов, с целью упростить создание новых приложений, добавить в них функции, исправить ошибки и т.п. Эта система управления проектами исповедует Agile-методику разработки.

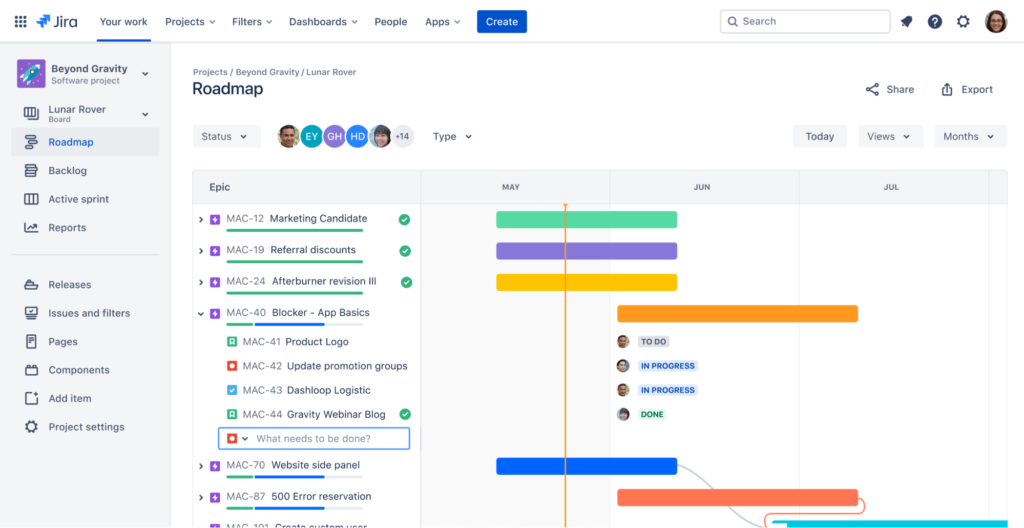
? Алгоритм работы с Jira

* Для начала нужно загрузить Jira, создать профиль и запустить утилиту.
* В окне приложения необходимо выбрать пункт Create Project.
* Программа предложит список шаблонов для доски с задачами (для разработчиков, для маркетологов и т.п.). Выбираем ту, что лучше всего соответствует целям команды и стилю работы в вашей компании.
* Настраиваем колонки под своим нужды (если то, что было предложено в шаблоне, не на 100% удовлетворяет вашим требованиям).
* Создаем задачу (пункт Create).
* Приглашаем других пользователей (то есть членов команды) работать с созданной вами доской (пункт Invite).

**? Как устроена Jira**

* Интерфейс

Интерфейс Jira делится на несколько ключевых вкладок. Во вкладке «Projects» хранятся все канбан/скрам-доски, которые вы можете просматривать или редактировать. Фактически это основное рабочее пространство. Здесь же можно перейти в режим отслеживания релизов продукта, взглянуть на все активные спринты, проанализировать отчеты о проделанной работе и т.п.



Также в списке вкладок есть окно с дашбордами – удобно скомпонованными аналитическими сводками. Отдельное окно со списком сотрудников, с которыми вы взаимодействуете, система планирования релизов на манер инструментов в духе OmniPlan и вкладка с приложениями от сторонних компаний, интегрированными в ваш профиль Jira.

* Задачи (Issue)

Issue – это единица информации. В нее закладывается либо какая-то функция, которую нужно реализовать, либо ошибка в программе, которую необходимо исправить.

Issues – это составные части проекта и спринта. Именно список задач формирует рабочий процесс. Поэтому он и состоит из создания задач, наблюдения за ними, выполнения, анализа, дополнения, изменения и т.п.

Типы задач – для более удобной категоризации можно выбрать один из вариантов, например

* новая функция,
* баг,
* подзадача,
* эпик и т.п.

Выбор типа задач зависит от целей команды и компании. Можно создавать свои типы для удобного распределения, фильтрации и поиска задач. Соответствующий раздел настроек находится в Project settings.

* Дорожная карта (расписание) (Roadmap), Epic

В этом разделе можно создавать цели и планировать работу команды наперёд. Ключевой единицей информации тут является эпик. Это объединение большого количества issues, связанных друг с другом.

К примеру, если есть ряд новых функций для приложения, которые совместно формируют какую-то общую важную особенность ПО, то их объединяют в эпик, как некую общую цель, к которой стремится команда в ходе спринта (или нескольких спринтов).

На дорожной карте хорошо видны далеко идущие планы компании, визуально оформленные в своего рода горизонтальный календарь.

* Релизы (Releases)

Каждый набор новых функций в приложении или пакет исправленных ошибок отправляется к пользователям в виде новой версии этого самого приложения. Версионность – самый удобный, часто используемый и фактически ставший индустриальным способ развития программных продуктов.

Поэтому в Jira такой акцент сделан на контроле новых версий. В соответствующем разделе можно создавать версии программ, указывать дату выпуска и закреплять за ними исправления багов, новые функции и issues, входящий в конкретный релиз.

* Код и деплой

Одно из преимуществ Jira – возможность тесно интегрировать её с другими продуктами, например с платформами Bitbucket, Github и Gitlab.

У лидеров команды появляется возможность наблюдать не только за прогрессом как за набором меняющихся активных задач, но и смотреть на реальные изменения в коде.

Интеграция позволяет разработчикам напрямую отправлять каждый коммит в Jira, чтобы другие члены команды могли видеть изменения из условного Github прямо в системе управления проектами.

* Pages

Проект Confluence – это что-то в духе Google Docs, только работающее в рамках Jira и менее функциональное. Это онлайн-текстовый редактор с базовыми инструментами для форматирования написанного. Суть Confluence в создании дополнительной удобной среды для общения лидеров команды и разработчиков.

* Дашборды

В Jira дашборды выводят кучу полезной информации. Всяческие отчеты, статистика комментариев, список выполняемых задач, аналитические данные, графики, таблицы, схемы. Полезно для аналитиков компании и бизнеса в целом. Собирают все необходимые данные в одно пространство без необходимости следить за процессом работы команды и фиксировать какие-то значимые аспекты, чтобы потом вручную делать аналитические сводки. В Jira можно формировать дашборды автоматически. Они подходят не только для аналитики, но и для постоянного наблюдения за тем, как протекает рабочий процесс, и для принятия радикальных решений в случае упадка производительности или возникновения других проблем.

* Плагины

Jira можно сделать еще функциональнее, если подключить к ней плагины сторонних компаний.

Некоторые из них продвигает сама Atlassian. Один из наиболее распространенных и очевидных вариантов использования плагинов – интеграция с Git-системами.

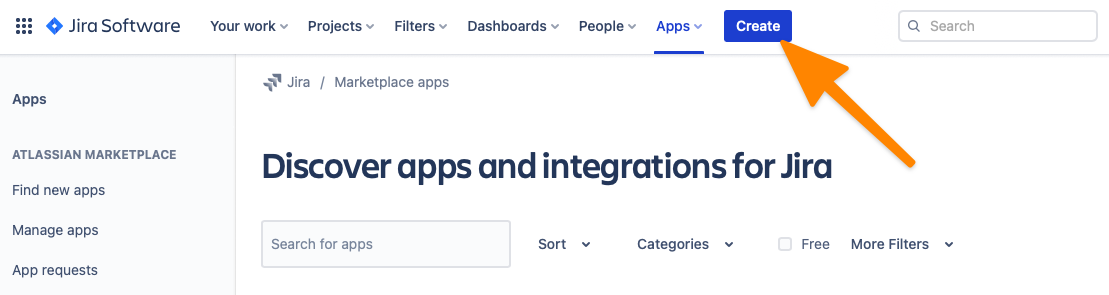
В коллекции плагинов можно найти инструменты для создания диаграмм, более удобной визуальной презентации рабочего расписания, отправки задач по почте и т.п.

? Как создать задачу в Jira

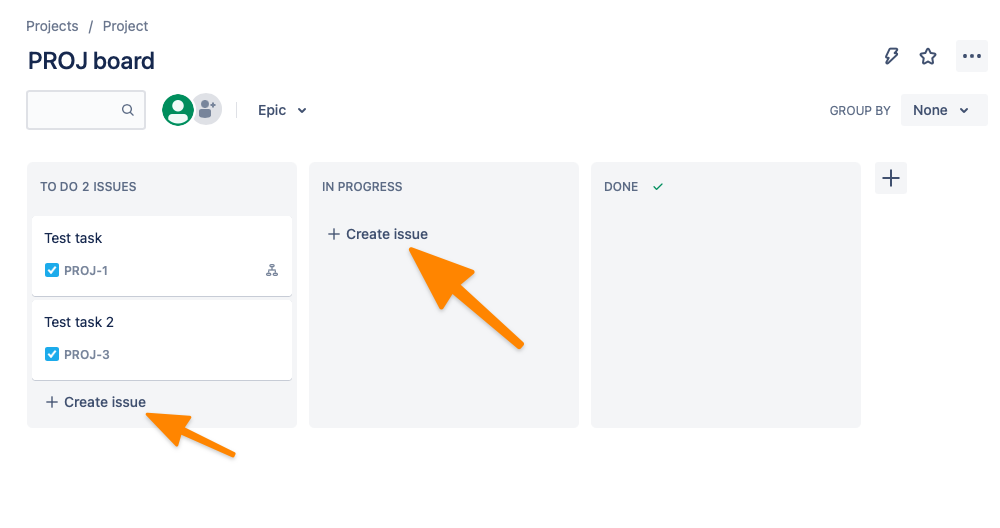
Создать новую задачу в Jira можно двумя путями:

* + Кликнув по кнопке Create в верхней панели управления.
  + Кликнув по кнопке Create issue в нужной колонке канбан-доски.

В первом случае нужно будет выбрать проект, в котором необходимо создать задачу.



Во втором – указать название задачи и прописать дополнительные атрибуты.



Уже после этого можно кликать по кнопке Create, и новая issue автоматически появится в списке и на выбранной доске. А если поставить галочку напротив Create another, то тут же появится окно для добавления еще одной задачи.

**? Атрибуты задач**

* **Summary**. Краткое описание текущей задачи. Буквально в одно предложение.
* **Description**. Полноценное описание, если таковое требуется.
* **Assignee**. Член команды, которому нужно делегировать создаваемую задачу.
* **Labels**. Что-то вроде тегов для более удобной сортировки задач по другим признакам, не входящим в список типов.
* **Fix** **version**. К какой версии относится создаваемая issue.
* **Story** **point** **estimate**. Потенциальные трудозатраты, требующиеся на добавление новой функции или исправление бага.
* **Reporter**. Пользователь, который будет отчитываться за выполнение задачи.
* **Attachment**. Файл, прикрепленный к задаче. Это может быть что угодно: аудио, картинка, документ docx и т.п.
* **Linked issues**. То, с чем связаны создаваемые задачи (другие задачи/проекты).

**? Настройка отчётов**

В графе Reports можно взглянуть на автоматически сгенерированные отчеты о проделанной работе. Пользователям Jira доступны 4 вида отчетов.

* Burnup report

График, показывающий результаты работы по конкретному спринту в сравнении с общей производительностью команды разработчиков. Его используют для оценки эффективности текущего спринта.

* Sprint burndown chart

График, показывающий, какой еще объем работы необходимо выполнить команде, чтобы продвинуться к завершению текущего спринта. Используется для оценки индивидуальной и общей производительности, а также для приблизительной оценки сроков реализации установленных планов.

* Velocity report

Используется для отображения потенциальной производительности команды в будущем. То есть на основе уже завершенных спринтов Jira пытается предугадать, как много задач удастся выполнить разработчикам в ходе следующего «забега».

* Cumulative flow diagram

Показывает, как менялся статус активных задач с течением времени. В каких колонках созданные issues задерживаются дольше всего. Используется для поиска так называемых бутылочных горлышек – проблемных этапов работы, на которых резко падает производительность всей команды.

**? Основные принципы повышения производительности в Jira**

Есть как минимум 5 способов сделать работу с Jira эффективнее.

* Делите большие задачи на мелкие

Это главная заповедь канбан и скрам, но люди все равно об этом забывают и продолжают лепить карточки с очень массивными задачами. Необходимо всегда создавать максимально компактные задачи. Такие, которые легко понять, выполнить, зафиксировать, объяснить и так далее. Каждая issue должна быть понятной единицей информации, представляющей собой компонент более глобальной цели.

* Комментируйте задачи

Сохранив все записи в едином пространстве, вы сохраните кучу времени себе в будущем, когда будете вспоминать или искать что-то связанное с конкретной задачей. Освобождайте свою голову сразу по ходу создания issues и работы с ними. Нужно помнить о какой-то особенности исправляемой ошибки? Есть какая-то идея по более прагматичной реализации запланированной функции?

* Записывайте все выполненные действия

Лог – это способ фиксировать коммиты в Jira. По сути, те же текстовые комментарии

Комментарии отражают процесс выполнения задачи и помогают с решением поставленных целей. Но есть еще логи. Они отражают результаты выполненной работы в течение определенного времени. Логи работают по тому же принципу, что и коммиты. Коммит – это фактически выгрузка любого изменения приложения в git-систему. Поменяли цвет иконки? Сделайте коммит и отправьте его в git-систему. Добавили новую функцию в код? Сделайте еще один коммит. И так на любой чих.

* Планируйте спринты

Спринт – удобная схема оптимизации рабочего процесса, но к ней тоже нужно готовиться. Важно заранее спланировать список задач, оценить адекватность поставленных целей, приблизительно оценить сроки выполнения работы, расставить приоритет по задачам. Заранее понять, что, скорее всего, будет отложено на следующий «забег», а что можно сделать быстро и в первую очередь.

* Делайте записи на регулярной основе

Вышеописанные процедуры нужно выполнять раз в час-два. Постоянно что-то коммитить, комментировать, записывать и т.п. Все, что не записано, то утеряно. Ваша задача – выработать полезную привычку фиксировать каждое выполненное действие, постоянно делать полезные заметки и всячески демонстрировать свою полезность и эффективность в команде. Большое количество записей действительно облегчает работу коллегам, так как канбан-доска постепенно обрастает всей необходимой информацией. Не приходится все искать самостоятельно.

? Аналоги Jira

* Trello
* Basecamp
* YouTrack
* ClickUp

**GIT**

? GIT – распределённая система контроля версий.

? Ещё распределённые системы контроля версий – SVN, Mercurial

? Git – надёжная система – не просто хранит файлы, а для всего вычисляет контр сумму и тут же сообщит о повреждениях файлов

? Код храниться на сервере. В код вносятся только изменения.

? Изменения можно откатить назад.

? GIT старается проверять изменения на мастер-ветке.

? xxx644 – файл неисполняемый / xxx755 – файл исполняемый

? Что такое репозиторий – Репозиторий Git представляет собой **каталог файловой системы**, в котором находятся файлы **конфигурации репозитория**, **файлы журналов**, хранящие операции, выполняемые над репозиторием, **индекс**, описывающий расположение файлов, и **хранилище**, содержащее собственно файлы

? Коммит – единица состояния проекта Git

? Коммит должен быть атомарным – реализовывать только одну цель

? Коммит должен быть консистентным – логически завершённым

? Commit early, commit often! – коммиты должны быть частыми и сразу, как изменения готовы

? Что такое GitHub – Веб-сервис для хостинга IT-проектов и их совместной разработки, основанный на Git

? Что означает статус файла «new» при выводе git status – Что файл только начал отслеживаться Git и пока не имеет истории

? Ветка – Это разные пути развития проекта, по сути, разные последовательности коммитов

? Чем отличается master и origin master – master принадлежит локальному репозиторию, а origin master – удаленному

? Слияние двух веток – Когда все коммиты, сделанные для одной ветки, становятся видимыми во второй ветке

? Почему бывают конфликты при слиянии веток – в обеих ветках есть изменения одних и тех же строк

? Как скачать ветку their\_branch, если она уже есть в удалённом (remote) репозитории, но нет локально –  
 git fetch origin their\_branch

? Для чего добавлять файлы в .gitignore – Чтобы Git не замечал их, и любые команды Git не могли их зааффектить

? Git не работает с пустыми директориями (папками)

? Файл «.gitignore» создаётся в корне проекта, там же где и каталог «.git»

? Переименования файлов рассматривается как = удаление файла со старым именем + создание файла с новым именем, т.к. сперва файла с новым именем не было в ИНДЕКСЕ и Гит о нём не знал, но после добавления в ИНДЕКС Гит подсчитывает контрольную сумму, и в итоге понимает, что файл был просто переименован

? Cherry-pick – коммит исправляющий ч.л. на ветке Master можно применить и к коммиту другой ветки

Index.html

Index.html

(изменён)

Index.html

(вся история)

WORK DIRECTORY

INDEX

LOCAL REPOSITORY

**Folder .git**

**git.add**

**git.commit**

Index.html

(вся история)

REMOTE REPOSITORY

**git.push**

**GIT HUB**

? Создание новой пустой ветки – это просто создание новой ссылки на коммит, тот же на котором в именно данный момент ссылается уже ведомая ветка (н-р Мастер)

? Создать Репозиторий только через CLI

mkdir test

cd test

git init

echo "This repo was created remotely" >> README.md

git add .

git commit -m "first commit"

git remote add origin git@github.com:USER/test.git

curl -u 'USER:TOKEN' https://api.github.com/user/repos -d '{"name":"test"}'

git push -u origin master

*(\*) USER - ваш логин на гитхабе  
TOKEN - токен типа 6ccfd64d55fc1ca1cc26ffe2b9351cc9 сгенерированный по инструкции*

? Проблема при мердже – одно из решений – зайти непосредственно в конфликтный файл в системе, удалить различия между репозиториями (будут помечены >>>>> и <<<<<), сохранить, и тогда add-commit-push

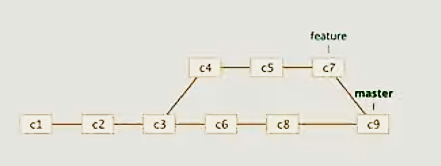
? 3 способа добавить файлы в РЕПО, минуя ИНДЕКС:

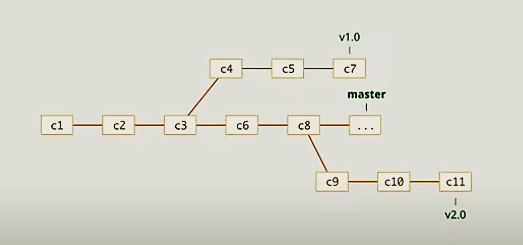
git commit -a

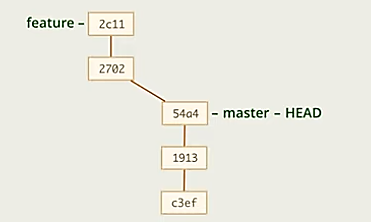
git commit <path>

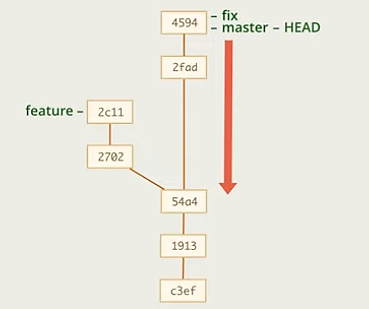
alias.commitall “!git add .; git commit”

? Ветвление

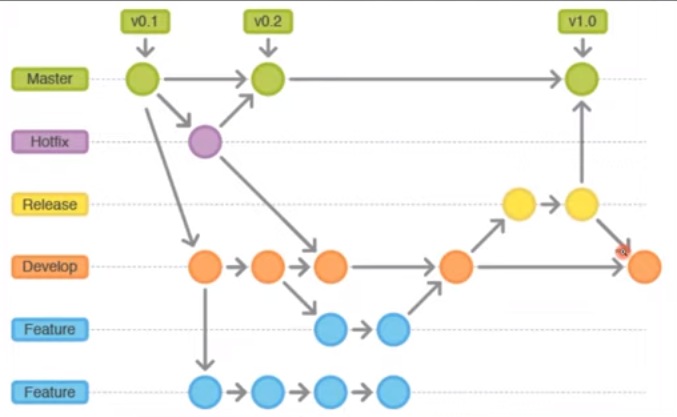








? Git-flow, рутинная работа выглядит:



**? Навигация в BASH:**

pwd – показать текущее местоположение

ls – показать файлы в папке, кроме скрытых

ls -f – показать файлы в папке, включая скрытые

cd c:/ – перейти в данный каталог

cd- – вернуться назад

cd .. – выйти на 1 уровень вверх

cd ../.. – выйти на 2 уровня вверх

mkdir – создать папку

cd!$ – перейти во вновь созданную папку

mkdir folder1 folder2 = mkdir -p {folder1,folder2} – создать сразу несколько папок folder1 и 2

mkdir -p folder/{subfolder1,subfolder2} – создать folder и сразу несколько подпапок subfolder1 и 2

touch file.html – создать файл

touch folder/{file.html,file.txt} – создать папку folder и несколько файлов

rm folder – удалить папку, только если она пустая

rm -r Folder – удалить папку Folder и файлы в ней

echo “abcdefg” > file.txt – перезаписывает файл новым содержанием “abcdefg”

echo “abcdefg” >> file.txt – добавляет в файл новое содержание “abcdefg”

cat file.txt – выводит текст из файла

q – закрытие встроенных текстовых программ

**? Команды в GIT**

РЕПОЗИТОРИЙ

git init – создание репозитория

git clone (URL) – копирование существующего репозитория, с текущим именем

git clone (URL) NEWNAME – копирование существующего репозитория, с новым именем NEWNAME

cat .git/config – инфо о содержимом

git config user.name – показывает логин

git config user.email – показывает имейл

git help config – помощь по config

/abcde – найти инфу abcde

n – переход вниз по найденному

<shift>n – переход вверх по найденному

git config --global core.excludesFile ~/.gitignore – открыть файл в редакторе и поместить туда игнорируемые файлы, каталоги – настройка глобального (а не локального) «.gitignore»

ПСЕВДОНИМЫ

git config --global alias.c config 🡪 git config = git c

ПОМОЩЬ

git add -h – краткая справка по add

git help add – полная справка по add

ИЗМЕНЕНИЕ

git status – состояние каталога и раздела проиндексированных файлов

git add file.txt – добавляет файл в индекс

git add -all = git add **.** – добавляет всё в индекс, отслеживает и новые файлы и изменения файлов **без удаления**

git add -u – отслеживает изменения файлов и удаление, **без создания новых файлов**

git add --force = git add -f – отслеживает файлы не смотря на нахождение их в «.gitignore»

git add -p File.txt – Git запрашивает какие именно изменения, строки внутри файла мы хотим сохранить

git add -A – добавит все изменения с самого корня проекта

git commit -m “text about commit” – («-m» message ОБЯЗАТЕЛЬНО!) берёт данные из индекса, сохраняет слепок в локальном репозитории и сдвигает маркер на этот слепок

git commit -m “text about commit” File.txt – берёт данные из индекса только для File.txt, сохраняет слепок в локальном репозитории и сдвигает маркер на этот слепок

git commit -a -m “text about commit” = git commit -am “text about commit” – («-a» = «--all», «-m» message) берёт все изменения, МИНУЯ ИНДЕКС, сохраняет слепок в локальном репозитории и сдвигает маркер на этот слепок, НО! «git commit -a» работает только с отслеживаемыми ранее файлами, в которые потом вносятся изменения, но не с вновь созданными (вновь созданные файлы нужно проиндексировать сначала через git add)

git commit --author=’Name Lastname <Lastname@gmai.com>’ --date=’…….’ – указывает имя автора и дату создания при коммите, но автор коммита и дата коммита будут нашими (но их тоже можно изменить)

git fetch – (приносит) показывает изменения В удалённом репозитории по сравнению с локальным

git pull – заливает изменения ИЗ удалённого репозитория в локальный (если в удалённом РЕПО к.л. или мы сделали изменения, и он уже не совпадает с последним снимком нашего локального, то при попытке запушить из локального в удалённый будет ошибка; сл-но перед push – надо обновить снимок pull)

git push = git push origin main – заливает изменения ИЗ локального репозитория в удалённый

git log – посмотреть историю коммитов

git log --author – посмотреть лог изменений определённого автора

git show – изменения, которые были в последнем коммите

git show adJuNKL34e33f(hash) – изменения, которые были в коммите hash № adJuNKL34e33f

git show e740 – изменения, которые были в указанном коммите № e740u5f (указать min 4 знака)

git blame file.txt | grep “Abcde” – инфо об изменениях «Abcde», которые вносились в файл file.txt

git blame file.txt | grep UserName – инфо об изменениях, которые вносились в файл file.txt автором UserName

git diff – показывает изменения, которые были произведены в файле относительно локального репозитория

git reset – отменить git add

git reset HEAD~1 – откатывает изменения на 1 шаг назад

git reset HEAD Folder – убирает изменения папки Folder в ИНДЕКСЕ

git reset hard HEAD~1 – безвозвратно удаляет

git rm File.txt – удаляет файл НО! если в файле сделали изменения и хотим его удалить Git выдаст конфликт, т.к. волнуется, что изменения могут пропасть безвозвратно, без сохранения в РЕПО

git rm -f File.txt – форсировано удаляет файл, не смотря на то, были ли в нём незакоммитченные изменения – удалит файл и из ДИРЕКТОРИИ и из ИНДЕКСА

git rm -r Folder – удаляет Folder из WORK DIRECTORY и ИНДЕКСА, и сохраняет изменения в индексе

(git rm -r Folder = rm -r Folder + git add Folder)

git rm -r --cashed Folder – удаляет файл из отслеживаемых в Индексе, но оставляет в Директории

git rm -r --cached Folder (+git commit -m “delete Folder” + git push origin main) – удалить папку Folder и файлы в ней из удалённого РЕПО

git mv OldName NewName – переименовывает одновременно и в Директории и добавляет инфо в Индекс

git checkout – удаляет последние изменения в файле

git checkout **.** – удаляет последние изменения в файлах

git checkout path/to/file – привести измененный файл в начальное состояние (до изменения)

git stash – сохраняет последние изменения в последнем файле во временное хранилище

git stash pop – достаёт обратно последние изменения в последнем файле из временного хранилища + показывает инфу о временном хранилище

git stash clear – очищает временное хранилище

git apply path/to/file – применить патч в Git

git clean -f – удалить все не отслеживаемые файлы

git clean -fd – удалить все не отслеживаемые файлы и папки

СОЗДАНИЕ ВЕТОК

git branch NewBranch – создаёт новую ветку

git branch – инфа о локальных ветках и местонахождении среди них

git branch -v – инфа о ветках с коммитами

HEAD – указатель на ветку где мы находимся

ПЕРЕКЛЮЧЕНИЯ МЕЖДУ ВЕТКАМИ

git checkout NewBranch – HEAD переключается с ветки Main на NewBranch

git checkout -b NewBranch – создаёт новую ветку и тут же переключается на неё (=branch+ checkout)

git push -u origin NewBranch – («-u» - upstream) сохранит новую ветку в удалённом РЕПО (иначе не получится запушить изменения/файлы, т.к. в удалённый РЕПО не видит ветки NewBranch)

git fetch origin NewBranch – скачать ветку, если её нет в локальном репозитории, но есть в удалённом

git show HEAD~ = git show HEAD~1 – показывает состояние на 1 коммит назад

git show HEAD~~ = git show HEAD~2 – показывает состояние на 2 коммита назад

git show HEAD~~~ = git show HEAD~3 – показывает состояние на 3 коммита назад и т.д.

(Обратно вернуться)

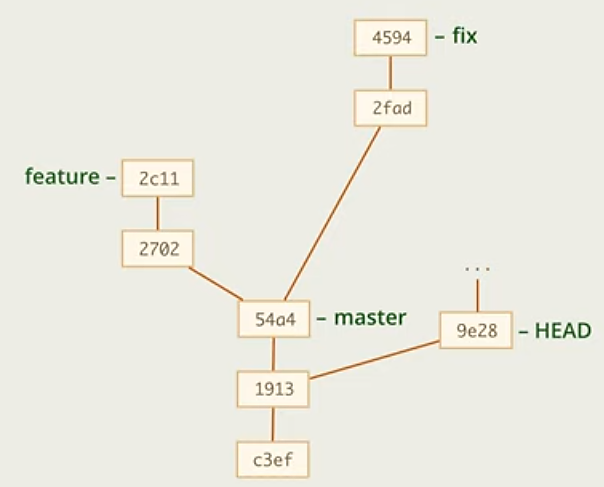
git checkout Main – HEAD переключается с NewBranch на последний коммит ветки Main, также обратно изменяются файлы в Директории на тот момент, НО! последние изменения в File.txt на NewBranch должны быть закоммитчены, иначе выйдет ошибка, чтоб не потерять последние изменения в File.txt на NewBranch

git checkout --force Main = git checkout -f Main – принудительное переключение с NewBranch на Main, изменения в File.txt на NewBranch будут потеряны

git checkout -f HEAD = git checkout -f – удалит все не сохранённые изменения в файле File.txt – перезапишет File.txt ранее закоммитченным файлом File.txt без изменений

git checkout 19а5 – HEAD переключается на коммит № 19а5 в середине к.л. ветки, переходит в состояние «отделённый HEAD», файлы вернуться в состояние коммита № 19а5, если делать изменения и коммиты с этой точки то получится как бы не созданная ветка без названия, с которой если уйти, то потом вернуться можно только по номеру коммитов на ней, а номера со временем могут забыться/затеряться (рис.)

git cherry-pick 9e28 – скопирует коммит с «пустой» ветки на текущую



git stash – сохраняет последние изменения в последнем файле во временное хранилище

git stash pop – достаёт обратно последние изменения в последнем файле из временного хранилища + показывает инфу о временном хранилище, нужно доставать НА ТОЙ ЖЕ ветке, где и применили

git stash clear – очищает временное хранилище

(Забыли создать ветку, находимся на Main)

1. git branch Fix – создать ветку Fix
2. git checkout Fix – переключить HEAD на ветку Fix
3. git branch -f Main 54o5 – откатить ветку Main к коммиту № 54o5 с которого бы следовало создать ветку Fix ранее, к коммиту № 54o5 переносится и HEAD
4. git branch -f Main Fix – если обратно передумали и вернуть Main к состоянию последнего коммита на Fix

СЛИЯНИЕ ВЕТОК

git checkout - – перейти с ветки NewBranch на ветку Main

git merge NewBranch – ветка NewBranch сольётся с веткой Main в локальном РЕПО

git merge --abort – отменить слияние веток, если произошёл конфликт

git push – ветка NewBranch сольётся с веткой Main в удалённом РЕПО

git push -u origin NewBranch – исправить ошибку "fatal: The current branch my\_branch has no upstream branch", возникающую при вводе git push

Конфликт: в удалённом и локальном репозитории изменили там и там разные файлы

1. git branch 2nd
2. git checkout 2nd
3. touch File-2.txt
4. git add File-2.txt
5. git commit -am “2nd”
6. git push -u origin 2nd
7. git checkout main
8. git merge 2nd

**Error**

1. git pull
2. git push

Если конфликт: **You have not concluded your merge (MERGE\_HEAD exists)**, выполнить:

1. git add .
2. git commit -m “message”
3. git push

или

1. git merge --abort

или

1. Зайти в папку, открыть файл, стереть лишнее (будут помечены >>>>> и <<<<<), сохранить
2. git add File.txt
3. git commit -am “message”
4. git push

ПЕРЕИМЕНОВАНИЕ ВЕТОК

git branch -m first\_branch second\_branch – first\_branch переименуется в second\_branch

Конфликт: в ветке содержится файл и она была запушена, переименовать second\_branch в first\_branch

1. git branch first\_name
2. git branch -m first\_branch second\_branch – first\_branch переименуется в second\_branch
3. git checkout second\_branch – переходим на ветку
4. touch File.txt – создаём файл
5. git commit -am “message” – коммитим
6. git push -u origin second\_branch – отправляем в удалённый РЕПО
7. git branch -m second \_branch first \_branch – **ОК НО!** в удалённом РЕПО ветка НЕ переименуется
8. git push origin :second\_name – удалит ветку в удалённом РЕПО
9. git push -u origin first\_branch – отправляем в удалённый РЕПО ветку с новым названием

УДАЛЕНИЕ ВЕТОК

git branch -d NewBranch – удаляет ветку на локальном репозитории

git fetch origin NewBranch – возвращает ветку в локальный репозиторий, НО! чтобы она появилась (в списке git branch) на неё надо перейти: git checkout NewBranch

**HTML**

? HTML – язык разметки гипертекста, НЕ является языком программирования

? Mosaic, Net Scape – ранние браузеры, могли открывать только 1 веб-сайт

? Терминология

• Элемент (element) — конструкция языка HTML. Это контейнер, **содержащий данные** и **позволяющий отформатировать** их определенным образом. Любая **Web-страница** представляет собой **набор элементов**. Одна из основных идей гипертекста — возможность вложения элементов.

• Тег (tag) — начальный или конечный маркеры элемента. Теги определяют **границы действия элементов** и **отделяют** элементы друг от друга. В тексте Web-страницы теги заключаются в угловые скобки, а конечный тег всегда снабжается косой чертой.

• Атрибут (attribute) — **параметр или свойство** элемента. Это, по сути, **переменная**, которая имеет **стандартное имя** и которой может присваиваться определенный **набор значений**: стандартных или произвольных. Предполагается, что символьные значения атрибутов заключаются в прямые кавычки, (но некоторые браузеры позволяют не использовать кавычки). Атрибуты располагаются внутри начального тега и отделяются друг от друга пробелами.

• Гиперссылка — **фрагмент текста**, который является **указателем** на другой файл или объект. Гиперссылки необходимы для того, чтобы обеспечить возможность перехода от одного документа к другому.

• Фрейм (frame) — этот термин имеет два значения. Первое — область документа со своими полосами прокрутки. Второе значение — одиночное изображение в сложном (анимационном) графическом файле (по аналогии с кадром кинофильма).

• Апплет (applet) — **программа, передаваемая** на компьютер клиента в виде **отдельного файла** и запускаемая при просмотре Web-страницы.

• Скрипт или сценарий (script) — **программа, включенная в состав** Web-страницы для расширения ее возможностей.

• Расширение (extension) — элемент, не входящий в спецификацию языка, но использующийся, обеспечивая возможность создания нового интересного эффекта форматирования.

• HTML-файл или HTML-страница — **документ**, созданный в **виде гипертекста** на **основе** языка HTML. Такие файлы имеют, как правило, расширения htm или html. В гипертекстовых редакторах и браузерах эти файлы имеют общее название «документ».

• Код HTML — гипертекстовый **документ на языке html в своем первоначальном виде**, когда **видны все элементы и атрибуты**.

• Язык HTML – **набор специальных правил**, каждому соответствует своё название

• Web-страница — документ (файл), подготовленный в формате гипертекста и размещенный в World Wide Web.

• World Wide Web, WWW или просто Web — Всемирная паутина, **распределенная система доступа к гипертекстовым документам**, существующая в Интернете. HTML является основным языком для создания документов в WWW.

• Сайт (site) — набор Web-страниц, принадлежащих одному владельцу.

• Браузер (browser) — программа для просмотра Web-страниц.

• Пользовательский агент (user agent) — браузер или другая программа, работающая на компьютере-клиенте.

• Загрузка (downloading) — копирование файлов с сервера на компьютер-клиент.

• URL (Uniform Resource Locator) или универсальный указатель ресурса — адрес некоторого объекта в Интернете.

• Базовый URL — часть адреса, которая является общей для всех ссылок текущей Web-страницы

? Тэги – группы, обособленные <>…….</>, с одинаковым св-вом и значением при контейнере «элемент»

? Все эл-ты делятся на 3 группы:

* Создание структуры
* Эффекты форматирования
* Управление программами

? Значение параметра ставится в двойные кавычки “……….”, разделяются пробелами – «;» между параметрами не ставится

<table width =”100%” border=”solid”>

? HTML Регистронезависимый: ABCDE = abcde

? <!DOCTYPE html> – эл-т для указания типа док-та из HTML4.01, 5, XHTML1.0, 1.1 (!DOCTYPE html – это HTML5)

? Разделы HTML:

* <!DOCTYPE html> - первоначальный эл-т
* Раздел <html> – док-т строится на HTML (мы не видим этот раздел)
* Раздел <head> – метаданные, шрифты (мы не видим этот раздел)
  + <meta charset=”utf-8”> – служебная инфа, находится внутри раздела head
  + <title> – (парный) эл-т заголовка, отображающийся на вкладке браузера и при поиске страницы
  + <style > – (парный) эл-т, задание стиля эл-тов, вместо заданного по умолчанию
* Раздел <body> – содержание Веб-страницы
  + Текст
  + Фон
  + Границы
  + Ссылки

? <stile> – свой стиль, вместо заданного по умолчанию

? <link> – (непарный) ссылка

? <!--comment--> – комментарий, игнорируется браузером; используются при разработке; потом удаляют – меньше вес кода

**? MIME (Multipurpose Internet Mail Extension, Многоцелевые расширения почты Интернета)** — спецификация для передачи по сети файлов различного типа: изображений, музыки, текстов, видео, архивов и др.

? CSS — Cascading Style Sheets «каскадные таблицы стилей» — формальный **язык описания внешнего вида документа** (веб-страницы), написанного **с использованием** языка разметки (чаще всего HTML или XHTML). Также может применяться к любым XML-документам, например, к SVG или XUL

**? Синтаксис HTML**

* Принцип матрёшки тегов: **<p><b>**sample**</b></p>**
* Значения в начальном теге, а не в конечном: <body **xxxxxx**> </body>
* Значения атрибута в кавычках: <body style=**“background color: yellow”**>
* Вставка изображения – указывается ресурс: <img src=”……” alt=”text image description”>
* Тег и атрибут: <**img** **src**=”……”> – **img** – тег, **src** – атрибут
* <!DOCTYPE html> – ключевой эл-т, находится в самом начале
* Пустые теги <area>, <base>, <br>, <col>, <command>, <embed>, <hr>, <img>, <input>, <keygen>, <link>, <meta>, <param>, <source>, <track>, <wbr>
* <strong> – important – не только делает тест жирным, но и определяет его приоритет важности
* <b> –делает текст жирным
* <em> – emphases – не только делает тест курсивом, но и определяет его приоритет важности
* <i> –делает текст курсивом
* <a href=”https://www.link.com”>www.link.com</a> – ссылка-текст
* <a href=”https://www.link.com”><img src=”……”></a> – ссылка-картинка
* <a href=”mailto:xxxx@xxx.com”> xxxx@xxx.com </a> – ссылка на отправку почты
* <a name=”anchor”>anchor</a> – якорь – закладка внутри страницы, кот явл-ся целью ссылки
* <a href=”url” target=”FrameName”> – Имя окна или фрейма, куда браузер будет загружать эл-т
* <input type=”…..”> – позволяет создавать разные эл-ты интерфейса:
  + текстовое поле (text),
  + поле с паролем (password),
  + переключатель (radio),
  + флажок (checkbox),
  + скрытое поле (hidden),
  + кнопка (button),
  + кнопка для отправки формы (submit),
  + кнопка для очистки формы (reset),
  + поле для отправки файла (file)
  + кнопка с изображением (image)
* <select> – создаёт эл-т интерфейса в виде раскрывающегося списка и список с 1 из множества выборов
  + <select>
  + <option>Пункт 1</option>
  + <option>Пункт 2</option>
  + </select>
* <textarea …..атрибуты….> – Создание текстовой области, в кот можно вводить текст – абзацы будут учитываться! – в отличие от <input>
* <body …..атрибуты….> – эл-т для хранения содержания (контента) веб-страницы: текст, изображения, теги, скрипты JavaScript и т.д.
  + alink – Устанавливает цвет активной ссылки.
  + background – Задает фоновый рисунок на веб-странице.
  + bg=”background.gif” – Задает фоновый рисунок в виде картинки «background.gif»
  + bgcolor – Цвет фона веб-страницы.
  + bgproperties – Определяет, прокручивать фон совместно с текстом или нет.
  + bottommargin – Отступ от нижнего края окна браузера до контента.
  + leftmargin – Отступ по горизонтали от левого края окна браузера до контента.
  + link – Цвет ссылок на веб-странице.
  + rightmargin – Отступ от правого края окна браузера до контента.
  + scroll – Устанавливает, отображать полосы прокрутки или нет.
  + text – Цвет текста в документе.
  + topmargin – от верхнего края окна браузера до контента.
  + vlink – Цвет посещённых ссылок.
* <iframe> – создает плавающий фрейм, который находится внутри обычного документа, он позволяет загружать в область заданных размеров любые другие независимые документы
* <canvas> – Создает область, в которой при помощи JavaScript можно рисовать разные объекты, выводить изображения, анимацию, игры
* <nav>ссылки</nav> – задает навигацию по сайту. Если на странице несколько блоков ссылок, то в <nav> обычно помещают приоритетные ссылки. Также допустимо использовать несколько тегов <nav> в документе. Запрещается вкладывать <nav> внутрь <address>.
* <aside> – Определяет блок сбоку от контента для размещения рубрик, ссылок на архив, меток и другой информации. Такой блок, как правило, называется «сайдбар» или «боковая панель»
* <p> – абзац, добавляет пустой отступ перед первой строкой.
* <br> – перевод строки, НЕ добавляет пустой отступ перед первой строкой
* <h1>…<h6> - HTML предлагает шесть заголовков разного уровня, которые показывают относительную важность секции, расположенной после заголовка
* <hr> – Рисует горизонтальную линию
* <pre> – пре-форматируемый текст – сохраняются все пробелы (если подряд) и перевод на новую строку
* <div> – является блочным элементом (“контейнер”) и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей. Чтобы не описывать каждый раз стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей, а для тега добавить атрибут class или id с именем селектора  
  <div **class=”japan”**></div>, где **. japan** было определено как класс с параметрами
* <font> - тип шрифта, атрибуты:
  + color – Устанавливает цвет текста.
  + face – Определяет гарнитуру шрифта.
  + size – Задает размер шрифта в условных единицах
* <ul> - unordered list – ненумерованный список, атрибуты
  + <ul type=”disc”> – список, перечисленный с помощью <li> будет: • • •
  + <ul type=”circle”> – список, перечисленный с помощью <li> будет: ◦ ◦ ◦
* <ol> - ordered list – нумерованный список
  + <ol type=”1”> – список, перечисленный с помощью <li> будет: 1, 2, 3
  + <ol type=”I”> – список, перечисленный с помощью <li> будет: I, II, III
* <li> - list - перечисление пунктов после задания <ul> или <ol>
* <text style="padding-left: 60px"> - одинарный тег – ТАБУЛЯЦИЯ!

Атрибуты

* required – Устанавливает поле формы обязательным для заполнения перед отправкой формы на сервер. Если обязательное поле пустое, браузер выведет сообщение, а форма отправлена не будет
* range – позволяют определить числовое значение, которое должно быть в пределах указанного промежутка. Обычно – слайдер или контроллер,
* <meter value="значение">текст</meter> – Используется для вывода значения в некотором известном диапазоне. Используется преимущественно для отображения числовых значений

? События:

* onblur – событие возникающее при потере эл-та фокуса (кликнуть на другой эл-т)   
  <input type="text" value="Имя" onblur="alert(this.value)">
* onfocus – событие возникающее при получении эл-том фокуса (кликнуть на данный эл-т)  
  <input type="text" class="text" value="Введите текст" onfocus="this.value=''" onblur="this.value='Введите текст'">

**? Отличие XML от HTML**

**XML** не является заменой **HTML**. Они предназначены для решения разных задач: **XML** решает задачу хранения и транспортировки данных, фокусируясь на том, что такое эти самые данные, **HTML** же решает задачу отображения данных, фокусируясь на том, как эти данные выглядят.

**? Респонсив и Адаптивный дизайн**

Отзывчивый дизайн – один URL, адаптация вёрстки только за счёт CSS (у сайта 1 дизайн и он программно адаптируется под разные разрешения, мониторы)

Адаптивный дизайн – разные URL, адаптация вёрстки CSS, HTTP (у сайта много дизайнов, для разных мониторов, телефонов, …)

Сайты, основывающиеся на **отзывчивом (responsive) дизайне**, предоставляют всем устройствам одни и те же наборы URL, где каждый URL обеспечивает все устройства общим HTML кодом, поэтому модификация сайта под различные экраны достигается путем использования лишь CSS кода.

Сайты, сверстанные на **адаптивном (adaptive) веб-дизайне**, динамически сообщают всем устройствам общие URL, но каждый URL содержит различный HTML (и CSS) код, в зависимости от типа аппаратного обеспечения.

**? Браузерный движок (Layout Engine)** — представляет собой программу, преобразующую содержимое веб-страниц (файлы HTML, XML, цифровые изображения и т. д.) и информацию о форматировании (в форматах CSS, XSL и т. д.) в интерактивное изображение форматированного содержимого на экране. Браузерный движок обычно используется в веб-браузерах (отсюда название), почтовых клиентах и других программах, нуждающихся в отображении и редактировании содержимого веб-страниц.

**CSS**

**? CSS** – Стилем или CSS (Cascading Style Sheets, Каскадные Таблицы Стилей) называется **набор параметров форматирования**, который применяется к **элементам документа**, чтобы изменить их внешний вид. Еще одним преимуществом стилей является то, что они предлагают намного больше возможностей для форматирования, чем обычный HTML. CSS представляет собой мощную систему, расширяющую возможности дизайна и верстки веб-страниц.

? Сам код HTML никаких изменений не претерпел и единственное добавление — это строка   
<link **rel="stylesheet" href="style.css"**>. Она ссылается на внешний файл с описанием стилей под именем style.css. Содержимое этого файла:

body {

font-family: Arial, Verdana, sans-serif; /\* Семейство шрифтов \*/

font-size: 11pt; /\* Размер основного шрифта в пунктах \*/

background-color: #f0f0f0; /\* Цвет фона веб-страницы \*/

color: #333; /\* Цвет основного текста \*/

}

h1 {

color: #a52a2a; /\* Цвет заголовка \*/

font-size: 24pt; /\* Размер шрифта в пунктах \*/

font-family: Georgia, Times, serif; /\* Семейство шрифтов \*/

font-weight: normal; /\* Нормальное начертание текста \*/

}

p {

text-align: justify; /\* Выравнивание по ширине \*/

margin-left: 60px; /\* Отступ слева в пикселах \*/

margin-right: 10px; /\* Отступ справа в пикселах \*/

border-left: 1px solid #999; /\* Параметры линии слева \*/

border-bottom: 1px solid #999; /\* Параметры линии снизу \*/

padding-left: 10px; /\* Отступ от линии слева до текста \*/

padding-bottom: 10px; /\* Отступ от линии снизу до текста \*/

}

? CSS

<stile>

**.japan {**

**width: ….px;**

**height: ….px;**

**border: ….px solid #FFFFFF;**

**}**

</stile>

<body>

<div **class=”japan”**>…………..</div>

</body>

**? Преимущества CSS**

Поскольку на файл со стилем можно ссылаться из любого веб-документа, это приводит в итоге к сокращению объёма повторяющихся данных. А благодаря разделению кода и оформления повышается гибкость управления видом документа и скорость работы над сайтом

**? Типы стилей CSS**

* Стиль браузера
* Стиль автора
* Стиль пользователя

**? Способы добавления стилей на страницу:**

* Связанные стили (External - by using a <link> element to link to an external CSS file)
* Глобальные стили (Internal - by using a <style> element in the <head> section)
* Внутренние стили (Inline - by using the style attribute inside HTML elements)

**? Связанные стили (External):**

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением «css», а для связывания документа с этим файлом применяется тег <link>

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<title>Стили</title>

**<link rel="stylesheet" href="mysite.css">**

**<link rel="stylesheet" href="http://www.htmlbook.ru/main.css">**

</head>

<body>

……

Файл со стилем:

H1 {

color: #000080;

font-size: 200%;

font-family: Arial, Verdana, sans-serif;

text-align: center;

}

P {

padding-left: 20px;

}

? An external style sheet is used to define the style for many HTML pages.

To use an external style sheet, add a link to it in the <head> section of each HTML page

<!DOCTYPE html>

<html>

**<head>**

**<link rel="stylesheet" href="styles.css">**

**</head>**

<body>

……..

**? Глобальные стили (Internal):**

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы.

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<title>Глобальные стили</title>

**<style>**

**H1 {**

**font-size: 120%;**

**font-family: Verdana, Arial, Helvetica, sans-serif;**

**color: #333366;**

**}**

**</style>**

</head>

<body>

………

? An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the <head> section of an HTML page, within a <style> element

<!DOCTYPE html>

<html>

<head>

**<style>**

**body {background-color: powderblue;}**

**h1 {color: blue;}**

**p {color: red;}**

**</style>**

</head>

<body>

…….

**? Внутренние стили (Inline):**

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы.

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<title>Внутренние стили</title>

</head>

<body>

<p **style="font-size: 120%; font-family: monospace; color: #cd66cc"**>Пример текста</p>

…….

? Inline CSS is used to apply a unique style to a single HTML element.

<h1 **style="color:blue;"**>A Blue Heading</h1>

<p **style="color:red;"**>A red paragraph</p>

**? Синтаксис CSS**

Стилевые правила записываются в своём формате, отличном от HTML. Основным понятием выступает селектор — это некоторое имя стиля, для которого добавляются параметры форматирования. В качестве селектора выступают теги, классы и идентификаторы. Общий способ записи имеет следующий вид.

Базовый синтаксис CSS

Вначале пишется имя селектора, например, TABLE, это означает, что все стилевые параметры будут применяться к тегу <table>, затем идут фигурные скобки, в которых записывается стилевое свойство, а его значение указывается после двоеточия. Стилевые свойства разделяются между собой точкой с запятой, в конце этот символ можно опустить.

**? Селектор** — это некоторое имя стиля, для которого добавляются параметры форматирования

**? В качестве селектора выступают:**

* Теги
* Классы
* Идентификаторы

**? CSS не чувствителен** к регистру, переносу строк, пробелам и символам табуляции.

**? Комментарии** – применяют следующую конструкцию /\* ... \*/

**? Комментарии в HTML и CSS**

/\* комментарий \*/ – в CSS

<!-- комментарий --> – в HTML

**? Строки**

Любые строки необходимо брать в двойные или одинарные кавычки. Если внутри строки требуется оставить одну или несколько кавычек, то можно комбинировать типы кавычек или добавить перед кавычкой слеш:

'Гостиница "Турист"'

"Гостиница 'Турист'"

"Гостиница \"Турист\""

**? Числа**

Целочисленные 0, …, 9 и десятичные, разделяемые точкой (0.7 = .7)

**? Проценты**

Процентная запись обычно применяется в тех случаях, когда надо изменить значение относительно родительского элемента или когда размеры зависят от внешних условий. Так, ширина таблицы 100% означает, что она будет подстраиваться под размеры окна браузера и меняться вместе с шириной окна.

Проценты не обязательно должны быть целым числом, допускается использовать десятичные дроби, вроде значения 56.8%

**? Размеры**

Для задания размеров различных элементов, в CSS используются абсолютные и относительные единицы измерения. Абсолютные единицы не зависят от устройства вывода, а относительные единицы определяют размер элемента относительно значения другого размера.

* Относительные единицы обычно используют для работы с текстом, либо когда надо вычислить процентное соотношение между элементами. В табл. 6.1 перечислены основные относительные единицы.
  + em – размер шрифта текущего эл-та (зависит от размера шрифта текущего элемента)
  + ex – высота символа «х» в нижнем регистре (распространяются те же правила, что и для em: он привязан к размеру шрифта, заданного в браузере по умолчанию, или к размеру шрифта родительского элемента)
  + px – пиксель (это элементарная точка, отображаемая монитором или смартфоном, размер пикселя зависит от разрешения устройства и его технических х-к)
  + % – процент
* Абсолютные единицы применяются реже, чем относительные и обычно при работе с текстом.
  + in – Дюйм (1 in = 2,54 cm)
  + cm – Сантиметр
  + mm– Миллиметр
  + pt – Пункт (1 pt = 1/72 in)
  + pc – Пика (1 pc = 12 pt = 1/6 in)

**? Цвет**

Цвет в стилях можно задавать тремя способами:

* по шестнадцатеричному значению,
* по названию и
* в формате RGB

По шестнадцатеричному значению:

Каждый из трех цветов — красный, зеленый и синий — может принимать значения от 00 до FF. Таким образом, обозначение цвета разбивается на три составляющие #rrggbb, где первые два символа отмечают красную компоненту цвета, два средних — зелёную, а два последних — синюю. Допускается использовать сокращенную форму вида #rgb, где каждый символ следует удваивать (#rrggbb). К примеру, запись #fe0 расценивается как #ffee00

По названию:

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Цвет** | **Код** | **Описание** |
| white |  | #ffffff или #fff | Белый |
| silver |  | #c0c0c0 | Серый |
| gray |  | #808080 | Тёмно-серый |
| black |  | #000000 или #000 | Чёрный |
| maroon |  | #800000 | Тёмно-красный |
| red |  | #ff0000 или #f00 | Красный |
| orange |  | #ffa500 | Оранжевый |
| yellow |  | #ffff00 или #ff0 | Жёлтый |
| olive |  | #808000 | Оливковый |
| lime |  | #00ff00 или #0f0 | Светло-зелёный |
| green |  | #008000 | Зелёный |
| aqua |  | #00ffff или #0ff | Голубой |
| blue |  | #0000ff или #00f | Синий |
| navy |  | #000080 | Тёмно-синий |
| teal |  | #008080 | Сине-зелёный |
| fuchsia |  | #ff00ff или #f0f | Розовый |
| purple |  | #800080 | Фиолетовый |

В формате RGB

Вначале указывается ключевое слово rgb, а затем в скобках, через запятую указываются компоненты цвета, например rgb(255, 0, 0) или rgb(100%, 20%, 20%)

**? Адреса**

Адреса (URI, Uniform Resource Identifiers, унифицированный идентификатор ресурсов) применяются для указания пути к файлу, например, для установки фоновой картинки на странице. Для этого применяется ключевое слово url(), внутри скобок пишется относительный или абсолютный адрес файла. При этом адрес можно задавать в необязательных одинарных или двойных кавычках

**? Ключевые слова**

В качестве значений активно применяются ключевые слова, которые определяют желаемый результат действия стилевых свойств. Ключевые слова пишутся без кавычек.

Правильно: P { text-align: right; }

НЕВЕРНО: P { text-align: "right"; }

**? Селекторы тегов**

В качестве селектора может выступать любой тег HTML, для которого определяются правила форматирования, такие как: цвет, фон, размер и т. д. Правила задаются в следующем виде.

Тег { свойство1: значение; свойство2: значение; ... }

Вначале указывается имя тега, оформление которого будет переопределено, заглавными или строчными символами не имеет значения. Внутри фигурных скобок пишется стилевое свойство, а после двоеточия — его значение. Набор свойств разделяется между собой точкой с запятой и может располагаться как в одну строку, так и в несколько

**? Классы**

Классы применяют, когда необходимо определить стиль для индивидуального элемента веб-страницы или задать разные стили для одного тега. При использовании совместно с тегами синтаксис для классов будет следующий.

Тег.Имя класса { свойство1: значение; свойство2: значение; ... }

Внутри стиля вначале пишется желаемый тег, а затем, через точку пользовательское имя класса. Имена классов должны начинаться с латинского символа и могут содержать в себе символ дефиса (-) и подчеркивания (\_). Использование русских букв в именах классов недопустимо. Чтобы указать в коде HTML, что тег используется с определённым классом, к тегу добавляется атрибут class="Имя класса"

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Классы**</title>**

**<style>**

**P** { /\* Обычный абзац \*/

text-align: justify; /\* Выравнивание текста по ширине \*/

}

**P**.cite { /\* Абзац с классом cite \*/

color: navy; /\* Цвет текста \*/

margin-left: 20px; /\* Отступ слева \*/

border-left: 1px solid navy; /\* Граница слева от текста \*/

padding-left: 15px; /\* Расстояние от линии до текста \*/

}

**</style>**

**</head>**

**<body>**

**<p>**Для искусственного освещения помещения применяются люминесцентные лампы.

Они отличаются высокой световой отдачей, продолжительным сроком службы,

малой яркостью светящейся поверхности, близким к естественному спектральным

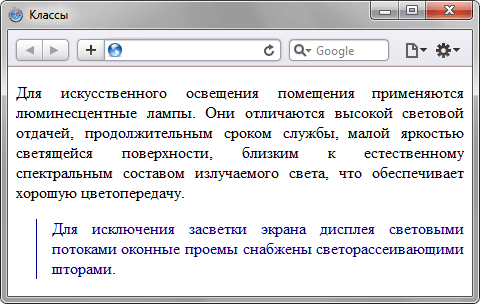
составом излучаемого света, что обеспечивает хорошую цветопередачу.**</p>**

**<p** class="cite"**>**Для исключения засветки экрана дисплея световыми потоками

оконные проемы снабжены светорассеивающими шторами.**</p>**

**</body>**

**</html>**



Можно, также, использовать классы и без указания тега. Синтаксис в этом случае будет следующий.

.Имя класса { свойство1: значение; свойство2: значение; ... }

При такой записи класс можно применять к любому тегу

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Классы**</title>**

**<style>**

.gost {

color: green; /\* Цвет текста \*/

font-weight: bold; /\* Жирное начертание \*/

}

.term {

border-bottom: 1px dashed red; /\* Подчеркивание под текстом \*/

}

**</style>**

**</head>**

**<body>**

**<p>**Согласно **<span** class="gost"**>**ГОСТ 12.1.003-83 ССБТ &quot;Шум. Общие

требования безопасности&quot;**</span>**, шумовой характеристикой рабочих

мест при постоянном шуме являются уровни звуковых давлений в децибелах

в октавных полосах. Совокупность таких уровней называется

**<b** class="term"**>**предельным спектром**</b>**, номер которого численно равен

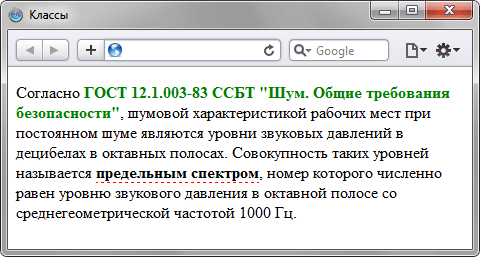
уровню звукового давления в октавной полосе со среднегеометрической

частотой 1000&nbsp;Гц.

**</p>**

**</body>**

**</html>**



? Идентификаторы

Идентификатор (называемый также «ID селектор») определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты.

Синтаксис применения идентификатора следующий.

#Имя идентификатора { свойство1: значение; свойство2: значение; ... }

При описании идентификатора вначале указывается символ решётки (#), затем идет имя идентификатора. Оно должно начинаться с латинского символа и может содержать в себе символ дефиса (-) и подчеркивания (\_). Использование русских букв в именах идентификатора недопустимо. В отличие от классов идентификаторы должны быть уникальны, иными словами, встречаться в коде документа только один раз.

Обращение к идентификатору происходит аналогично классам, но в качестве ключевого слова у тега используется атрибут id, значением которого выступает имя идентификатора (пример 9.1). Символ решётки при этом уже не указывается.

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Идентификаторы**</title>**

**<style>**

#help {

position: absolute; /\* Абсолютное позиционирование \*/

left: 160px; /\* Положение элемента от левого края \*/

top: 50px; /\* Положение от верхнего края \*/

width: 225px; /\* Ширина блока \*/

padding: 5px; /\* Поля вокруг текста \*/

background: #f0f0f0; /\* Цвет фона \*/

}

**</style>**

**</head>**

**<body>**

**<div** id="help"**>**

Этот элемент помогает в случае, когда вы находитесь в осознании того

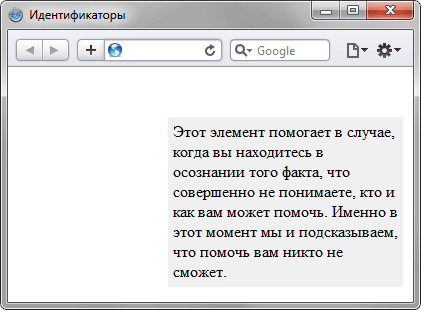
факта, что совершенно не понимаете, кто и как вам может помочь. Именно

в этот момент мы и подсказываем, что помочь вам никто не сможет.

**</div>**

**</body>**

**</html>**



**Data Bases**

? Базы данных в жизни – Библиотека, записная книжка, …

? Базы данных на ПК – Access, Oracle, …

? Базы данных:

* Иерархические – минус в том, что если надо найти эл-т на нижнем, надо двигаться сверху-вниз
* Сетевые
* Объектно-ориентированные

ЛЮДИ

ИНЖЕНЕРЫ

QA

…….

…….

* Реляционные

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Строка = Запись = Кортеж

Столбец = Колонка = Поле

? Реляционные и нереляционные БД – В отличие от SQL, в NoSQL вся информация хранится без четко установленной структуры и явных связей между всеми данными.

Реляционные БД хранят структурированные данные, которые обычно представляют объекты реального мира. Это могут быть сведения о человеке, или о содержимом корзины для товаров в магазине, сгруппированные в таблицах, **формат которых задан** на этапе проектирования хранилища.  
  
Нереляционные БД устроены иначе. Например, документо-ориентированные базы хранят информацию в виде иерархических структур данных. Речь может идти об объектах с произвольным набором атрибутов. То, что в реляционной БД будет разбито на несколько взаимосвязанных таблиц, в нереляционной может храниться в виде целостной сущности.

? СУБД и БД

БД – база данных

СУБД – система управления базами данных

* Oracle
* MS SQL Server
* MySQL
* MS Access

Не все системы, позиционирующие себя как СУБД, таковыми являются.

Должны выполняться 12 правил Кодда (Эдгара Кодда)

? **12 правил Кодда (Эдгара Кодда) / (Codd’s 12 rules)** (на самом деле их 13: Правило 0, 1, 2, …, 12) – 13 правил, которым должна удовлетворять каждая система управления **реляционными** базами данных.

**Правило 0: Основное правило (Foundation Rule):**

Система, которая рекламируется или позиционируется как реляционная система управления базами данных, должна быть способной управлять базами данных, используя исключительно свои реляционные возможности.

**Правило 1: Информационное правило (The Information Rule):**

Вся информация в реляционной базе данных на логическом уровне должна быть явно представлена единственным способом: значениями в таблицах.

**Правило 2: Гарантированный доступ к данным (Guaranteed Access Rule):**

В реляционной базе данных каждое отдельное (атомарное) значение данных должно быть логически доступно с помощью комбинации имени таблицы, значения первичного ключа и имени столбца.

**Правило 3: Систематическая поддержка отсутствующих значений (Systematic Treatment of Null Values):**

Неизвестные, или отсутствующие значения NULL, отличные от любого известного значения, должны поддерживаться для всех типов данных при выполнении любых операций. Например, для числовых данных неизвестные значения не должны рассматриваться как нули, а для символьных данных — как пустые строки.

**Правило 4: Доступ к словарю данных в терминах реляционной модели (Active On-Line Catalog Based on the Relational Model):**

Словарь данных должен сохраняться в форме реляционных таблиц, и СУБД должна поддерживать доступ к нему при помощи стандартных языковых средств, тех же самых, которые используются для работы с реляционными таблицами, содержащими пользовательские данные.

**Правило 5: Полнота подмножества языка (Comprehensive Data Sublanguage Rule):**

Система управления реляционными базами данных должна поддерживать хотя бы один реляционный язык, который

(а) имеет линейный синтаксис,

(б) может использоваться как интерактивно, так и в прикладных программах,

(в) поддерживает операции определения данных, определения представлений, манипулирования данными (интерактивные и программные), ограничители целостности, управления доступом и операции управления транзакциями (begin, commit и rollback).

**Правило 6: Возможность изменения представлений (View Updating Rule):**

Каждое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции выборки, вставки, изменения и удаления данных.

**Правило 7: Наличие высокоуровневых операций управления данными (High-Level Insert, Update, and Delete):**

Операции вставки, изменения и удаления данных должны поддерживаться не только по отношению к одной строке реляционной таблицы, но и по отношению к любому множеству строк.

**Правило 8: Физическая независимость данных (Physical Data Independence):**

Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютеров, на которых находится реляционная база данных.

**Правило 9: Логическая независимость данных (Logical Data Independence):**

Представление данных в приложении не должно зависеть от структуры реляционных таблиц. Если в процессе нормализации одна реляционная таблица разделяется на две, представление должно обеспечить объединение этих данных, чтобы изменение структуры реляционных таблиц не сказывалось на работе приложений.

**Правило 10: Независимость контроля целостности (Integrity Independence):**

Вся информация, необходимая для поддержания целостности, должна находиться в словаре данных. Язык для работы с данными должен выполнять проверку входных данных и автоматически поддерживать целостность данных.

**Правило 11: Независимость от расположения (Distribution Independence):**

База данных может быть распределённой, может находиться на нескольких компьютерах, и это не должно оказывать влияния на приложения. Перенос базы данных на другой компьютер не должен оказывать влияния на приложения.

**Правило 12: Согласование языковых уровней (The Nonsubversion Rule):**

Если используется низкоуровневый язык доступа к данным, он не должен игнорировать правила безопасности и правила целостности, которые поддерживаются языком более высокого уровня.

**? Терминология названия строк и столбцов:**

Строка = Запись = Кортеж

Столбец = Колонка = Поле

**? Для того, чтобы получить данные из БД надо знать 3 параметра:**

1. Имя таблицы
2. Название столбца
3. Ключ = связь

**? Синтаксис – используются /не используются в название Таблицы, Столбца, Ключа**

Никогда не используются – Кириллица (и др.) и Пробелы

Используются – Латиница и «\_»/сочетания «CapsSmall»

? Виды реляции (отношения) :

* 1 к 1
* 1 ко многим
* Многие ко многим

? Введение реляции

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Students | | | |  |  | Pers\_data | | | | |  |
| ID | Name | Age | Mark |  |  | ID | City | Address | Phone | Student\_id |  |
| 2145 | Julia | 23 | 98% |  |  | 1 | Odessa | Shevchenko ave, # | +38067xxxxxxx | 2145 |  |
| 18 | Oleg | 35 | 67% |  |  | 2 | Texas | Sunnymoon rd, # | +38068xxxxxxx | 18 |  |
| 35 | Maria | 29 | 80% |  |  | 3 | Lviv | Lesi Ukraiinki blvd | +38093xxxxxxx | 35 |  |
| 60 | Julia | 23 | 50% |  |  | 4 | Odessa | Levitana str, # | +38095xxxxxxx | 60 |  |
| … | … | … | … |  |  | … | … | … | … | … |  |

* Есть таблица со студентами, возрастом и оценками.
* Появляется студент с таким же именем – Юлия.
* Выход – ввести фамилию, но фамилии тоже могут быть одинаковыми.
* Выход – различать Юль по возрасту, но и возраст может совпадать.
* Тогда надо смотреть на оценки, но это уже становится сложным, да и они могут совпасть.
* Выход – вводим уникальный id.
* Тут id – Первичный ключ (Primary Key)
* Но есть много сопутствующих данных (Персональные данные), которые или в данный момент не нужны, или доступ к ним открыт не для всех (Перс данные в банковских приложениях)
* Создание общей дополнительной расширенной таблицы – не вариант – лишние дублирующиеся данные занимают память, и это не решает проблему доступа к части данных.
* Тогда создаётся таблица «Перс данные» и устанавливается связь с таблицей «Студенты» по уникальному идентификатору. Эта связь и есть – реляция.
* В таблице «Pers\_data» есть столбец «Student\_id» (это Внешний ключ), который указывает на таблицу «Students», столбик «ID» (это Первичный ключ)
* Students.ID – Первичный ключ (Primary Key)
* Pers\_data. Student\_id – Внешний ключ (Foreign Key)

**? Нормальные формы**

Нормальная форма (форма нормализации) – оптимизированная форма, без лишнего / совокупность требований, которым должно удовлетворять отношение

1ая нормальная форма (1NF) – говорит о том, что данные не должны повторяться

2ая нормальная форма (2NF) – данные находятся в 1NF и зависят от ключа

3ая нормальная форма (3NF) – данные находятся в 2NF и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых

Нормальная форма Бойса — Кодда (BCNF)

4ая нормальная форма (4NF)

5ая нормальная форма (5NF)

Доменно-ключевая нормальная форма (DKNF)

6ая нормальная форма (6NF)

**? Атомарность значений**

Значения в ячейках должны быть атомарными – не валить несколько значений в одну ячейку.

ФИО – не является атомарным. Применяется принцип декомпозиции: ФИО 🡪 Ф, И, О

**? Однотипные данные**

Все данные должны быть одного типа

**? Реляции**

Реляции (зависимости) должны выноситься в новые таблицы, а не валить в одну

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | |  | Auto | | | |  |
| ID | Maker | | | Model | | | Price | ~~Supplier~~ | ~~Discount~~ | | |  | Supplier | Discount |  |
| 1 | Mercedes | | | S500 | | | 20000 | ~~Adis~~ | ~~5%~~ | | |  | Adis | 5% |
| 2 | Audi | | | T-8 | | | 18000 | ~~Ital~~ | ~~7%~~ | | | 🡪 | Ital | 7% |
| 3 | Audi | | | T-10 | | | 22000 | ~~Genner~~ | ~~10%~~ | | |  | Genner | 10% |
| 4 | Toyota | | | CHR | | | 25000 | ~~Ital~~ | ~~7%~~ | | |  | Ital | 7% |
| … | … | | | … | | | … | … | … | | |  | … | … |

**SQL**

**? SQL (Structured Query Language)** — «язык структурированных запросов» — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

**? Синтаксис SQL**

**ВЫБОРКА**

SELECT – данные, кот надо выбрать

FROM – таблица из которой надо выбрать

WHERE – параметры по которым надо выбрать

SELECT maker, make, model FROM auto WHERE id>2

Automarket.Auto.Id – База.Таблица.Столбец

!= или <> – не равно

; - в конце строки – классика, но сейчас опционально

\* – выбрать всё – SELECT \* FROM auto – выводит все значения из табл «auto»

AND, OR – и, или – AND – значения обоих критериев вместе, OR – значения каждого критерия

SELECT \* FROM auto WHERE id>2 AND make=”BMW” – только те значения где («id>2» вместе с «BMW»)

SELECT \* FROM auto WHERE id>2 OR make=”BMW” – и те значения где «id>2» + и те где «BMW»

IN – множество перечисленных критериев

… ~~WHERE id=2 AND id=3 AND id=4 AND id=7~~

… WHERE id IN (2, 3, 4, 7) – выбор значений с id=2, id=3, id=4, id=7

BETWEEN … AND … – множество между перечисленными критериями

… WHERE id BETWEEN 300 AND 685 – выбор значений с id=300, id=301, id=302, …, id=685

LIKE … – выбор удовлетворяющий значению, как «=»

… WHERE maker LIKE “Volkswagen”

% – любое кол-во и значение символов в слове-запросе (если не помним/не важна часть слова-запроса)

… WHERE maker LIKE “%agen” – выдаст Volkswagen (и даже если вдруг есть и Falxvagen, и 154wagen)

\_ – кол-во символов и их порядок при выборке

\_ \_ \_ – 3 символа

\_о% – 1ая буква одна, 2ая буква «о», дальше сколько угодно любых символов

%tori% – всё что угодно, содержащее «…tori…» в любой позиции

% % % – всё содержащее 2 пробела: «World Wide Web»

%a%d – слово, содержащее буквы «a» и «d»

AVG (…) – среднее арифметическое

SELECT AVG (Price) FROM auto – выведет среднее значение «Price», н-р: «21250»

SUM (…) – сумма

SELECT SUM (Price) FROM auto – выведет суммарное значение «Price», н-р: «85000»

MIN (…) – минимальное значение

SELECT MIN (Price) FROM auto – выведет минимальное значение «Price», н-р: «18000»

MAX (…) – максимальное значение

SELECT MAX (Price) FROM auto – выведет максимальное значение «Price», н-р: «25000»

COUNT (…) – счёт по значению

SELECT COUNT (Price) FROM auto – выведет кол-во строк, где «Price» имеет значение (кроме NULL), н-р: «24»

COUNT (\*) – счёт строк

SELECT COUNT (\*) FROM auto – выведет полное кол-во строк (не взирая на NULL), н-р: «25»

TOP … – первые … строк

SELECT TOP 5 FROM auto – выведет первые 5 строк

DISTINCT – исключает одинаковые значения

SELECT DISTINCT model FROM auto – выведет «model» только с уникальными значениями (без повторений)

EXCEPT … – выбор исключающий значение, как «≠»

EXCEPT SELECT model FROM auto WHERE maker LIKE “Volkswagen”

**ПСЕВДОНИМЫ (ALIASES)**

AS – присваивает наименование выводимому значению ИЛИ псевдоним колонке, таблице

SELECT AVG (Price) FROM auto – выведет «21250»

SELECT AVG (Price) AS Average\_Price FROM auto – выведет «Average\_Price 21250»

SELECT AVG (Price) FROM auto – будет использоваться оригинальное имя таблицы «auto»

SELECT AVG (Price) FROM auto AS a – будет использоваться имя таблицы «а» вместо «auto»

**СОРТИРОВКА**

ORDER BY – сортировка заданным образом

ASC – сортировка по возрастанию

DESC – сортировка по убыванию

SELECT \* FROM auto ORDER BY id DESC – выведет все строки из «auto», отсортирует по убыванию «id»

**ГРУППИРОВКА**

GROUP BY – создаёт сводную таблицу, сгруппированную из повторяющихся полей

SELECT COUNT (Model) AS Qty, Maker FROM auto GROUP BY Maker

|  |  |
| --- | --- |
| Maker | Qty |
| Mercedes | 1 |
| Audi | 2 |
| Toyota | 1 |

**РАЗГРУППИРОВКА**

FROM – используется в одной таблице для учёта одного столбца дважды

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Computers | | | | |
| id | maker | model | speed | ram |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Найти пары моделей ПК, имеющих одинаковую скорость и оперативку.

Каждая пара указывается только один раз, т.е. из эл-тов «i» и «j» указать пару (i–j), а пару (j–i) – нет.

SELECT DISTINCT A.model, B.model, A.speed, A.ram /вводим разбиение на «А» и «В»/

FROM pc A, pc B /разбиваем таблицу на «А» и «В»/

WHERE A.speed=B.speed AND A.ram=B.ram AND A.model>B.model

**ФИЛЬТРАЦИЯ ГРУППИРОВКИ**

HAVING используется после GROUP BY для фильтрации по значениям агрегатных функций из групп полученных строк (а не из отдельных строк как в агрегатных функциях после WHERE)

SELECT model, COUNT(model) AS Qty\_model, AVG(price) AS Avg\_price

FROM pc

GROUP BY model

HAVING AVG(price) < 800;

|  |  |  |
| --- | --- | --- |
| model | Qty\_model | Avg\_price |
| 1260 | 1 | 350 |
| 1232 | 4 | 425 |

Найдите размеры жестких дисков, совпадающих у двух и более PC.

SELECT hd

FROM pc

GROUP BY hd

HAVING COUNT(hd)>1

**? Порядок запросов**

1. SELECT ( AVG SUM MIN MAX COUNT DISTINCT TOP AS )
2. FROM ( AS )
3. JOIN table ON key
4. WHERE ( AND OR BETWEEN LIKE % )
5. GROUP BY
6. HAVING
7. ORDER BY ( ASC DESC )

**ОБЪЕДИНЕНИЕ ТАБЛИЦ**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Auto | | | |  | Characteristics | | |  | Showroom | | | |
| Id | Maker | Model | Price |  | Id | Color | Auto\_Id |  | Id | Discount % |  | Caract-Id |
| 1 | Mercedes | S500 | 20000 |  | 1003 | Red | 1 |  | 557 | 5 |  | 1003 |
| 2 | Audi | T-8 | 18000 | 🡪 | 1004 | Green | 2 | 🡪 | 558 | 7 |  | 1004 |
| 3 | Audi | T-10 | 22000 |  | 1005 | Red | 3 |  | 559 | 12 |  | 1005 |
| 4 | Toyota | CHR | 25000 |  | 1006 | Black | 4 |  | 560 | 9 |  | 1006 |
| … | … | … | … |  | … | … | … |  | … |  |  | … |

Нужно выбрать машины зелёного цвета.

Но «Maker» и «Color» находятся в разных таблицах.

SELECT Maker FROM Auto WHERE Color=”Green” – не пройдёт: Maker и Color в разных табл

Есть 3 способа объединения таблиц

**I СПОСОБ – Через AND**

SELECT Maker FROM Auto, Characteristics WHERE Color=”Green” AND Auto.Id=Charachteristics.Auto\_Id

или

SELECT Maker FROM Auto AS a, Characteristics AS c WHERE Color=”Green” AND a.Id=c.Auto\_Id

**II СПОСОБ – Подзапрос**

SELECT Maker FROM Auto WHERE Id IN (SELECT Auto-Id FROM Characteristics WHERE Color=”Green”)

**III СПОСОБ – Через JOIN**

JOIN – объединение таблиц

SELECT Maker

FROM Auto

JOIN Characteristics ON Auto.Id=Charachteristics.Auto-Id

JOIN Showroom ON Charachteristics.Id=Showroom.Characht-Id

WHERE Discount > 10

🡪 Связь №1 – таблица «Auto» связывается с «Characteristics» по ключу Auto.Id = Charachteristics.Auto-Id

🡪 Связь №2 – таблица «Characteristics» связывается с «Showroom» по ключу Charachteristics.Id = Showroom.Characht-Id

SELECT a.model, price

FROM( SELECT model, price FROM pc

UNION

SELECT model, price FROM laptop

UNION

SELECT model, price FROM printer)

AS a

JOIN product ON a.model=product.model

WHERE product.maker=’B’

**? различие между JOIN и UNION**

[**UNION**](http://msdn.microsoft.com/en-us/library/ms180026.aspx) объединяет результаты двух или более запросов в единый результирующий набор, включающий все строки, принадлежащие всем запросам в союзе.

Используя [**JOINs**](http://msdn.microsoft.com/en-us/library/ms191517.aspx), вы можете извлекать данные из двух или более таблиц на основе логических отношений между таблицами. Соединения указывают, как SQL должен использовать данные из одной таблицы для выбора строк в другой таблице.

**? Виды JOIN и механика их формирования**

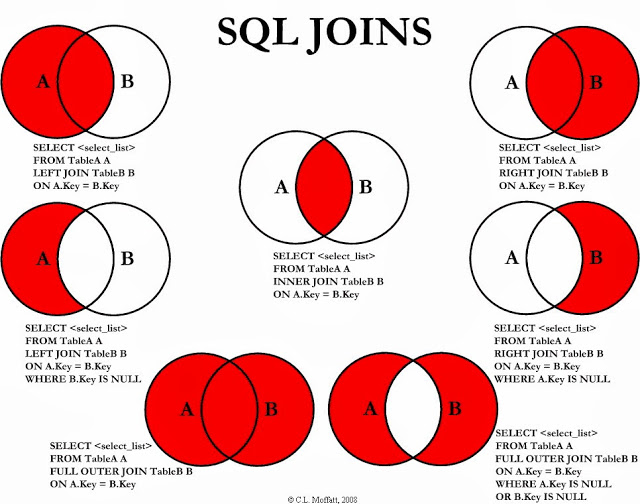
JOIN = INNER JOIN – результаты пересечения (совпадения) двух таблиц

LEFT JOIN – результаты пересечения двух таблиц + данные из левой таблицы

RIGHT JOIN – результаты пересечения двух таблиц + данные из правой таблицы

FULL OUTER JOIN – результаты пересечения, правой таблицы, левой таблицы, (где не определено – NULL)

CROSS JOIN – формируется таблица, где группируются каждое с каждым, (где не определено – NULL)



**? Как БД понимает какая таблица левая, а какая правая в JOIN-ах:**

Относительно того с какой стороны от знака «=» мы прописываем таблицу

**? CROSS JOIN** – Декартово произведение двух таблиц

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Authors | |  | Books | |
| A\_id | A-name |  | B\_id | B-name |
| 1 | Cooper |  | 28 | Wood |
| 2 | Jason |  | 29 | Land |

SELECT \* FROM Authors CROSS JOIN Books

|  |  |  |  |
| --- | --- | --- | --- |
| A\_id | A-name | B\_id | B-name |
| 1 | Cooper | 28 | Wood |
| 1 | Cooper | 29 | Land |
| 2 | Jason | 28 | Wood |
| 2 | Jason | 29 | Land |

? FULL JOIN – Оператор FULL JOIN можно воспринимать как сочетание операторов [INNER JOIN](http://2sql.ru/advanced/sql-inner-join/) + [LEFT JOIN](http://2sql.ru/advanced/sql-left-join/) + [RIGHT JOIN](http://2sql.ru/advanced/sql-right-join/). Алгоритм его работы следующий:

* Сначала формируется таблица на основе внутреннего соединения (оператор [INNER JOIN](http://2sql.ru/advanced/sql-inner-join/)).
* Затем, в таблицу добавляются значения, **не** **вошедшие** в результат формирования из правой таблицы ([LEFT JOIN](http://2sql.ru/advanced/sql-left-join/)). Для них, соответствующие записи из правой таблицы заполняются значениями NULL.
* Наконец, в таблицу добавляются значения, **не** **вошедшие** в результат формирования из левой таблицы ([RIGHT JOIN](http://2sql.ru/advanced/sql-right-join/)). Для них, соответствующие записи из левой таблицы заполняются значениями NULL.

**? Типы данных SQL**

При создании таблицы для всех ее столбцов необходимо указать определенный тип данных. Тип данных определяет, какие значения могут храниться в столбце, сколько они будут занимать места в памяти.

В зависимости от характера значений все их можно разделить на группы.

**ЧИСЛОВЫЕ ТИПЫ ДАННЫХ**

* BIT: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.
* TINYINT: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.
* SMALLINT: хранит числа от –32 768 до 32 767. Занимает 2 байта
* INT: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.
* BIGINT: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.
* DECIMAL: хранит числа c фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой. Данный тип может принимать два параметра precision и scale: DECIMAL(precision, scale). Параметр precision представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18. Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0.
* NUMERIC: данный тип аналогичен типу DECIMAL.
* SMALLMONEY: хранит дробные значения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу DECIMAL(10,4).
* MONEY: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу DECIMAL(19,4).
* FLOAT: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части. Может иметь форму определения в виде FLOAT(n), где n представляет число бит, которые используются для хранения десятичной части числа (мантиссы). По умолчанию n = 53.
* REAL: хранит числа от –340E+38 to 3.40E+38. Занимает 4 байта. Эквивалентен типу FLOAT(24).

Примеры числовых столбцов:

1 Salary MONEY,

2 TotalWeight DECIMAL(9,2),

3 Age INT,

4 Surplus FLOAT

**ДАТА И ВРЕМЯ**

* DATE: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.
* TIME: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт. Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.
* DATETIME: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.
* DATETIME2: хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Занимает от 6 до 8 байт в зависимости от точности времени. Может иметь форму DATETIME2(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.
* SMALLDATETIME: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.
* DATETIMEOFFSET: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

Распространенные форматы дат:

yyyy-mm-dd - 2017-07-12

dd/mm/yyyy - 12/07/2017

mm-dd-yy - 07-12-17 В таком формате двузначные числа от 00 до 49 воспринимаются как даты в диапазоне 2000-2049. А числа от 50 до 99 как диапазон чисел 1950 - 1999.

Month dd, yyyy - July 12, 2017

Распространенные форматы времени:

hh:mi - 13:21

hh:mi am/pm - 1:21 pm

hh:mi:ss - 1:21:34

hh:mi:ss:mmm - 1:21:34:12

hh:mi:ss:nnnnnnn - 1:21:34:1234567

**СТРОКОВЫЕ**

* CHAR: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode. Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.
* VARCHAR: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX). Не подходит для многих языков, так как хранит символы не в кодировке Unicode. В отличие от типа CHAR если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столбце будет сохранено именно пять символов.
* NCHAR: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15)
* NVARCHAR: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта. Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.
* Еще два типа TEXT и NTEXT являются устаревшими, и поэтому их не рекомендуется использовать. Вместо них применяются VARCHAR и NVARCHAR соответственно.

Примеры определения строковых столбцов:

1 Email VARCHAR(30),

2 Comment NVARCHAR(MAX)

**БИНАРНЫЕ ТИПЫ ДАННЫХ**

* BINARY: хранит бинарные данные в виде последовательности от 1 до 8 000 байт.
* VARBINARY: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до 2^31–1 байт при использовании значения MAX (VARBINARY(MAX)).
* Еще один бинарный тип - тип IMAGE является устаревшим, и вместо него рекомендуется применять тип VARBINARY.

**ОСТАЛЬНЫЕ ТИПЫ ДАННЫХ**

* UNIQUEIDENTIFIER: уникальный идентификатор GUID (по сути строка с уникальным значением), который занимает 16 байт.
* TIMESTAMP: некоторое число, которое хранит номер версии строки в таблице. Занимает 8 байт.
* CURSOR: представляет набор строк.
* HIERARCHYID: представляет позицию в иерархии.
* SQL\_VARIANT: может хранить данные любого другого типа данных T-SQL.
* XML: хранит документы XML или фрагменты документов XML. Занимает в памяти до 2 Гб.
* TABLE: представляет определение таблицы.
* GEOGRAPHY: хранит географические данные, такие как широта и долгота.
* GEOMETRY: хранит координаты местонахождения на плоскости.

**? Число в скобках** – НЕ кол-во символа, а размер в байтах!

**DML**

? DML – это короткое имя языка манипулирования данными, которое связано с манипулированием данными и включает в себя наиболее распространенные операторы SQL, такие как SELECT, INSERT, UPDATE, DELETE и т.д., и используется для хранения, изменения, извлечения, удаления и обновления данных в базе данных.

* SELECT – получить данные из базы данных
* INSERT – вставить данные в таблицу
* UPDATE – обновляет существующие данные в таблице
* DELETE – удалить записи из таблицы
* MERGE – операция UPSERT (вставить или обновить)
* CALL – вызвать подпрограмму PL / SQL или Java
* EXPLAIN PLAN – интерпретация пути доступа к данным
* LOCK TABLE – контроль параллелизма

**СОЗДАНИЕ**

CREATE DATABASE AutoMarket

BACKUP DATABASE AutoMarket TO DISC F:/…../……

~~CREATE TABLE Auto~~ – НЕ СРАБОТАЕТ! – пустая таблица

CREATE TABLE Auto (id int PRIMARY KEY NOT NULL, Maker varchar(30), Make varchar(30), Model varchar(30))

**УДАЛЕНИЕ**

DROP – для удаление Баз и Таблиц

DELETE – для удаление Строк

DROP DATABASE AutoMarket – удаление Базы

DROP TABLE Auto – удалить таблицу

Удалить из таблицы строки с определёнными характеристиками

DELETE

FROM pc

WHERE hd=(SELECT MIN(hd) FROM pc) OR ram=(SELECT MIN(ram) FROM pc)

Удалить из таблицы «А», удовлетворяющие условиям из таблицы «В»

DELETE

FROM Ships

WHERE name IN (SELECT ship FROM outcomes WHERE result='sunk')

**ИЗМЕНЕНИЕ**

ALTER – существенные изменения в таблице, н-р новые столбцы

ALTER TABLE Auto ADD Price money

UPDATE – изменение на уровне данных

UPDATE (TABLE) Auto SET Maker=”ВАЗ” WHERE ID=7

Вместе с UPDATE надо обязательно указать WHERE, а то перезапишется (т.е. удалится) всё!

Производитель «A» передал производство принтеров производителю «Z»

UPDATE product SET maker = 'Z' WHERE maker='A' AND type='printer'

**ДОБАВЛЕНИЕ**

INSERT INTO – вставить значения, сначала указывается Столбец, потом значение

INSERT INTO Auto (ID, Maker, Make, Model) VALUES (11, Toyota, Toyota, Auris) – данные не чётко соответствуют структуре, н-р для всех остальных машин есть ещё значения для колонки «Price»

или

INSERT INTO Auto VALUES (11, Toyota, Toyota, Auris) – данные чётко соответствуют структуре

INSERT INTO product VALUES ('Z', 4001, 'pc'), ('Z', 4002, 'laptop'), ('Z', 4003, 'printer') – Сразу несколько строк

INSERT INTO pc VALUES (22, 4444, 1200, DEFAULT, DEFAULT, DEFAULT, 1350) – не все данные известны

Вставить из одной таблицы в другую, значение «cd» (…, ram, hd, **cd**, price) по умолчанию – (пропускаем), каждый параметр изменён определённым образом

INSERT INTO pc (code, model, speed, ram, hd, price)

SELECT

MIN(code) + 20,

model + 1000,

MAX(speed),

MAX(ram) \*2,

MAX(hd) \*2,

MAX(price)/1.5

FROM laptop

GROUP BY model

**Представление,**

**Пользовательская функция,**

**Хранимая Процедура,**

**Триггер**

**? ПРЕДСТАВЛЕНИЕ**

Представление – виртуальная таблица, именованный запрос, кот будет выполняться как подзапрос при использовании Представления. Это блок-подзапрос, кот нужен для того, чтобы по многу раз не создавать гигантские подзапросы, а просто обращаться к нему как к Представлению.

CREATE VIEW Xxxxx

AS

SELECT Maker FROM Auto WHERE Id IN (SELECT Auto-Id FROM Charachteristics WHERE Color=”Green”)

🡪

SELECT Maker FROM Xxxxx WHERE …

**? ПОЛЬЗОВАТЕЛЬСКАЯ ФУНКЦИЯ**

Пользовательская функция – программируемый объект базы данных, кот возвращает значение

Создать функцию для расчёта A × B

CREATE FUNCTION func(@a int, @b int)

RETURNS int

AS

BEGIN

RETURN (@a\*@b)

END;

GO

SELECT dbo.func(5, 8)

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

🡪 40

Создать функцию для расчёта площади круга по диаметру π × r2 = π × (D/2)2 = π × D2/22 = π × D2/4 = π/4 × D2

CREATE FUNCTION dbo.CircleS (@D dec(6,3)) / dbo.CircleS – создаётся ф-ция как БД, @... вводится переменная/

RETURNS dec(6,3) /параметр возврата данных: dec(6,3) – дробное число, всего 6 символов, 3 из 6 после «,»/

BEGIN /начало действия ф-ции/

RETURN (PI()/4)\*POWER(@D, 2) /сама ф-ция/

END /конец действия ф-ции/

GO /начало исполнения ф-ции/

PRINT dbo.CircleS(15) /действие – вывести функцию-БД, где значение переменной = 15/

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

🡪 176.715 /результат – тип данных дробное число, всего 6 символов, 3 из 6 после «,»/

CREATE FUNCTION funcA(@a int)

RETURNS varchar(30)

AS

BEGIN

DECLARE @res varchar(30)

IF (@a=0)

BEGIN

SET @res = 'This is zero, guys!'

END

ELSE IF (@a%2=0)

BEGIN

SET @res = 'Wow! Even!'

END

ELSE

BEGIN

SET @res = 'Wow! Odd!'

END

RETURN @res

END

GO

PRINT dbo.funcA(11)

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

🡪 Wow! Odd!

…

PRINT dbo.funcA(12)

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

🡪 Wow! Even!

…

PRINT dbo.funcA(0)

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

🡪 This is zero, guys!

**? ХРАНИМАЯ ПРОЦЕДУРА**

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. У них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML).

CALL процедура(…) – вызов хранимки, 1 способ

EXECUTE процедура(…) = EXEC процедура(…) – вызов хранимки, 2 способ

CREATE PROCEDURE = CREATE PROC

Создать хранимую процедуру, которая отобразит информацию о совершенных заказах, стоимость которых выше числа, заданного в аргументе.

DELIMITER //

CREATE PROCEDURE info\_orders (arg int)

BEGIN

SELECT onum, amt, odate

FROM orders

WHERE amt > arg;

END //

DELIMITER ;

CALL info\_orders(100);

**? Отличие Хранимой процедуры от Триггеров**

Отличаются от триггеров тем, что их надо запускать.

**? ТРИГГЕРЫ**

Триггер - это механизм, который вызывается, когда в указанной таблице происходит определенное действие.

Отличаются от хранимок тем, что их НЕ надо запускать.

3 типа SQL-триггеров:

* DML (INSERT, UPDATE, DELETE)
* DDL (CREATE, ALTER, DROP)
* Триггеры входа (LOG ON)

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

Пример-1:

CREATE TRIGGER UpdAfterDel

ON Auto FOR DELETE

AS

IF @@ROWCOUNT = 1 - - удаляем 1 запись

BEGIN

DECLARE @y int

- - создать переменную, в неё поместится код удалённой записи

SELECT @y = Id

FROM deleted

- - теперь откорректируем кол-во машин

UPDATE Showroom

SET Presence=’n’

WHERE Charact\_Id IN (SELECT Id FROM Characteristics WHERE Auto.Id=@y)

END

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

Пример-2:

Для рассмотрения операций с триггерами определим следующую базу данных productsdb:

Здесь определены две таблиц: Products - для хранения товаров и History - для хранения истории операций с товарами.

CREATE DATABASE productsdb;

GO

USE productsdb;

CREATE TABLE Products

(

Id INT IDENTITY PRIMARY KEY,

ProductName NVARCHAR(30) NOT NULL,

Manufacturer NVARCHAR(20) NOT NULL,

ProductCount INT DEFAULT 0,

Price MONEY NOT NULL

);

CREATE TABLE History

(

Id INT IDENTITY PRIMARY KEY,

ProductId INT NOT NULL,

Operation NVARCHAR(200) NOT NULL,

CreateAt DATETIME NOT NULL DEFAULT GETDATE(),

);

Добавление

При добавлении данных (при выполнении команды INSERT) в триггере мы можем получить добавленные данные из виртуальной таблицы INSERTED.

Определим триггер, который будет срабатывать после добавления:

USE productsdb

GO

CREATE TRIGGER Products\_INSERT

ON Products

AFTER INSERT

AS

INSERT INTO History (ProductId, Operation)

SELECT Id, 'Добавлен товар ' + ProductName + ' фирма ' + Manufacturer

FROM INSERTED

Этот триггер будет добавлять в таблицу History данные о добавлении товара, которые берутся из виртуальной таблицы INSERTED.

Выполним добавление данных в Products и получим данные из таблицы History:

USE productsdb;

INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)

VALUES('iPhone X', 'Apple', 2, 79900)

SELECT \* FROM History



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

? БД, API, HTML/CSS – тестирование серого ящика

? SQL – Авто-инкремент – автоматически прописывает Primary Key

? AVG (Average) сработает с GROUP BY, и не сработает с DISTINCT

? Нереляционные базы данных строятся на структуре JSON – ключ-значение

? В БД 4 поля, при отправке данных заполняются только 3 из 4. В чём может быть проблема?

* Разные типы данных: отправляемых и в 4ом поле (может быть даже разная на фронте и на беке, на беке и в БД и т.д.)
* Поле может быть необязательным и клиент может его просто не заполнить
* Валидация на фронте и на беке различная (н-р разное допустимое кол-во символов)
* Неправильно сформировалась джейсоночка
* В JSON может быть неправильно (с ошибкой) прописан Key для 4го поля
* В JSON может быть неправильно (с ошибкой) прописан Value для 4го поля, просто «””» с пустотой
* На уровне API – не улетел JSON, был не правильно прописан метод для этого поля, или просто не прописан URL
* Бэк эндеры уже прописали новый вариант прилаги с энд-поинтом в котором 4 поля, а фрон-эндеры ещё не прописали это и запрос шлётся на энд-поинт в 3 поля
* Посылается с фронта, но на беке не обрабатывается
* На бэк-энде не прописано что 4ое значение добавляется в 4ое поле
* Ошибка в SQL-запросе на бэк-энде
* ORM (Object relation method) – когда с БД отправляются не напрямую SQL-запросами, а на каком-либо языке программирования – могут быть ошибки
* На уровне БД – нет 4го поля
* На уровне БД – в 4ом поле NULL
* На уровне БД – поломана связь таблиц

**Client-Server model**

**? Клиент-серверная архитектура**

Клиент-серверная архитектура (client–server) — архитектура, в которой сетевая нагрузка распределены между поставщиками услуг — серверами, и заказчиками услуг — клиентами.

Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных компьютерах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

* **Сервер** – логический процесс, который обеспечивает некоторый сервис по запросу от клиента. Обычно сервер не только выполняет запрос, но и управляет очередностью запросов, буферами обмена, извещает своих клиентов о выполнении запроса и т. д.
* **Клиент** – процесс, который запрашивает обслуживание от сервера. Процесс не является клиентом по каким-то параметрам своей структуры, он является клиентом только по отношению к серверу. При взаимодействии клиента и сервера инициатором диалога с сервером, как правило, является клиент, сервер сам не инициирует совместную работу.
* **Сеть, протоколы** – третий компонент, который обеспечивает обмен информацией между клиентом и сервером.

Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, потоковое мультимедиа, работа с БД) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине).

Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

**Преимущества**

* Отсутствие дублирования кода программы-сервера программами-клиентами.
* Так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются.
* Все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

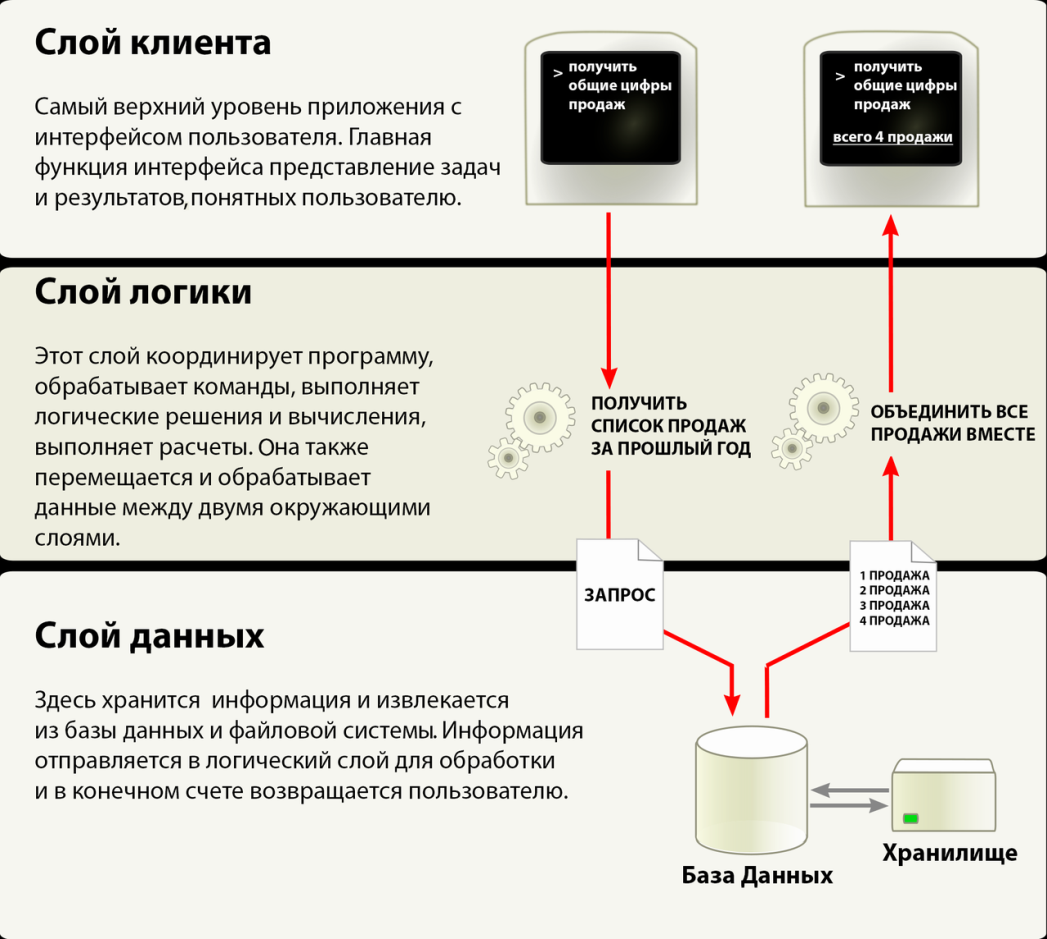
**Недостатки**

* Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п.
* Поддержка работы данной системы требует отдельного специалиста — системного администратора.
* Высокая стоимость оборудования.

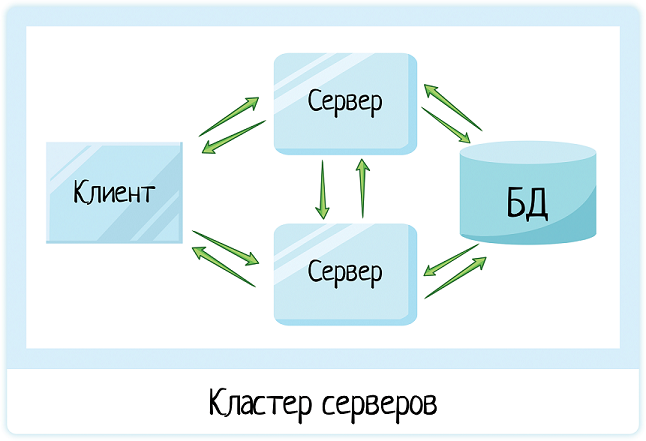
Многоуровневая архитектура «клиент — сервер» — разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Трехуровневая архитектура — архитектурная модель программного комплекса, 3 компонента:

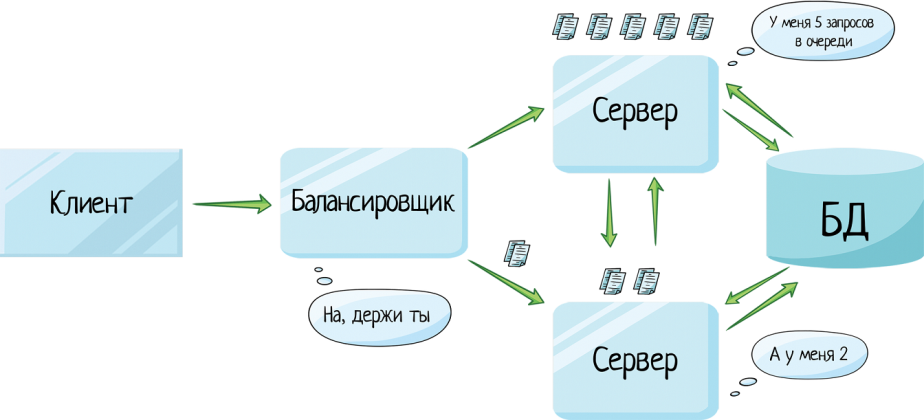
* клиент,
* сервер приложений (для которых подключено клиентское приложение)
* сервер баз данных (с этими работает сервер приложений).



В бизнес-критичном ПО архитектуру усложняют и даже дублирую



Перед серверами ставят балансировщик, и клиент шлет запрос туда. Сколько бы серверов не поставили в кластер, клиенту это не интересно. У него есть один URL — адрес балансировщика. Балансировщик спрашивает каждый из серверов – у кого сколько запросов в очереди. Сервера отвечают балансировщику. Балансировщик отправляет сам запрос на сервер с меньшим кол-вом запросов в очереди.

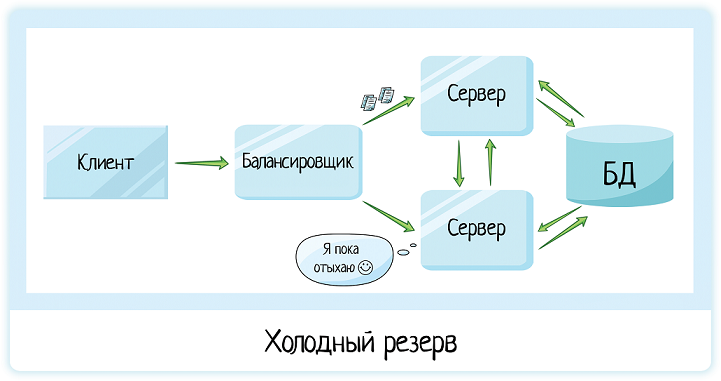


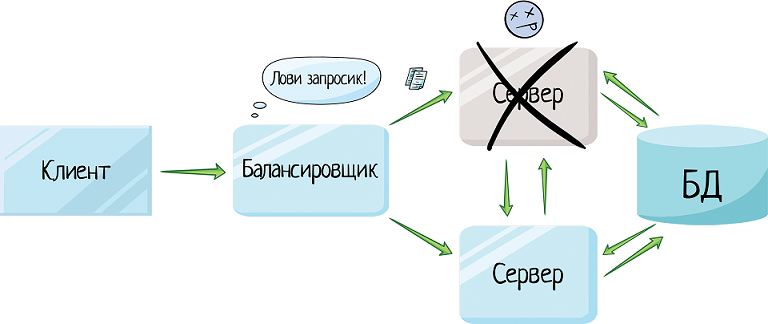
Также может быть несколько копий баз данных на самых разных машинах, и балансировщик отправляет запросы то к одной, то ко второй.

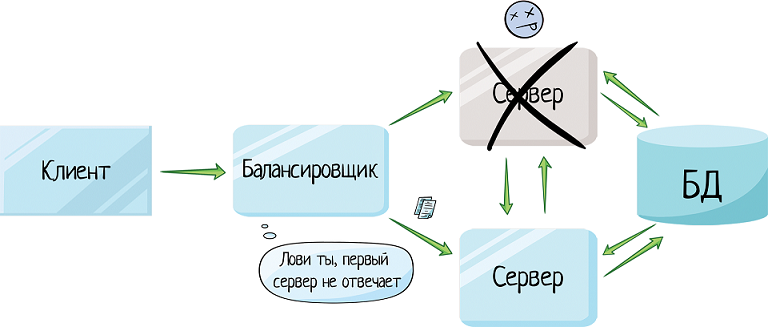


Такая схема называется горячий резерв — когда у нас есть несколько серверов, работающих в параллель, и балансировщик распределяет нагрузку между ними.

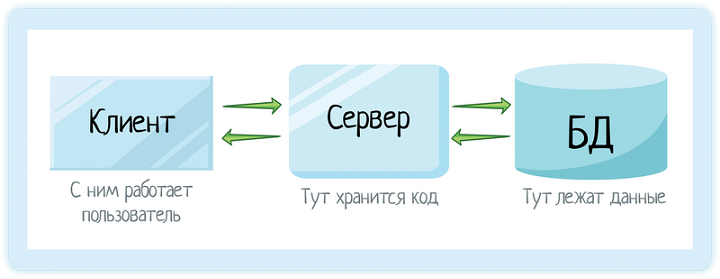
При этом может быть и схема холодного резерва — когда у нас второй сервер является резервной копией «на всякий случай». Все запросы идут на первый сервер, второй отдыхает. В это время у администраторов будет время разобраться с проблемой на сервере 1.







Итого

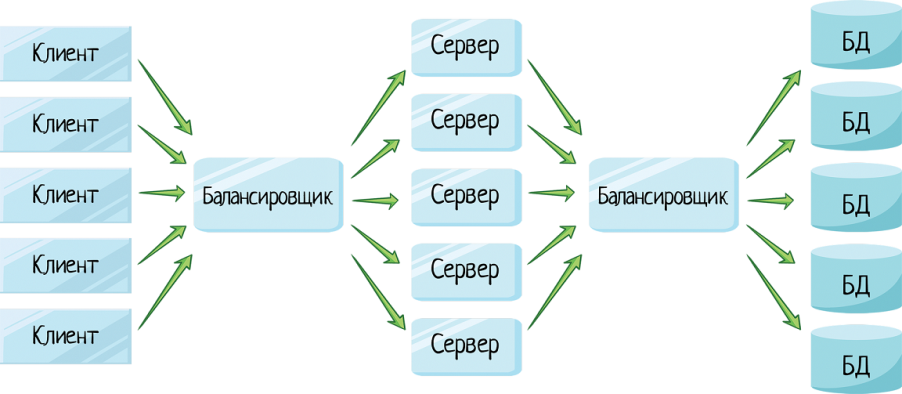


**Клиент** — та программа, с которой работает пользователь. Он знать не знает, это у него на компьютере программа целиком, или где-то за ней прячутся сервер с базой. Он работает в браузере или с desktop-приложением. И всё, что ему нужно знать — это «куда тут тыкать». Клиенту не нужно много памяти, места на диске и других ресурсов. Поэтому рабочие места относительно дешево стоят.

**Сервер** — компьютер, на котором хранится само приложение. Весь код, вся логика, все дополнительные материалы и справочники — они тоже занимают место, как сами по себе, так и в памяти приложения. Иногда говорят «**сервер приложения**» и «**сервер БД**». Это нормально, ведь фактически сервер — это просто машина, компьютер. А базу и сервер приложения обычно хранят на разных машинах, ради безопасности. В таком случае, если говорят «сервер приложения» — речь о втором звене нашей схемы.

**БД (база данных)** — хранилище данных. Тут можно легко поискать информацию + уверены в том, что она сохранится, даже если в приложении что-то сломается. Отдельной базы может не быть, тогда структура станет двухзвенной: клиент-сервер. И все!

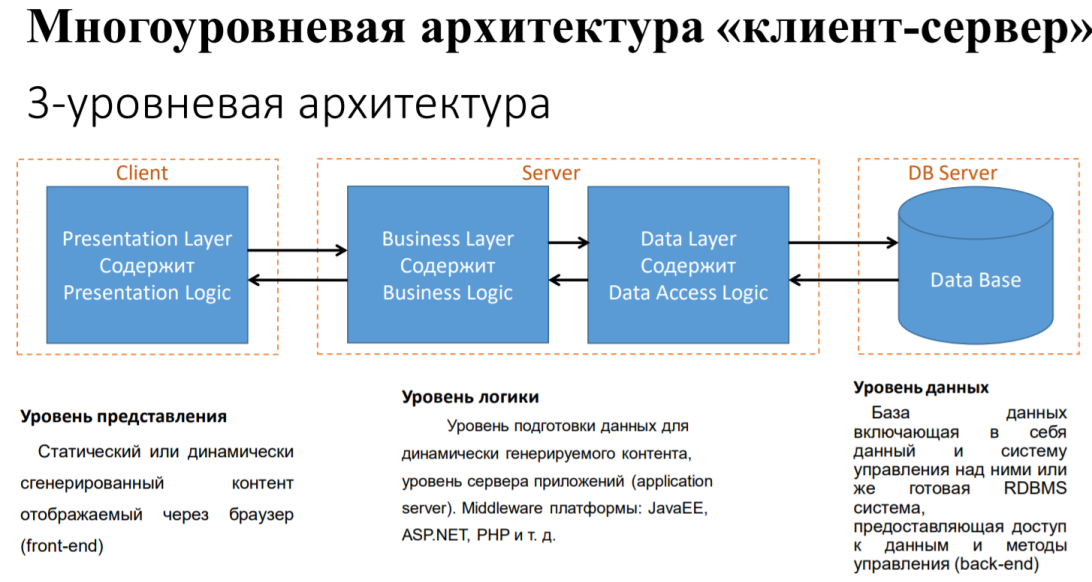
Схема условная, в реальной жизни как минимум будет больше клиентов. А если приложение высоконагруженное, то будет несколько серверов и несколько баз данных:



**? Толстый/Тонкий клиент:**

* «**Толстый клиент**» – на сервере реализованы главным образом функции доступа к базам данных, а основные прикладные вычисления выполняются на стороне клиента.
* «**Тонкий клиент**» – на сервере выполняется основная часть прикладной обработки данных, а на клиентские рабочие станции передаются уже результаты обработки данных для просмотра и анализа пользователем с возможностью их последующей обработки (в минимальном объеме).

**? Многоуровневая архитектура**



**? Отличие сервиса от сервера**

Сервис относится к программным функциям (таким как получение указанной информации или выполнение набора операций) или «механизм, обеспечивающий доступ к одной или нескольким возможностям, где доступ предоставляется с использованием предписанного».

Сервер – это часть оборудования или компьютерная программа, которая обеспечивает функциональные возможности для других программ или устройств, называемых «клиентами». Серверы могут предоставлять различные функции, часто называемые «сервисами», такие как совместное использование данных или ресурсов между несколькими клиентами или выполнение вычислений для клиента.

Веб-сервис – тот же дополнительный сервер, выполняющий некую определённую функцию, кот запущен на своей железяке, у него есть свой порт, свой API, свой HTTP

Н-р: Мы запрашиваем самый выгодный курс валют в Минске.

Запрос Клиента Серверу

Сервер обращается к Сервису

Сервис обращается через API на другие серверы: нацбанка, бирж, банков

Сервис получает ответы от серверов нацбанка, бирж, банков

Вычисление выгодного курса (сравн. результатов) может производиться и Клиентом и Сервером и Сервисом

**? Отличие Веб-Сервиса от Веб-Сайта**

* Веб-сервис не имеет пользовательского интерфейса.   
  Веб-сайт имеет пользовательский или графический интерфейс.
* Веб-сервисы предназначены для взаимодействия других приложений через Интернет.   
  Веб-сайты предназначены для использования людьми.
* Веб-сервисы не зависят от платформы, так как используют открытые протоколы.   
  Веб-сайты являются кроссплатформенными, так как требуют настройки для работы в разных браузерах, операционных системах и т. д.
* Доступ к веб-сервисам осуществляется с помощью HTTP-методов - GET, POST, PUT, DELETE и т. д.   
  Доступ к веб-сайтам осуществляется с помощью компонентов GUI - кнопок, текстовых полей, форм и т. д.
* Например, Google maps API - это веб-сервис, который может использоваться веб-сайтами для отображения Карт путем передачи ему координат.   
  Например, ArtOfTesting.com - это веб-сайт, на котором есть коллекция связанных веб-страниц, содержащих учебные пособия.

**API**

**? API (Application Programming Interface) Программный Интерфейс Приложения** — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. Обычно входит в описание какого-либо интернет-протокола (например, RFC), программного каркаса (фреймворка) или стандарта вызовов функций операционной системы. Часто реализуется отдельной программной библиотекой или сервисом операционной системы.

**? API** – интерфейс, способ общения программ с сервером

**? API** – набор энд-поинтов

**? Сравнение IRL** – ресторан

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Клиент-Сервер: | Клиент: программа,  браузер, | 🡨🡪 | API | 🡨🡪 | Сервер | 🡨🡪 | База Данных |
| Ресторан | Клиент | 🡨🡪 | Официант | 🡨🡪 | Кухня | 🡨🡪 | Склад |

**? Тонкий/Толстый Клиент (Thin/Thick Client)**

Тонкий Клиент – основные прикладные вычисления выполняются на стороне сервера

Толстый Клиент – основные прикладные вычисления выполняются на стороне клиента (пример в ресторане – клиент с сумкой со своей едой)

**? Примеры API в жизни:**

* Клиент в ресторане
* Переводчик между туристом и местными
* Розетка для приборов

**? Можно ли с помощью АПИ достучаться до сервера без браузера** – да, примеры:

* Сканер QR кодов в кинотеатре (доступ к серверу с инфо о действительности билета)
* Сканер штрих-кода на кассе (доступ к серверу с БД товаров)

**? Список статус кодов HTTP ответа сервера**

Коды ответа ( для тестировщика) – это просто удобное понимание, как именно отреагировал сервер на web или API запрос.

Разделяются на 5 групп:

* 1xx – Info – Информационные (100-105)
* 2xx – Success – Успешные (200-226)
* 3xx – Redirect – Перенаправление (300-307)
* 4xx – Client error – Ошибка клиента (400-499)
* 5xx – Server error – Ошибка сервера (500-510)

*Подробнее – секция «Сети»*

**? Почему ошибка 404 относится к 4\*\* - клиентской, если по идее должна быть 5\*\*?**

Хотя интуитивно можно подумать, что данная ошибка должна относиться к ошибкам со стороны сервера, 404 по задумке является клиентской ошибкой, то есть подразумевается, что клиент (Вы) должен был знать, что URL страницы был перемещен или удален и Вы пытаетесь открыть несуществующую страницу.

**? Curl (Client URL) – отправить запросы на сервер из Командной строки**

Curl (Client URL) – это утилита командной строки, которая позволяет выполнять HTTP-запросы с различными параметрами и методами. Вместо того, чтобы переходить к веб-ресурсам в адресной строке браузера, можно использовать командную строку, чтобы получить те же ресурсы, извлеченные в виде текста.

Утилита доступна в большинстве систем на основе Unix и предназначена для проверки подключения к URL-адресам. Кроме того команда Curl — отличный инструмент передачи данных. Curl работает на libcurl, которая является бесплатной библиотекой для передачи URL на стороне клиента.

**? Как картинка улетает на сервер**

Фото 🡪 преобразуется в строку Base64 🡪 строка POSTом улетает на сервер 🡪 дальше в БД 🡪 и хранится там в виде строки.

**? Токен (также** [**программный токен**](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D1%82%D0%BE%D0%BA%D0%B5%D0%BD&action=edit&redlink=1)**)** – ключ для доступа к службам, кот выдаётся пользователю после успешной [авторизации](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F). Токены предназначены для электронного [удостоверения личности](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%B5%D1%80%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BB%D0%B8%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8) (н-р, клиента, получающего доступ к банковскому счёту), при этом они могут использоваться как вместо [пароля](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D0%BB%D1%8C), так и вместе с ним. В некотором смысле токен — это электронный ключ для доступа к чему-либо.

**? Токен –** комбинация букв, цифр, символов, генерирующийся сервером

**? Откуда система берёт информацию об у-ве клиента**

Кроме токена – из юзер-агента

**? Идентификация / Аутентификация / Авторизация**

* Идентификация — процесс распознавания пользователя по его идентификатору.
* Аутентификация — процедура проверки подлинности, доказательство, что пользователь именно тот, за кого себя выдаёт.
* Авторизация — предоставление определённых прав

? **AJAX, Ajax (Asynchronous JavaScript and XML)** — «асинхронный [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) и [XML](https://ru.wikipedia.org/wiki/XML)» — подход к построению интерактивных [пользовательских интерфейсов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), заключающийся в «фоновом» [обмене данными](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D0%B5%D0%BD_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8) [браузера](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) с [веб-сервером](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80). В результате при обновлении данных [веб-страница](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:

* использование технологии динамического обращения к [серверу](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) «на лету», без перезагрузки всей страницы полностью, например, с использованием [XMLHttpRequest](https://ru.wikipedia.org/wiki/XMLHttpRequest) (основной объект);
* использование [DHTML](https://ru.wikipedia.org/wiki/DHTML) для динамического изменения содержания страницы;

В классической модели веб-приложения:

1. Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
2. [Браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) формирует и отправляет [запрос](https://ru.wikipedia.org/wiki/HTTP) [серверу](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80);
3. В ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д., после чего браузер полностью перезагружает всю страницу.

При использовании AJAX:

1. Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
2. [Скрипт](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82) (на языке [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)) определяет, какая информация необходима для обновления страницы;
3. [Браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) отправляет соответствующий запрос на [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5));
4. Сервер возвращает только ту часть документа, на которую пришёл запрос;
5. Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

Преимущества:

* Экономия траффика
* Уменьшение нагрузки на сервер
* Ускорение реакции интерфейса
* Возможности для интерактивной разработки
* Мультимедиа не останавливается

**? Снифферы** – Charles/Fiddler – программы для настройки прокси и отслеживания траффика

**? Есть ли у веб-сервиса API** – У веб-сервиса есть API

**REST vs SOAP**

**? Что такое RSET и SOAP**

**REST (Representational State Transfer)** — архитектурный стиль взаимодействия компьютерных систем в сети основанный на методах протокола HTTP. Данные по умолчанию передаются в JSON (также могут и в XML).

**SOAP (Simple Object Access Protocol)** — стандартный протокол обмена структурированными сообщениями в распределенной вычислительной среде. Данные передаются **только** в XML.

**? Отличие RSET от SOAP**

REST и SOAP на самом деле не сопоставимы.

* REST — это архитектурный стиль.
* SOAP — это формат обмена сообщениями.

**? Популярные реализации стилей REST и SOAP.**

* Пример реализации **RESTful**: JSON через HTTP
* Пример реализации **SOAP**: XML поверх SOAP через HTTP

На верхнем уровне SOAP ограничивает структуры ваших сообщений, тогда как REST — это архитектурный подход, ориентированный на использование HTTP в качестве транспортного протокола.

**? Специфика REST и SOAP**

* Специфика SOAP — это формат обмена данными. С SOAP это всегда SOAP-XML, который представляет собой XML, включающий:
  + Envelope (конверт) – корневой элемент, который определяет сообщение и пространство имен, использованное в документе,
  + Header (заголовок) – содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации,
  + Body (тело) – содержит сообщение, которым обмениваются приложения,
  + Fault – необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений. И запрос, и ответ должны соответствовать структуре SOAP.
* Специфика REST — использование HTTP в качестве транспортного протокола. Он подразумевает наилучшее использование функций, предоставляемых HTTP — методы запросов, заголовки запросов, ответы, заголовки ответов и т. д.

#### ? Формат обмена сообщениями

* В SOAP вы используете формат SOAP XML для запросов и ответов.
* В REST такого фиксированного формата нет. Вы можете обмениваться сообщениями на основе XML, JSON или любого другого удобного формата. JSON является самым популярным среди используемых форматов.

#### ? Определения услуг

* SOAP использует **WSDL** (Web Services Description Language) — язык описания веб-сервисов и доступа к ним, основанный на языке XML, структура, правило того, как должен работать SOAP-сервис.
* REST не имеет стандартного языка определения сервиса. WADL (Web Application Description Language) был одним из первых предложенных стандартов, но он не очень популярен. Более популярно использование Swagger или Open API. Так же JSON-схема – редко используется – структура JSON, которая должна прилететь с сервера и её можно провалидировать.

#### ? Транспорт

* SOAP не накладывает ограничений на тип транспортного протокола. Можно использовать либо Web протокол HTTP, либо MQ.
* REST подразумевает наилучшее использование транспортного протокола HTTP

#### ? Простота реализации REST и SOAP

* REST веб-сервисы, как правило, гораздо проще реализовать, чем веб-сервисы на основе SOAP.  
  REST обычно использует JSON, который легче анализировать и обрабатывать.  
  REST не требует наличия определения службы для предоставления веб-службы.
* Однако в случае SOAP необходимо определить свой сервис с использованием WSDL.  
  При обработке и анализе сообщений SOAP-XML возникают большие накладные расходы.

**? Что такое JSON и XML**

JSON (Java Script Object Notation) – это текстовый формат, написанный на JavaScript.

XML (eXtension Markup Language) – это язык разметки, а не язык программирования, в котором существуют теги для определения элементов.

**? Принципиальная разница между JSON и XML**

* XML – это язык markup (как он на самом деле говорит в своем названии), протокол, транспортировка и хранение данных
* JSON – это способ представления объектов (как также отмечено в его названии)

**? Ещё определение JSON**

JSON – Неупорядоченное множество «ключ – значение»

**? Как передаётся по сети JSON**

JSON текстовый формат данных – передаётся по сети как текст, всё в одну строку в общих кавычках – сначала его надо парсить (перевести JSON формат в одну строку), потом распарсить (перевести одну строку обратно в JSON формат).

**? Примеры JSON и XML**

**XML**

<?xml version="1.0" encoding="UTF-8"?>

<root>

<cотрудники>

<id>01</id>

<имя>Влад</имя>

<команда>Разработчик</команда>

<технология>Веб</технология>

<должность>Инженер</должность>

</cотрудники>

</root>

**JSON**

{

"cотрудники": [{

"id": "01",

"имя": "Влад",

"команда": "Разработчик",

"технология": "Веб",

"должность": "Инженер",

}]

}

**? Есть ли тело у GET** – у GET нету body.

**? Различие GET и HEAD**

**GET** – служит для получение информации с ресурса. В ответе на GET есть тело.

**HEAD** – служит для проверки существования ресурса, он полностью аналогичен **GET**, но без возврата тела ответа.

**? Модель CRUD** – create read update delete

|  |  |  |  |
| --- | --- | --- | --- |
| CRUDE-model | SQL (Base, Table Level) | SQL (Data Level) | API (REST style) |
| CREATE | CREATE | INSERT | POST |
| READ | SELECT | SELECT | GET |
| UPDATE | ALTER | UPDATE | PUT, PATCH |
| DELETE | DROP | DELETE | DELETE |

**? План работы по тестированию Веб-Сервиса**

* Изучить требования
* Написать тест-кейсы
* Протестировать

**? Требования, входные данные:**

* Какие параметры/атрибуты есть у веб-сервиса
* Образцы запросов (у разработчиков)

**? Какие Тест-кейсы**

* Smoke (самые основные)
* Critical-Pass (валидные)
* Extended (невалидные)

**? Специфические тесты**

* Пустые эл-ты
* Комментарии
* Валидность ответов , согласно схеме (при наличии, по WSDL)
* Проверка с обязательными и необязательными атрибутами
* Проверки с дополнительными и недопустимыми значениями
* Различные типы данных
* Дубликаты атрибутов эл-тов
* Порядок атрибутов/эл-тов
* Длина строк данных
* Невалидные данные (синтаксис XML и JSON)

**? Что смотреть в ответе**

* Статус-код
* Тело ответа
* Текст ошибки

**Postman**

**? Что такое Postman**

Постман представляет собой мультитул для тестирования API (создан для автоматизации тестирования API)

**? Что можно делать в Postman**

В нем можно создавать коллекции запросов, проектировать дизайн API и создавать для него моки (заглушки-имитации ответов реального сервера), настраивать мониторинг (периодическая отправка запросов с журналированием), для запросов возможно написание тестов на JS, есть собственный Runner и т.д. Однако постман сложно назвать подходящим инструментом для серьезной автоматизации ввиду сложности поддержки тестов, но при этом он хорошо подойдет в простых случаях или как инструмент поддержки а анализа: проверка работоспособности endpoint, дебаг тестов, простая передача информации о дефектах (можно сохранить запрос в curl, ответ в json и т.п.). Postman также может работать без графического интерфейса (newman).

**? Базовые возможностями инструмента Postman:**

* Составить и отправить запросы;
* Задать параметры в запросах;
* Создать и переключиться между окружениями;
* Написать базовые тесты;
* Создать и отредактировать коллекции;
* Выполнить запуск тестов.

**? Навигация в Постмане:**

**The left sidebar** provides:

* Collections – access to your collections,
* API
* Environments
* Mock Servers
* Monitors
* History

**The header** allows you to create workspaces, access reports, explore the public API network, search within Postman, view sync status and notifications, move and invite collaborators to workspaces, capture requests and cookies, access your settings, account, and Postman plan.

**The center area** is where you build and work with requests.

**The status bar** along the bottom allows you to show/hide the sidebar, find and replace, and open the console on the left. On the right you can launch the Bootcamp, collection runner, trash, two pane view, and access help resources.

**? Что такое Workspace**

Workspace позволяют вам организовывать проекты API и совместно работать над ними с командой. В каждой рабочей области вы можете делиться API, коллекциями, средами и другими компонентами работы.

**? Создание Рабочего пространства / Коллекции / Запроса:**

Создать Рабочее пространство: Workspace 🡪 «+ New Workspace» ИЛИ «New» 🡪 Workspace

Создать Коллекцию: «+» или «New» 🡪 Collection

В Коллекции можно добавить папки: «◦◦◦» 🡪 Add Folder

Создать Запрос: «◦◦◦» 🡪 Add Request

? Синтаксис запроса

«URL» «/ tasks/rest/» «action» «?» «parameter=value» «&» «parameter=value» «&» «parameter=value»

**? Запросы:**

**РЕГИСТРАЦИЯ (doregister)**

Create Collection, Add Request

Выбрать метод: POST

Копировать и вставить URL сайта: http://users.bugred.ru/

На сайте посмотреть поля для регистрации (корректные имена через DevTool): name, email, password

Регистрация через строку – дописать «/tasks/rest/**doregister**»

Тег «Params» – прописать «Key» «Value»

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| name | Myname |  |
| email | mymail@gmail.com |  |
| password | 12345 |  |

Строка допишется автоматически парами «Ключ – Значение»

Или

Прописать пары «Ключ – Значение» в строку: начиная с «?», и разделяя пары «&» (таблица заполнится автоматически)

**POST** | http://users.bugred.ru/tasks/rest/doregister?name=Myname&email=mymail@gmail.com&password=12345

Нажать «Send»

Регистрация на сайте произойдёт (также как и через браузер).

**АВТОРИЗАЦИЯ (dologin)**

Add Request

Выбрать метод: GET

Копировать и вставить URL сайта: http://users.bugred.ru/

На сайте (в спеке) посмотреть, что для регистрации нужен только и-мейл и пароль

Тег «Params» – прописать «Key» «Value»

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| email | mymail@gmail.com |  |
| password | 12345 |  |

Строка допишется автоматически парами «Ключ – Значение»

Или

Прописать пары «Ключ – Значение» в строку: начиная с «?», и разделяя пары «&» (таблица заполнится автоматически)

**GET** | http://users.bugred.ru/tasks/rest/dologin?email=mymail@gmail.com&password=12345

Нажать «Send»

Авторизацияна сайте произойдёт (также как и через браузер).

**ПОИСК ПО ПОЧТЕ (getuser)**

Add Request

Выбрать метод: GET

Копировать и вставить URL сайта: http://users.bugred.ru/

Тег «Params» – прописать «Key» «Value»

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| email | mymail@gmail.com |  |

Строка допишется автоматически парой «Ключ – Значение»

Или

Прописать пары «Ключ – Значение» в строку: начиная с «?» (таблица заполнится автоматически)

**GET** | http://users.bugred.ru/tasks/rest/getuser?email=mymail@gmail.com

Нажать «Send»

Поиск и отображениена сайте произойдёт (также как и через браузер).

**УДАЛЕНИЕ ПОЛЬЗОВАТЕЛЯ (deleteuser)**

Add Request

Выбрать метод: DELETE

Копировать и вставить URL сайта: http://users.bugred.ru/

Тег «Params» – прописать «Key» «Value»

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| email | mymail@gmail.com |  |

Строка допишется автоматически парой «Ключ – Значение»

Или

Прописать пары «Ключ – Значение» в строку: начиная с «?» (таблица заполнится автоматически)

**DELETE** | http://users.bugred.ru/tasks/rest/deleteuser?email=mymail@gmail.com

Нажать «Send»

Удалениена сайте произойдёт (также как и через браузер).

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

**ЗАПРОСЫ**

Тренировочный сайт: **reqres.in**

БД «виртуальных» пользователей: id, email, first\_name, ….

На сайте выведены правильные коды состояния + содержание тела ответа (своего рода «спецификация»)

**ПОЛУЧЕНИЕ СПИСКА ПОЛЬЗОВАТЕЛЕЙ**

Выбираем на сайте секцию: GET – LIST USERS

Дан правильный результат:

Статус-код: 200,

Тело ответа:

{

"page": 2,

"per\_page": 6,

"total": 12,

"total\_pa……

На сайте «https://reqres.in» копируем энд-поинт «/api/users?page=2»: https://reqres.in/api/users?page=2

В Постмане создаём Запрос, вставляем URL + end-point: **GET** | https://reqres.in/api/users?page=2

«Send»

Запрос проходит успешно – результат совпадает с образцом на сайте:

Статус-код: 200,

Тело ответа:

{

"page": 2,

"per\_page": 6,

"total": 12,

"total\_pa……

**ПОЛУЧЕНИЕ КОНКРЕТНОГО ПОЛЬЗОВАТЕЛЯ**

Выбираем на сайте секцию: GET – SINGLE USER

Дан правильный результат

На сайте «https://reqres.in» копируем энд-поинт «/api/users/2»: https://reqres.in/api/users/2

В Постмане создаём Запрос, вставляем URL + end-point: **GET** | https://reqres.in/api/users/2

«Send»

Запрос проходит успешно – результат совпадает с образцом на сайте:

Статус-код: 200,

Тело ответа:

{

"data": {

"id": 2,

"email": "janet.weaver@reqres.in",

"first\_name": "Janet",

….

**ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯ**

Выбираем на сайте секцию: POST – CREATE

На сайте «https://reqres.in» копируем энд-поинт «/api/users»: https://reqres.in/api/users

В Постмане создаём Запрос, вставляем URL + end-point: **POST** | https://reqres.in/api/users

В тело запроса прописываем юзера, кот хотим создать (на сайте есть пример, можно скопировать):

Тег «Params» 🡪 «Body» 🡪 «(•) raw» + «text ˅ 🡪 JSON ˅»

В тело вставляем:

{

"name": "morpheus",

"job": "leader"

}

«Send»

Запрос проходит успешно – результат совпадает с образцом на сайте:

Статус-код: 201,

Тело ответа:

{

"name": "morpheus",

"job": "leader",

"id": "218",

"createdAt": "2021-12-18T13:26:31.007Z"

}

**АВТОРИЗАЦИЯ ПОЛЬЗОВАТЕЛЯ (ЛОГИН)**

Выбираем на сайте секцию: POST – CREATE

На сайте «https://reqres.in» копируем энд-поинт «/api/register»: https://reqres.in/api/register

В Постмане создаём Запрос, вставляем URL + end-point: **POST** | https://reqres.in/api/register

В тело запроса прописываем юзера, кот хотим создать (на сайте есть пример, можно скопировать):

Тег «Params» 🡪 «Body» 🡪 «(•) raw» + «text ˅ 🡪 JSON ˅»

В тело вставляем:

{

"email": "eve.holt@reqres.in",

"password": "pistol"

}

«Send»

Запрос проходит успешно – результат совпадает с образцом на сайте:

Статус-код: 200,

Тело ответа:

{

"id": 4,

"token": "QpwL5tke4Pnpja7X4"

}

**ПЕРЕМЕННЫЕ**

В Постмане есть 3 типа переменных:

* « **C** » Переменные Коллекции (Collection Variables) – работают на ур-не Коллекции
* « **E** » Окружения (Environment Variables) – работают для всех Коллекций в Окружении
* « **G** » Глобальные (Global Variables) – работают для всех Окружений

**ОКРУЖЕНИЕ**

Чтобы каждый раз не менять значения параметров (Key) внутри каждого Запроса в Коллекции, можно создать окружение с набором переменных – значений (Values), которые будут подставляться в каждый Запрос.

Создаём Окружение: New 🡪 Environment: «Test Env»

|  |  |  |
| --- | --- | --- |
| VARIABLE | INITIAL VALUE  (шарится на сервер для всей команды) | CURRENT VALUE  (не шарится, видно только нам) |
| name | Newname | Newname (копируется автоматически) |
| email | Newmail@gmail.com | Newmail@gmail.com (копируется авто) |
| password | 56789 | 56789 (копируется автоматически) |

*\* В целях безопасности колонку INITIAL VALUE можно не заполнять значениями, а заполнить только CURRENT VALUE*

Сохранить Окружения SAVE

Вернуться к Коллекции

В КАЖДОМ запросе в Параметрах «Params» прописываем названия переменных в «{{ … }}»

Либо вручную прописать в строке запроса – таблица «Params» и строка связанны автоматически

В Запросах пар-ры «Key» будут обращаться к значению «VALUE», получать направление на Окружение к колонке «VARIABLE» и получать значения «VALUE» оттуда

**POST** | http://users.bugred.ru/tasks/rest/doregister?name={{name}}&email={{email}}&password={{password}}

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| name | {{name}} |  |
| email | {{email}} |  |
| password | {{password}} |  |

**GET** | http://users.bugred.ru/tasks/rest/dologin?email={{email}}&password={{password}}

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| email | {{email}} |  |
| password | {{password}} |  |

**GET** | http://users.bugred.ru/tasks/rest/getuser?email={{email}}

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| email | {{email}} |  |

**DELETE** | http://users.bugred.ru/tasks/rest/deleteuser?email={{email}}

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| email | {{email}} |  |

**СОЗДАНИЕ ПЕРЕМЕННОЙ ДЛЯ URL**

Чтобы по многу раз не копировать URL (или что либо ещё) можно создать для него переменную

I СПОСОБ

Перейти в заголовок самой коллекции «˅ Collection»

Тег «Authorization» 🡪 «Variables»

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| url | https://reqres.in | https://reqres.in (копируется авто) |

II СПОСОБ

В строке запроса выделить необходимую часть: https://reqres.in/api/register

Во всплывающем теге кликнуть «Set as variable»

Атрибуты в таблице «Variables» коллекции «Collection» заполнятся автоматически

**ИСПОЛЬЗОВАНИЕ ЦЕЛОГО НАБОРА ПЕРЕМЕННЫХ ИЗ ВНЕШНИХ ФАЙЛОВ**

Выбираем на сайте секцию: POST – CREATE

На сайте «https://reqres.in» копируем энд-поинт «/api/register»: https://reqres.in/api/register

В Постмане создаём Запрос, вставляем URL + end-point: **POST** | https://reqres.in/api/register

В тело запроса прописываем юзера, кот хотим создать (на сайте есть пример, можно скопировать):

В тело вставляем:

{

"email": "eve.holt@reqres.in",

"password": "pistol"

}

Создаём Окружение: New 🡪 Environment: «Test Env» и прописываем Переменные

|  |  |  |
| --- | --- | --- |
| VARIABLE | INITIAL VALUE  (шарится на сервер для всей команды) | CURRENT VALUE  (не шарится, видно только нам) |
| email | eve.holt@reqres.in | (копируется автоматически) |
| password | pistol | (копируется автоматически) |

*\* В целях безопасности колонку INITIAL VALUE можно не заполнять значениями, а заполнить только CURRENT VALUE*

В теле изменяем:

{

"email": “{{email}}”,

"password": “{{password}}”

}

Запрос проходит успешно – результат совпадает с образцом на сайте:

Статус-код: 200,

Тело ответа:

{

"id": 4,

"token": "QpwL5tke4Pnpja7X4"

}

**СПИСОК ПЕРЕМЕННЫХ В CSV ФАЙЛЕ**

Создать внешний CSV файл «File.csv» с переменными: File.txt 🡪 File.csv

Открывается в Excel

Создать список переменных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| email | password |  |  |  |
| 11@test.com | 11111 |  |  |  |
| 22@test.com | 22222 |  |  |  |
| eve.holt@reqres.in | pistol |  |  |  |
|  |  |  |  |  |

В текстовом редакторе изменить файл, разбив значения запятыми и убрать пробелы (чтобы Постман правильно его считал):

|  |
| --- |
| email,password  11@test.com,11111  22@test.com,22222  eve.holt@reqres.in,pistol |

В Постмане запускаем (ранним) Коллекцию

В преднастройках выбираем «No Environment», «Data File Type: text/csv», «Data: Select File: File.csv»

Проверяем список переменных «Preview»:

|  |  |  |
| --- | --- | --- |
| Iteration | email | password |
| 1 | “11@test.com” | 11111 |
| 2 | “22@test.com” | 22222 |
| 3 | “eve.holt@reqres.in” | pistol |
|  |  |  |

«Run»

Выполниться 3 итерации, из которых 2 первые не пройдут (код 400), т.к. данные не совпадают с данными сайта, а последний – успешно пройдёт (код 200)

**СПИСОК ПЕРЕМЕННЫХ В JSON ФАЙЛЕ**

Создать внешний JSON файл «File.json» с переменными: File.txt 🡪 File.json

Открывается в Текстовом редакторе

[

{

"email":"111@gmail.com",

"password":"11111"

},

{

"email":"222@gmail.com",

"password":"22222"

},

{

"email":"eve.holt@reqres.in",

"password":"pistol"

}

]

В Постмане запускаем (ранним) Коллекцию

В преднастройках выбираем «No Environment», «Data File Type: application/json», «Data: Select File: File.json»

Проверяем список переменных «Preview»:

|  |  |  |
| --- | --- | --- |
| Iteration | email | password |
| 1 | “11@test.com” | 11111 |
| 2 | “22@test.com” | 22222 |
| 3 | “eve.holt@reqres.in” | pistol |
|  |  |  |

«Run»

Выполниться 3 итерации, из которых 2 первые не пройдут (код 400), т.к. данные не совпадают с данными сайта, а последний – успешно пройдёт (код 200)

**ТЕСТИРОВАНИЕ В POSTMAN**

Перейти к Коллекции, запустить Раннер: «◦◦◦» 🡪 «Run Collection»,

Указать какие из Запросов из Коллекции проводить «🗹»

Указать кол-во итераций (сколько подходов)

«Run»

Предварительно надо создать тесты и прописать их во вкладке «Tests», т.е. чтобы ответ Запроса содержал Фактический результат, который можно будет сравнить с Ожидаемым результатом прописанным в сниппете.

Справа появится список «Snippets» - готовых шаблонов теста

* проверить статус-код ответа
* проверить наличие «string» строки в ответе или конкретного значения внутри JSON
* проверить время ответа
* проверить наличие определенного заголовка в ответе
* конвертировать XML ответ в JSON объект

Сниппеты – это готовые кусочки JS, их можно изменять, подставляя свои параметры, названия.

Примеры:

* Статус код 200
* Тело Ответа содержит строку «….»
* Номер веб-страницы равен «…»
* Время Ответа < 150ms

**pm.test("Status code is 200", *function* () {**

**pm.response.to.have.status(200);**

**});**

**pm.test("Body matches string", *function* () {**

**pm.expect(pm.response.text()).to.include("https://reqres.in/img/faces/7-image.jpg");**

**});**

**pm.test("Verify page # is 2", *function* () {**

***var* jsonData = pm.response.json();**

**pm.expect(jsonData.page).to.eql(2);**

**});**

**pm.test("Response time is less than 150ms", *function* () {**

**pm.expect(pm.response.responseTime).to.be.below(150);**

**});**

В каждом из Запросов в Ответе (внизу) перейти с тега «Body» на тег «Test Results»

Там можно посмотреть Результаты по каждому по тестам «All», «Passed», «Skipped», «Failed»

**« PASS » Test Name** – Пройден

**« FAIL » Test Name** – Не пройден

**« SKIPPED » Test Name** – Пропущен

Тег «Pre-request script» - можно ввести какие-либо предварительные скрипты, которые будут выполняться перед запуском теста.

Например:

Создать запрос GET

Тег «Pre-request script»

Пишем скрипт на создание глобальной переменной:

pm.globals.set("filter", "user-1");

Тег «Parms» вводим «Key» и «Value»

|  |  |  |
| --- | --- | --- |
| KEY | VALUE | DESCRIPTION |
| query | {{filter}} |  |

**GET** | https://postman-echo.com/get?query={{filter}}

В Ответе видим что для значения приходит значение из Предзапроса «user-1»

**ДОКУМЕНТАЦИЯ**

Выбрать Коллекцию

Справа на вкладке «Documentation» нажать «Edit»

Добавить описание Коллекции

Также, справа, можно добавить описание (Документацию) для каждого из Запросов в коллекции

Внизу «View complete collection documentation 🡪» просмотреть описание Коллекции и Запросов

**СОЗДАНИЕ СИМУЛЯТОРА ОТВЕТА СЕРВЕРА (MOCK SERVER)**

«New» 🡪 «Mock Server»

Выбрать между «Create a new collection» и «Select existing collection»

Создать симулированный Ответ со статус кодом и телом:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Request Method | Request URL | Response Code | Response Body | ◦◦◦ |
| GET | status | 304 | Info older than my grandpa:) |  |
|  |  |  |  |  |

Ввести имя Мок-Сервера: Mock 304

Выбрать «No Environment»

🗹 Save the mock server URL as an environment variable

«Create Mock Server»

Создастся 3 × «Mock 304» Мок-Сервер + Окружение + Коллекция с запросом GET

Выбрать Окружение «Mock 304»

В Коллекции «Mock 304» отправить запрос – получим ответ «304 Not Modified / Info older than my grandpa:)»

**МОНИТОРИНГ**

В Коллекции «◦◦◦» 🡪 Monitor Collection

Ввести имя Name: Monitor-1, опции мониторинга: в какие дни и время мониторить, присылать результат на почту и т.д.

**ТЕСТИРОВАНИЕ SOAP/XML/WSDL**

На тестовом сайте «https://qahacking.ru/lessons/testovye-wsdl-dlya-testirovaniya-soapui» находим WSDL

Переходим на страницу с WSDL:

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:tns="http://www.dataaccess.com/webservicesserver/" name="NumberConversion" targetNamespace="http://www.dataaccess.com/webservicesserver/">

<types>

<xs:schema elementFormDefault="qualified" targetNamespace="http://www.dataaccess.com/webservicesserver/">

<xs:elemen……..

………

Копируем URL «https://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL»

В Постмане создаём новую Коллекцию, Запрос, вставляем URL

**POST** | https://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL

Тег «Params» 🡪 «Body» 🡪 «(•) raw» + «text ˅ 🡪 XML ˅»

Тег «Body» 🡪 «Headers» 🡪 Проверить или добавить вручную «Content-Type: text/XML»

В тело вставляем:

Для представления WSDL в удобном виде запустить в Chrom 🡪 Widzler

Выбрать из списка предложенные варианты (кот в куче содержаться в данной WSDL):

Numbers to Dollars

Numbers to Word

Часть выбранной WSDL преобразуется в XML формат

Копировать его и вставить в тело запроса POST в Постмане

<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">

<Body>

<NumberToWords xmlns="http://www.dataaccess.com/webservicesserver/">

<ubiNum>**[unsignedLong]**</ubiNum>

</NumberToWords>

</Body>

</Envelope>

Подставить параметр, н-р «25»

<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">

<Body>

<NumberToWords xmlns="http://www.dataaccess.com/webservicesserver/">

<ubiNum>**25**</ubiNum>

</NumberToWords>

</Body>

</Envelope>

«Send»

В теле ответа получим результат:

<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">

<Body>

<NumberToWords xmlns="http://www.dataaccess.com/webservicesserver/">

<ubiNum>**twenty five**</ubiNum>

</NumberToWords>

</Body>

</Envelope>

**ОСНОВНЫЕ КОМАНДЫ, ДЛЯ НАПИСАНИЯ СКРИПТОВ.**

Установка и получение переменных

// глобальные переменные

pm.globals.set(“key”, “value”);

pm.globals.get(“key”);

// переменные окружения

pm.environment.set(“key”, “value”);

pm.environment.get(“key”);

// локальные переменные

pm.variables.set(“key”, “value”);

pm.variables.get(“key”); // если нет локальной, будет искать на уровне выше

Тестирование или asserts

// с использованием анонимной функции и специальных assert конструкций

pm.test(“Название теста”, **function** () {

pm.response.to.be.success;

pm.expect(“value”).to.be.true;

pm.expect(“other”).to.equal(“other”);

});

// с использованием простого условия и массива tests

tests[“Название теста”] = (“a” != “b”);

tests[“Название теста 2”] = true;

Создание запросов

// пример get запроса

pm.sendRequest(“https://postman-echo.com/get”, function (err, res) {

console.log(err);

console.log(res);

});

// пример post запроса

**let** data = {

url: “https://postman-echo.com/post”,

method: “POST”,

body: { mode: “raw”, raw: JSON.stringify({ key: “value” })}

};

pm.sendRequest(data, **function** (err, res) {

console.log(err);

console.log(res);

});

Получение ответа для основного запроса

pm.response.json(); // в виде json

pm.response.text(); // в виде строки

responseBody; // в виде строки

Ответ доступен только во вкладке “Tests”

**NEWMAN**

Newman – тот же Postman только CLI приложение которое позволит запускать тесты с консоли, а значит, что UI и графическая оболочка не нужна и можно интегрировать его в CI.

Нужно сохранить коллекцию тестов, есть два варианта:

* сгенерировать URL – share
* сохранить файлом – export

Перед тем как запускать тесты – надо создать файл – data.json, в котором сохранить переменные – если они могут быть для разных env. Самая простая структура этого файла

[{

"playlist": "url для запроса"

}]

Запустить тесты в консоли:

newman run https://www.getpostman.com/collections/f3579fa0738c702676d1 **(**-e**)** data.json

newman run COLLECTION.postman\_collection.json **(**-e**)** data.json

**SWAGGER**

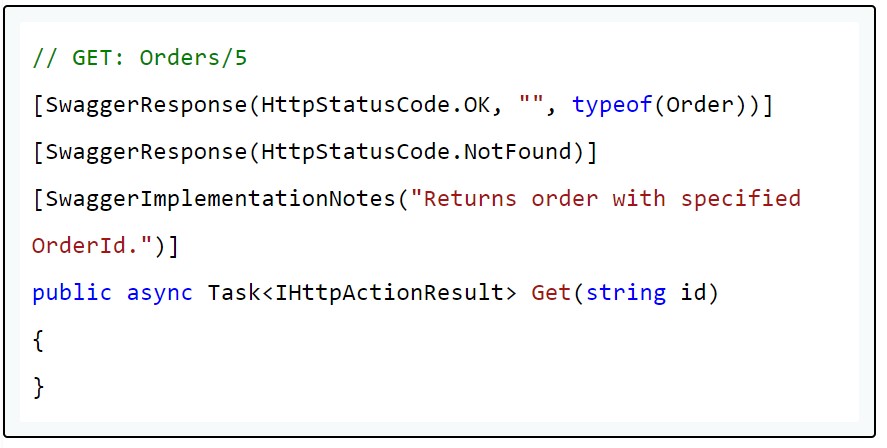
**? Swagger** **– умная документация RESTful web-API**

Swagger – это фреймворк для спецификации RESTful API. Его прелесть заключается в том, что он дает возможность не только интерактивно просматривать спецификацию, но и отправлять запросы – так называемый Swagger UI.

Пример – «https://petstore.swagger.io/» – Полное описание методов, включая модели, коды ответов, параметры запроса – всё наглядно.

**? Как работает Swagger**

Идея в конфигурации отображения с помощью специальных аннотаций у методов API, вот пример:





**? Swagger Codegen**

Генератор кода Swagger-Codegen нужен, чтобы сгенерировать непосредственно клиента или сервер по спецификации API Swagger. Возможности:

Проект Swagger Codegen позволяет генерировать клиентские библиотеки API (SDK generation), Заглушки Сервера и документация автоматически представленную как OpenAPI Spec. В данный момент поддерживается большинство языков и фреймворков:

* API clients: ActionScript, Ada, Apex, Bash, C# (.net 2.0, 3.5 or later), C++ (cpprest, Qt5, Tizen), Clojure, Dart, Elixir, Elm, Eiffel, Erlang, Go, Groovy, Haskell (http-client, Servant), Java (Jersey1.x, Jersey2.x, OkHttp, Retrofit1.x, Retrofit2.x, Feign, RestTemplate, RESTEasy, Vertx, Google API Client Library for Java, Rest-assured), Kotlin, Lua, Node.js (ES5, ES6, AngularJS with Google Closure Compiler annotations) Objective-C, Perl, PHP, PowerShell, Python, R, Ruby, Rust (rust, rust-server), Scala (akka, http4s, swagger-async-httpclient), Swift (2.x, 3.x, 4.x), Typescript (Angular1.x, Angular2.x, Fetch, jQuery, Node)
* Server stubs: Ada, C# (ASP.NET Core, NancyFx), C++ (Pistache, Restbed), Erlang, Go, Haskell (Servant), Java (MSF4J, Spring, Undertow, JAX-RS: CDI, CXF, Inflector, RestEasy, Play Framework, PKMST), Kotlin, PHP (Lumen, Slim, Silex, Symfony, Zend Expressive), Python (Flask), NodeJS, Ruby (Sinatra, Rails5), Rust (rust-server), Scala (Finch, Lagom, Scalatra)
* API documentation generators: HTML, Confluence Wiki
* Configuration files: Apache2
* Others: JMeter

**SoapUI**

**? SOAP-сообщение** — это обычный XML-документ, содержащий следующие элементы:

* Конверт — определяет начало и конец сообщения. Это обязательный элемент.
* Заголовок — содержит любые необязательные атрибуты сообщения, используемые при обработке сообщения, либо в промежуточной точке, либо в конечной точке. Это необязательный элемент.
* Тело — содержит данные XML, содержащие отправляемое сообщение. Это обязательный элемент.
* Неисправность — необязательный элемент неисправности, который предоставляет информацию об ошибках, возникающих при обработке сообщения.

**? SOAP — структура сообщения**

<?xml version = "1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

<SOAP-ENV:Header>

...

...

</SOAP-ENV:Header>

<SOAP-ENV:Body>

...

...

<SOAP-ENV:Fault>

...

...

</SOAP-ENV:Fault>

</SOAP-ENV:Body>

</SOAP\_ENV:Envelope>

**? SoapUI** — это инструмент, который можно использовать как для функционального, так и нефункционального тестирования. Он не ограничивается веб-сервисами, хотя это инструмент де-факто, используемый при тестировании веб-сервисов.

**? SoapUI — Важные функции**

* Он способен выполнять роль как клиента, так и службы.
* Это позволяет пользователям быстро и эффективно создавать функциональные и нефункциональные тесты, используя единую среду.
* Он лицензируется в соответствии с условиями GNU Leaser General Public License (LGPL).
* Он реализован исключительно на платформе JAVA.
* Он поддерживает Windows, Mac, несколько диалектов Linux.
* Это позволяет тестировщикам выполнять автоматизированные функциональные, регрессионные, тесты на соответствие и нагрузочные тесты на различных веб-API.
* Он поддерживает все стандартные протоколы и технологии для тестирования всех видов API.

**? SoapUI — Возможности**

SoapUI богат следующими пятью аспектами:

* Функциональное тестирование
* Тестирование безопасности
* Нагрузочное тестирование
* Протоколы и технологии
* Интеграция с другими инструментами

**? Преимущество SoapUI перед Postman**

В SoapUI по сравнению с Постманом параметры, их значения, тип данных известны заранее – прописаны

**? Что такое WSDL**

WSDL расшифровывается как язык описания веб-сервисов. Это стандартный формат для описания веб-службы.

WSDL — это основанный на XML протокол для обмена информацией в децентрализованной и распределенной среде. Некоторые из других функций WSDL следующие:

* Определения WSDL описывают, как получить доступ к веб-службе и какие операции она будет выполнять.
* Это язык для описания того, как взаимодействовать со службами на основе XML.
* Он является неотъемлемой частью универсального описания, обнаружения и интеграции (UDDI), всемирного бизнес-реестра на основе XML.
* WSDL — это язык, который использует UDDI.

WSDL часто используется в сочетании с SOAP и XML-схемой для предоставления веб-сервисов через Интернет. Клиентская программа, подключающаяся к веб-службе, может прочитать WSDL, чтобы определить, какие функции доступны на сервере. Все используемые специальные типы данных встраиваются в файл WSDL в форме XML-схемы. Затем клиент может использовать SOAP для фактического вызова одной из функций, перечисленных в WSDL.

? Проекты SOAP основаны на WSDL. Нет необходимости начинать с импорта WSDL, но это облегчает тестирование, поскольку WSDL содержит всю информацию, необходимую для тестирования веб-службы, такую ​​как информация о запросах и ответах, их содержимом и многое другое, что упрощает тестирование SoapUI.

**? Работа с SoapUI**

**СОЗДАНИЕ ПРОЕКТА, КОЛЛЕКЦИИ, ЗАПРОСОВ (SOAP-ЗАПРОС)**

* File 🡪 New SOAP Project
* В окне «New Soap Project»:

Project Name: BUGRED (или подставится автоматически, если сперва вставить WSDL)

Initial WSDL: http://users.bugred.ru/tasks/soap/WrapperSoapServer.php?wsdl

Create Requests: 🗹

Create TestSuit: 🞎

Relative Paths: 🞎

«ОК»

* Появится дерево

«BUGRED» Проекта, содержащее WSDL с уже присвоенном названием

«WrapperSoapServerBinding», содержащее уже прописанные действия, в каждый добавлен запрос

[–] «AddTaskInCron»,

Request-1

[–] «CreateCompany»,

Request-1

[–] «CreateTask»,

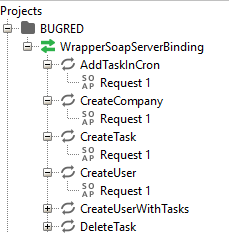
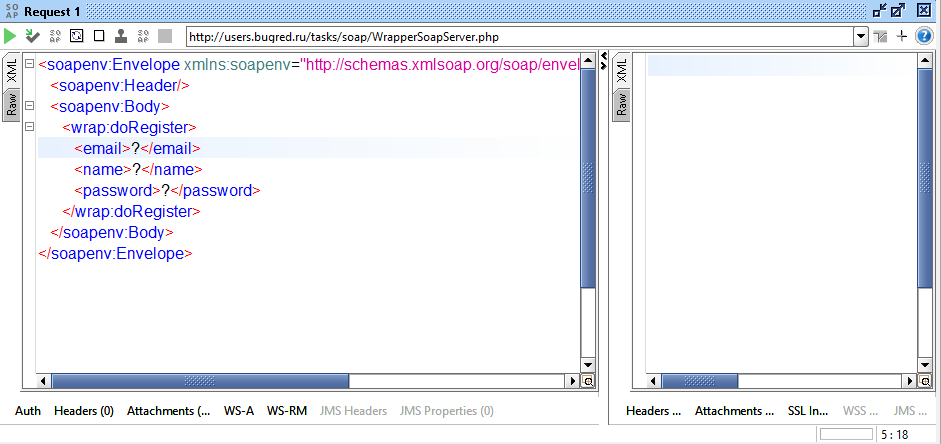
Request-1

[–] «CreateUser»,

Request-1

[+] «…»,

[+] «…»,

* В каждом из запросов уже содержится готовый XML-шаблон, значения параметров отмечены «?»
* Подставив вместо «?» значения – нажать «⯈»
* В правой части окна придёт Ответ, и можно выбрать на тэгах сбоку:
  + «XML» формат – ответ в виде XML
  + «Raw» формат – ответ в виде структуры, со статус кодом, заголовками, телом ответа

**РЕГИСТРАЦИЯ ЮЗЕРА (SOAP-ЗАПРОС)**

* Н-р, как и в Постмане создадим юзера на «users.bugred.ru»
* Создать Проект на WSDL «http://users.bugred.ru/tasks/soap/WrapperSoapServer.php?wsdl» с запросами
* В дереве выбрать действие «doRegister»
* В XML-запросе «Request1» изменить «?» на

<email>antoniy@gmail.com</email>

<name>Antoniy</name>

<password>123456</password>

* Отправить «⯈»
* Получим ответ на XML, с нашими введёнными данными, а на вкладке «Raw»: HTTP/1.1 200 OK

**АВТОРИЗАЦИЯ ЮЗЕРА (SOAP-ЗАПРОС)**

* Теперь, в дереве выбрать действие «doLogin»
* В XML-запросе «Request1» изменить «?» на введённые данные при регистрации:

<email>antoniy@gmail.com</email>

<password>123456</password>

* Отправить «⯈»
* Получим ответ на XML, со значением «true», а на вкладке «Raw»: HTTP/1.1 200 OK

**СОЗДАНИЕ ТЕСТ-СЬЮТОВ И ТЕСТ-КЕЙСОВ (SOAP-ЗАПРОС)**

* Действие «doRegister»
* Правый клик на запросе «Request1»
* «Add to TestCase»
* Диалог создать сперва новый «TestSuite» – Name: BUGRED
* Диалог создать в «TestSuite» новый «TestCase» – Name: Smoke
* Окно «Add Request to TestCase»:

Name: doRegister

Add Soap Request Assertion: 🗹

Shows TestCase Editor: 🗹

(Others) 🞎

* Добавить в Тест-кейс правым кликом ещё запросы: «doLogin», «getUser», «deleteUser»
* Запросы «doRegister», «doLogin», «getUser», «deleteUser» превратились в **Шаги** Тест-кейса

Отличие – внизу окна каждого Шага есть кнопки «Assertions (…)» и «Request Log (…)», а в простых Запросах этих кнопок нет. **Assertions** – это и есть непосредственно сами тесты, проверки

* Открыть один из Шагов: «doLogin»
* Клик на «Assertions (…)» – снизу выпадет поле, для добавление Ассётов
* По умолчанию уже проставлен 1 Ассёрт «● SOAP Request – UNKNOWN»
* Статус «UNKNOWN» – т.к. не было действий
* Отправить «⯈»
* Вне зависимости, какой результат придёт в теле ответа: «true/false» статус будет «VALID», потому что проверка идёт на соответствие ответа формату XML: «● SOAP Request – VALID»

**ДОБАВЛЕНИЕ АССЁРТОВ (SOAP-ЗАПРОС)**

* В окне Шага нажать «**+**»
* Появится окно «Add Assertion», со списком разделов Ассёртов:

Recently Used

Property Content

Compliance, Status and Standards

Script

SLA

JMS

JDBC

Security

* Выбрать из раздела, н-р, «Compliance, Status and Standards» Ассёрт «Valid HTTP status codes»
* Ввести желаемые статус-коды, н-р, «200, 201, 202»
* Ассёрт добавится к списку внизу:

«● SOAP Request – VALID»

«● Valid HTTP status codes – VALID»

* Каждый Ассёрт можно переименовать
* Теперь можно запустить Тест-Кейс, который сам по себе и есть Раннер
* Есть ещё Раннер под правой кнопкой, но его надо подгружать из файла

**СОЗДАНИЕ ПРОЕКТА, КОЛЛЕКЦИИ, ЗАПРОСОВ (REST-ЗАПРОС)**

* File 🡪 New REST Project
* В окне «New Rest Project»:

URI: http://users.bugred.ru/tasks/rest/

* Окно «Request 1», с методом GET по умолчанию, и автоматич. разбитым URI на «Endpoint» и «Resource»
* Заменить метод на POST
* В поле «Resource» дописываем действие «doregister»
* В область «Name – Value» вводим параметры и значения (параметры узнать из спеки)

name MarcusLucius

email marcuslucius@gmail.com

password 123456

* В поле «Parameters» значения пропишутся автоматически:  
  «?name=MarcusLucius&email=marcuslucius@gmail.com&password=123456»
* В поле Ответа Переключится на тег «JSON»
* Отправить «⯈»

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Request 1 | | | | | | | | | | |
| ⯈■√ | | | Method | | Endpoint | | | Resource | | Parameters |
| GET ˅ | | http://users.bugred.ru | | | /tasks/rest/doregister | | ?name=MarcusLucius&email=ma….. |
|  | Requet | **+ ×** | | | | | | XML | {  "name": "MarcusLucius",  "avatar": "http://users.bugred.ru//tmp/default\_avatar.jpg",  "password": "ba3253876aed6bc22d4a6ff53d8406c6ad864195ed144ab5c87621b6c233b548baeae6956df346ec8c17f5ea10f35ee3cbc514797ed7ddd3145464e2a0bab413",  "birthday": 0,  "email": "marcuslucius@gmail.com",  "gender": "",  "date\_start": 0,  "hobby": ""  } | |
| Name | | Value | | Style | Level |
|  | |  | |  |  |
| name | | MarcusLucius | |  |  | JSON |
| Raw | email | | marcus@gmail.com | |  |  |
| password | | 123456 | |  |  | HTML |
|  | |  | |  |  |
|  |  | | | | | | Raw |
|  |
|  |  |

* Дерево имеет вид:

🖿 REST project BUGRED

**└─⮀** http://users.bugred.ru

└─ 🞎 Rest [/tasks/rest/doregister]

└─ ■ Rest 1

└─ ⯏Request 1

* Добавить новый запрос получится только с уровня «**⮀**http://users.bugred.ru» (правый клик 🡪 «New Resource»), иначе, и Ресурс «/tasks/rest/doregister», и Параметры «?name=MarcusLucius&email=ma…..» будут меняться и на всех уровнях ниже:  
  🞎 Rest [/tasks/rest/doregister]

■ Rest 1

⯏Request 1

**СОЗДАНИЕ ТЕСТ-СЬЮТОВ И ТЕСТ-КЕЙСОВ (REST-ЗАПРОС)**

* Открыть Запрос
* Нажать на неприметную галочку рядом с «Method» (рядом с кнопками «Play» и «Stop»)
* Диалог создать сперва новый «TestSuite» – Name: REST
* Диалог создать в «TestSuite» новый «TestCase» – Name: Smoke
* Окно «Add Request to TestCase»:

Name: doRegister

Close Request Window: 🗹

Shows TestCase Editor: 🗹

Shows TestCase Editor: 🗹

* Добавить в Тест-кейс правым кликом ещё запросы: «doLogin», «getUser», «deleteUser»
* Запросы «doRegister», «doLogin», «getUser», «deleteUser» превратились в Шаги Тест-кейса

Отличие – внизу окна каждого Шага есть кнопки «Assertions (…)» и «Request Log (…)», а в простых Запросах этих кнопок нет. Assertions – это и есть непосредственно сами тесты, проверки

* Открыть один из Шагов: «doLogin»
* Клик на «Assertions (…)» – снизу выпадет поле, для добавления Ассётов
* По умолчанию, в отличие от SOAP-запроса, Ассёрты не проставлены.

**ДОБАВЛЕНИЕ АССЁРТОВ (REST-ЗАПРОС)**

* В окне Шага нажать «**+**»
* Появится окно «Add Assertion», со списком разделов Ассёртов:

Recently Used

Property Content

Compliance, Status and Standards

Script

SLA

JMS

JDBC

Security

* Выбрать из раздела, н-р, «Compliance, Status and Standards» Ассёрт «Valid HTTP status codes»
* Ввести желаемые статус-коды, н-р, «200, 201, 202»
* Ассёрт добавится к списку внизу:

«● Valid HTTP status codes – VALID»

* Каждый Ассёрт можно переименовать
* Теперь можно запустить Тест-Кейс, который сам по себе и есть Раннер
* Есть ещё Раннер под правой кнопкой, но его надо подгружать из файла

**ДОБАВЛЕНИЕ ПЕРЕМЕННЫХ (REST-ЗАПРОС)**

На каждом ур-не (Проект, Тест-Сьют, Тест-Кейс) можно добавить переменные

* File 🡪 New SOAP Project
* В окне «New Soap Project»:
* Initial WSDL: https://qahacking.ru/lessons/testovye-wsdl-dlya-testirovaniya-soapui
* Имя присваивается автоматически из WSDL: «Numberconversion»
* Дабл-клик на Проекте «Numberconversion»
* В нижней части окна «Numberconversion» нажать «**+**»
* Ввести произвольную переменную

|  |  |
| --- | --- |
| Name | Value |
| name | ABC |

* В дереве выбрать Действие «NumberToDollars»
* В «Request 1» вместо «?» подставить «12»
* В Ответе получим «Twelve Dollars»
* В дереве: Правый клик на запросе «Request1»
* «Add to TestCase»
* Создать Тест-Сьют и Тест-Кейс с Шагом «Request1»
* В дереве: Правый клик на шагах «TestSteps»
* «Add Step» 🡪 «Add Groovy Script»
* Появится окно Groovy-скрипта «★Groovy» (Groovy – чувствителен к регистру!)
* Ввести скрипт: **log.info "Hello World!"**, где «**log.info**» означает «вывести в консоль»
* Выполнить «⯈»
* Снизу в окне скрипта в консоли придёт ответ: Mon Dec 20 15:07:25 EET 2021:INFO:Hello World!
* Имеется заданная переменная:

|  |  |
| --- | --- |
| Name | Value |
| name | ABCD |

* В скрипте задаём (define) имя переменной (где «name» в скрипте совпадает со значением колонки «Name: name» из таблицы): def name
* Присваиваем переменной ссылку на переменную проекта: **def name=context.expand("${#Project#Name}")**
* Выводим переменную в консоль: **log.info ("Project property name is " + name)**
* Запрос и Ответ в консоли:

**log.info "Hello World!"**

**def name=context.expand("${#Project#Name}")**

**("Project property name is " + name)**

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

Mon Dec 20 16:24:14 EET 2021:INFO:Hello World!

Mon Dec 20 16:24:14 EET 2021:INFO:Project property name is ABCD

* Можно посмотреть расположение программы SoapUI на компьютере:

**def soapLocation=context.expand("${#System#user.home}")**

**("Location of the SoapUI program is " + soapLocation)**

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

Mon Dec 20 16:36:03 EET 2021:INFO:Location of the SoapUI program isC:\Users\Admin1

* Можно добавить переменную на уровне Тест-Сьюта
* Дабл-клик по «TestSuite»
* В нижней части окна «TestSuite» нажать «**+**»
* Ввести произвольную переменную

|  |  |
| --- | --- |
| Name | Value |
| TSproperty | Odessa |

* Задать в скрипте переменную для Тест-сьюта внутри Тест-кейса, внутри Тест-Раннера («**def variable**») и вывести её («**log.info** »)

**def variable = testRunner.testCase.testSuite.getPropertyValue ("TSProperty")**

**log.info ("TestSuite Property is " + variable)**

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

Mon Dec 20 17:16:47 EET 2021:INFO:TestSuite Property is Odessa

* Если «**variable**» взять в кавычки, то «variable» выведется просто как текст, а не как переменная, с помощью «+» можно добавлять новые «склейки» текста для вывода:

**log.info ("TestSuite Property is " + "variable " + "Qwerty")**

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

Mon Dec 20 17:16:47 EET 2021:INFO:TestSuite Property is variable Qwerty

* Можно Груви-скриптом прописать проверки тестов, которые использовать как Ассёрты.
* Использовать переменную «status», присвоить ему значение имени Шага «NumberToDollars - Request 1»

**def status = testRunner.runTestStepByName ("NumberToDollars - Request 1")**

**log.info ("Status of the test step is " + status)**

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

Mon Dec 20 17:42:36 EET 2021:INFO:Status of the test step is com.eviware.soapui.impl.wsdl.teststeps.WsdlTestRequestStepResult@204a1346

* Получаем ответ, но он зашифрован
* Ввести ещё одну переменную «result», которая будет принимать значение переменной «status» по методу «getStatus» текстовой строкой «toString»: def result = status.getStatus().toString() и вывести в консоль «log.info»
* Получаем расшифровку: ОК

**def status = testRunner.runTestStepByName ("NumberToDollars - Request 1")**

**log.info ("Status of the test step is " + status)**

**def result = status.getStatus().toString()**

**log.info ("The result is " + result)**

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

Mon Dec 20 17:42:36 EET 2021:INFO:Status of the test step is com.eviware.soapui.impl….epResult@204a1346

Mon Dec 20 17:58:30 EET 2021:INFO:The result is OK

* Можно добавить с помощью «if» и «else» некоторое значение, которое будет выводиться при определённом условии, н-р, когда статус совпадает или нет с «ОК»

**def status = testRunner.runTestStepByName ("NumberToDollars - Request 1")**

**log.info ("Status of the test step is " + status)**

**def result = status.getStatus().toString()**

**log.info ("The result is " + result)**

**if (result=="OK")**

**{log.info ("Congrats! Yr test passed!")}**

**else**

**{log.info ("Damn it! This is weird")}**

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

Mon Dec 20 17:42:36 EET 2021:INFO:Status of the test step is com.eviware.soapui.impl….epResult@204a1346

Mon Dec 20 17:58:30 EET 2021:INFO:The result is OK

Mon Dec 20 18:19:19 EET 2021:INFO:Congrats! Yr test passed!

* Если в «if» прописать результат отличный от ОК: «555»

**def status = testRunner.runTestStepByName ("NumberToDollars - Request 1")**

**log.info ("Status of the test step is " + status)**

**def result = status.getStatus().toString()**

**log.info ("The result is " + result)**

**if (result=="OK")**

**{log.info ("Congrats! Yr test passed!")}**

**else**

**{log.info ("Damn it! This is weird")}**

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

Mon Dec 20 17:42:36 EET 2021:INFO:Status of the test step is com.eviware.soapui.impl….epResult@204a1346

Mon Dec 20 17:58:30 EET 2021:INFO:The result is OK

Mon Dec 20 18:20:45 EET 2021:INFO:Damn it! This is weird

**CURL**

**? Curl (Client URL) – отправить запросы на сервер из Командной строки**

Curl (Client URL) – это утилита командной строки, которая позволяет выполнять HTTP-запросы с различными параметрами и методами. Вместо того, чтобы переходить к веб-ресурсам в адресной строке браузера, можно использовать командную строку, чтобы получить те же ресурсы, извлеченные в виде текста.

Утилита доступна в большинстве систем на основе Unix и предназначена для проверки подключения к URL-адресам. Кроме того команда Curl — отличный инструмент передачи данных. Curl работает на libcurl, которая является бесплатной библиотекой для передачи URL на стороне клиента.

**? Использование Curl**

* БРАУЗЕР: Используем тренировочный сайт по запросам JSON: https://jsonplaceholder.typicode.com
* БРАУЗЕР: Добавляем «/posts»: https://jsonplaceholder.typicode.com/posts
* БРАУЗЕР: Приходит ответ со списком пользователей:

[

{

"userId": 1,

"id": 1,

"title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",

"body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"

},

{

"userId": 1,

"id": 2,

…

…

* Запустить CLI
* CLI: Команда: **curl https://jsonplaceholder.typicode.com/posts**
* CLI: Получаем тот-же результат со списком пользователей в командной строке:

**[**

**{**

**"userId": 1,**

**"id": 1,**

**"title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",**

**"body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"**

**},**

**{**

**"userId": 1,**

**"id": 2,**

**…**

**…**

* CLI: Для получения только юзера с id 20: **curl https://jsonplaceholder.typicode.com/posts/20**

**{**

**"userId": 2,**

**"id": 20,**

**"title": "doloribus ad provident suscipit at",**

**"body": "qui consequuntur ducimus possimus quisquam amet similique\nsuscipit porro ipsam amet\neos veritatis officiis exercitationem vel fugit aut necessitatibus totam\nomnis rerum consequatur expedita quidem cumque explicabo"**

**}**

* CLI: Опция «i» – получение информации о заголовках + само тело: **curl -i https://jsonplaceholder.typi…**

**C:\Users\Admin1>curl -i https://jsonplaceholder.typicode.com/posts/20**

**HTTP/1.1 200 OK**

**Date: Mon, 20 Dec 2021 17:10:39 GMT**

**Content-Type: application/json; charset=utf-8**

**Content-Length: 308**

**Connection: keep-alive**

**x-powered-by: Express**

**…**

**…**

**…**

**{**

**"userId": 2,**

**"id": 20,**

**"title": "doloribus ad provident suscipit at",**

**"body": "qui consequuntur ducimus possimus quisquam amet similique\nsuscipit p**

**orro ipsam amet\neos veritatis officiis exercitationem vel fugit aut necessitati**

**bus totam\nomnis rerum consequatur expedita quidem cumque explicabo"**

**}**

* CLI: Опция «I» – получение только информации о заголовках: **curl -I https://jsonplaceholder.typi…**

**C:\Users\Admin1>curl -i https://jsonplaceholder.typicode.com/posts/20**

**HTTP/1.1 200 OK**

**Date: Mon, 20 Dec 2021 17:10:39 GMT**

**Content-Type: application/json; charset=utf-8**

**Content-Length: 308**

**Connection: keep-alive**

**x-powered-by: Express**

**…**

**…**

**…**

* CLI: Опция «o» – сохранить ответ в файл «File.txt»:   
  **curl -o File.txt https://jsonplaceholder.typi…/posts**
* CLI: Опция «O» – сохранить ответ в текстовый файл «posts» без расширения:   
  **curl -O https://jsonplaceholder.typi…/posts**
* CLI: Обновить пользователя id 20: Опция «X» – метод запроса HTTP, опция «d “…”» обновить данные (параметры смотрим в списке пользователей)  
  **curl -X PUT -d “title=MarcusLucius&body=Emperor” https://jsonplaceholder.typicode.com/posts/20**

**C:\Users\Admin1>curl -X PUT -d "title=MarcusLucius&body=Emperor" https://jsonplaceholder.typicode.com/posts/20**

**{**

**"title": "MarcusLucius",**

**"body": "Emperor",**

**"id": 20**

**}**

* CLI: Создать пользователя (id будет: последний+1): опция «- - data “…”» создать данные (параметры смотрим в списке пользователей)  
  **curl --data “title=MiramacOrdus&body=Wiper” https://jsonplaceholder.typicode.com/posts/**

**C:\Users\Admin1>curl -X PUT -d "title=MarcusLucius&body=Emperor" https://jsonplaceholder.typicode.com/posts/20**

**{**

**"title": "** **MiramacOrdus ",**

**"body": "** **Wiper ",**

**"id": 101**

**}**

* CLI: Загрузить картинку
* БРАУЗЕР: копировать URL картинки «https://i.imgur.com/IBItATn.png»
* CLI: Загрузить картинку: curl -O https://i.imgur.com/IBItATn.png

**Performance**

? Тестирование производительности:

* Тестирование ёмкости/способностей (Capacity testing)
* Стрессовое (Stress testing)
* Нагрузочное (Load testing)
* Объёмное тестирование (Volume testing)
* Выносливости (Soak/Endurance testing)
* Стабильности/надёжности (Stability / Reliability testing)
* Шиповое (Spike)
* Отказоустойчивости (Stability testing)
* Масштабируемости (Scalability test)

**? Что такое тестирование производительности**

Это класс тестирования ПО, который фокусируется на производительности системы при определенной нагрузке. Он не ищет напрямую ошибки или дефекты. Он производит аналитику на основе эталонных тестов и предоставляет разработчику всю диагностическую информацию, необходимую для выявления проблем производительности и узких мест. При этом происходит:

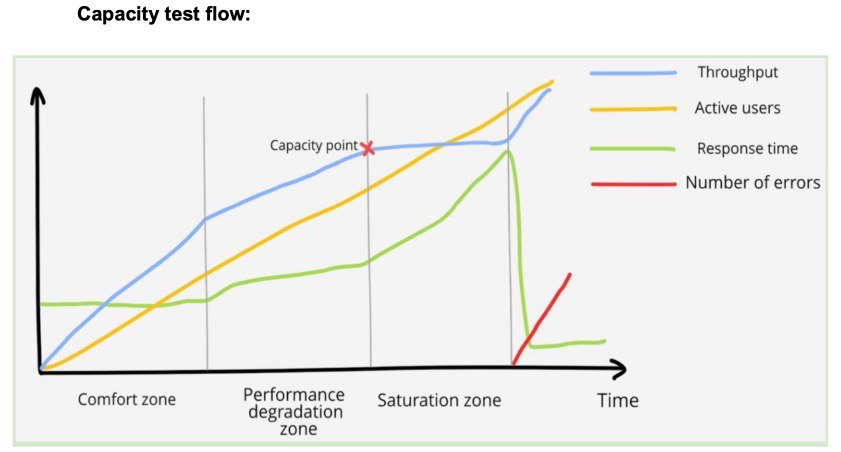
* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* определение количества пользователей, одновременно работающих с приложением
* определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
* исследование производительности на высоких, предельных, стрессовых нагрузках

Важно понимать, что все подвиды тестирования производительности это, грубо говоря, одно и то же, просто в зависимости от конкретного подвида выбираются разные параметры (показатели нагрузки/пользователей, длительности и т.п.) и собираются соответствующие метрики. Точкой отсчета принято брать результаты Capacity testing.

В реальном мире проводят только часть из перечисленных подвидов в соответствии с бюджетом и приоритетами данного ПО, а параметры тестов и метрики могут корректироваться в разных ситуациях.

**? Тестирование ёмкости/способностей? (Capacity)**

Capacity — базовый тест, который обычно выполняется первым. Все последующие тесты на среднее время ответа, регенерацию системы, утилизацию ресурсов нужно выполнять с оглядкой на результаты Capacity. Ёмкость системы измеряется в rps (requests per second). Используемый подход: ступенчато повышаем нагрузку до момента, когда время ответа начнет расти экспоненциально. Экспоненциальный рост времени ответа, как правило, связан с полной утилизацией одного из ресурсов, из-за которого запросы вместо моментальной обработки выстраиваются друг за другом и ждут своей очереди на обработку.



Capacity point – точка, где перестает расти Throughput, но увеличивается Response Time, то есть система перестает справляться с запросами.

Исходя из этого тестирования выбираются значения для stress, load и soak/endurance тестов. Для stress берется значение близкое к capacity point, но немного меньше. Для load количество пользователей из зоны деградации.

Важно, ваша цель, не получить кол-во rps, при котором все взрывается, а понять, какой именно ресурс станет узким местом, при повышении нагрузки. Поэтому, если обстреливаемый вами сервер не покрыт мониторингом — можете даже не начинать тест. Общий подход к сбору телеметрии — чем больше метрик собирается, тем лучше.

В некоторых случаях Capacity называют так же Scalability (масштабируемость), но в целом это не равнозначные понятия.

**? Что означает тестирование масштабируемости? (Scalability)**

Профиль нагрузки тот же, что и при нагрузочном тестировании. Что получаем в результате? Ответы на следующие вопросы:

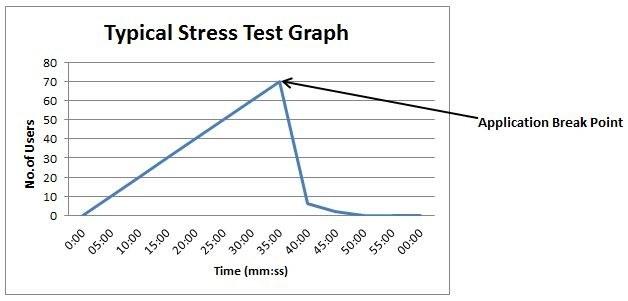
* Увеличится ли производительность приложения, если добавить дополнительные аппаратные ресурсы?
* Увеличится ли производительность пропорционально количеству добавленных аппаратных средств?
* Разница между тестированием ёмкости/способностей и тестированием масштабируемости? (Capacity vs Scalability)

Тестирование масштабируемости – это тестирование программного приложения для измерения его способности увеличивать или уменьшать масштаб с точки зрения любых его нефункциональных возможностей. Тестирование производительности, масштабируемости и надежности обычно группируется аналитиками качества ПО.

Тестирование ёмкости измеряет, сколько пользователей может обработать приложение. Это подмножество тестирования масштабируемости, в котором при тестировании масштабируемости вы получите меру ёмкости приложения. Тестирование масштабируемости измеряет, насколько хорошо приложение справляется с растущим числом пользователей. Если вы тестируете масштабируемость до тех пор, пока приложение не выйдет из строя, у вас будет мера того, сколько пользователей (ёмкость) может обработать приложение.

**? Стрессовое тестирование? (Stress testing)**

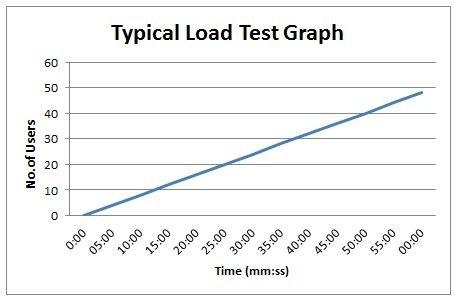
Стрессовое тестирование выполняется самым первым, если нет отдельного Capacity тестирования, хотя по факту это все равно будет Capacity, т.к. нагрузка берется «с потолка». Позволяет проверить насколько приложение и система в целом работоспособны в условиях высокой нагрузки. Нагрузка на систему будет возрастать непрерывно до тех пор, пока не будет достигнут один из критериев его остановки. Пример: стресс-тест банковской системы был остановлен при превышении отметки в 1500 пользователей, когда высокая загруженность процессора (более 80%) привела к увеличению среднего времени отклика в пять раз и массовому появлению ошибок HTTP(S).



**? Нагрузочное тестирование? (Load)**

Нагрузочное тестирование - это тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе. Этот тип тестирования производительности выполняется для диагностики поведения системы при увеличении рабочей нагрузки.

Нагрузка на систему обычно подается на протяжении 1-2 часов (в некоторых источниках: 4-8 часов, хотя это уже больше endurance), количество пользователей для нагрузочного теста берется из зоны деградации (в некоторых источниках: определяется в количестве 80% от результата максимальной производительности). В это время собираются метрики производительности: количество запросов в секунду, транзакций в секунду, время отклика от сервера, процент ошибок в ответах, утилизация аппаратных ресурсов и т. д.



**? Объёмное тестирование (Volume testing)**

Объемное тестирование предназначено для прогнозирования того, может ли система / приложение обрабатывать большой объем данных. Это тестирование сосредоточено на наполнении базы данных продукта в реальных сценариях использования.

Пример 1: отправка через POST-запросы очень большого количества данных.

Пример 2: как изменится производительность приложения спустя X лет, если аудитория приложения вырастет в Y раз?

**? Тестирование выносливости/стабильности/надежности (Soak/Endurance/Stability/Reliability testing)**

Задачей тестирования стабильности является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит проверка на утечки памяти, время отклика, правильность подключения и закрытия подключения к модулям (например, БД) и т.п. в течение длительного времени, чтобы гарантировать, что после длительного периода время отклика системы останется таким же или лучше, чем на начало теста. Этот тип тестирования выполняется в самом конце (а где-то по ночам). Так же он помогает управлять будущими нагрузками, ведь нам необходимо понять, сколько дополнительных ресурсов (таких как ЦП, емкость диска, использование памяти или пропускная способность сети) необходимы для поддержки использования в будущем.

**?Спайк/шиповое тестирование? (Spike)**

Этот вид тестирования предназначен для определения поведения системы при внезапном увеличении нагрузки (большого количества пользователей) на систему. Например, дни распродаж в интернет-магазине.

**? Тестирование устойчивости? (Resilence)**

Проверка устойчивости проводится для того, чтобы убедиться, что система способна вернуться в исходное состояние после кратковременного напряжения. Например, если в интернет-магазине действует скидка на определенные товары на короткое время, скажем, один час в день.

**JMeter**

**? Apache JMeter** — инструмент для проведения нагрузочного тестирования

Возможно создание большого количества запросов с помощью нескольких компьютеров при управлении этим процессом с одного из них. Архитектура, поддерживающая плагины сторонних разработчиков, позволяет дополнять инструмент новыми функциями. В программе реализованы механизмы авторизации виртуальных пользователей, поддерживаются пользовательские сеансы. Организовано логирование результатов теста и разнообразная визуализация результатов в виде диаграмм, таблиц и т. п.

**? Одна из главных метрик** – время отклика (в мс)

**? Работа с JMeter**

**СОЗДАТЬ ПРОЕКТ**

Каждый новый проект – открывается вместо старого. 2 проекта в рабочем пространстве быть не могут.

Практически вся работа происходит через правый клик.

В корне дерева будет «Символ иконы "Векторная лаборатория" — стоковый вектор TestPlan» – это как Коллекция в Postman или Проект в SoapUI

К Тест-Плану добавляем Группу, с которой будем работать:

В дереве «Символ иконы "Векторная лаборатория" — стоковый вектор TestPlan» (правый клик) 🡪 Add 🡪 Thread (Users) 🡪 Thread Group

В окне «Thread Group»:

|  |  |  |
| --- | --- | --- |
| Name: | Group1 Имя Группы | |
| Comments: | Комменты | |
| Action to be taken after a Sampler error ---------------------------------------------------------------------------------- | | |
| 🞊Continue ⭘Start next thread Loop ⭘Stop Thread ⭘Stop Test ⭘ Stop Test Now | | |
| Thread Properties: --------------------------------------------------------------------------------------------------------------- | | |
| Number of Threads (users): | | 10 Кол-во пользователей, кот будут отправлять запрос (н-р 100, 7088, …) |
| Rump-up period (seconds): | | 1 С какой периодичностью пользователи будут отправлять запросы |
| Loop Count: 🞎 Infinity | | 5 Кол-во итераций запроса (если 🗹 – зациклена итерация) |
| 🗹 Same user on each iteration | | |
| 🞎 Delay Thread creation until needed | | |
| 🞎 Specify Thread lifetime | | |

**СОЗДАТЬ ЗАПРОС**

В дереве «🟓Group1» 🡪 Add 🡪 Sampler 🡪 HTTP Request (можно выбрать запросы разных типов)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name: | | | | | HTTP-Request Имя Запроса | | | | | | | | | | | | | | | | | | |
| Comments: | | | | | Комменты | | | | | | | | | | | | | | | | | | |
| Basic | | Advanced | | | | | | |  | | | | | | | | | | | | | | |
| Web Server --------------------------------------------------------------------------------------------------------------------------- | | | | | | | | | | | | | | | | | | | | | | | |
| Protocol [http]: | | | | | | http/ https | | | | | Server name or IP: | | | | mysite.net | | | Port Number: | | | | (XXXX) | |
| HTTP Request: ----------------------------------------------------------------------------------------------------------------------- | | | | | | | | | | | | | | | | | | | | | | | |
| GET | 🞃 | | Path: | | | | /menu/adults/page1 | | | | | | | | | | | Content Encoding: | | | | |  |
| 🞎 Redirect Automatically 🗹 Follow Redirects 🗹 Use Keep Alive 🞎 Use multipart/form-data 🞎 Browser-compatible headers | | | | | | | | | | | | | | | | | | | | | | | |
| Parameters | | | | Body Data | | | | | | Files Upload | | | (в теге Body можно отправить JSON, в теге Files Upload – н-р, картинку) | | | | | | | | | | |
| Send Parameters With the Request | | | | | | | | | | | | | | | | | | | | | | | |
| Name | | | | | | | | Value | | | | | | URL Encode? | | Content Type | | | | Include Equals? | | | |
| В теге Parameters можно отправлять переменные и значения в API, вбиваются они сюда | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | Detail | | | | Add | | Add form Clipboard | | Delete | Up | | Down | |  | | |

Добавился «🎟HTTP Request» запрос на сайт

JMeter обязует сохранить Запрос: «Save»

Start «⯈»

Запрос на сайт выполнился, имитируя 10 пользователей, заходивших через 1 сек; всё это повторилось 5 раз.

Если прилетели ошибки (вверху справа) – можно посмотреть их, кликнув «▲» – внизу откроется вкладка.

Чтобы понять, что происходило в момент такого Запроса, нужен некий отчёт.

Для этого добавляются сущности Слушатели «Listener».

**ДОБАВИТЬ ЛИСТЕНЕРА**

В дереве: Group1 🡪 Add 🡪 Listener 🡪 (выбрать) View Results in Table

Добавился Слушатель, выделить его «⯋◗View Results in Table»

Start «⯈»

Идёт Запрос, в это время результаты начинают приходить в виде таблицы, где можно видеть номер и время подзапросов, кол-во пользователей за итерацию (1 – 10), отклик в мс, статус, сколько байт отправлено, Letency время от момента отправления запроса до начала получения данных и т.д.

Это своего рода отчёт.

Добавим ещё Слушателя: Group1 🡪 Add 🡪 Listener 🡪 View Results in Tree

«⯋◗View Results in Tree»

Start «⯈»

Обрабатывается запрос – результаты будут в виде Дерева, по содержанию схожи с «View Results in Table»

Добавим Слушателя: Group1 🡪 Add 🡪 Listener 🡪 Graph Results

«⯋◗Graph Results»

Start «⯈»

Обрабатывается запрос – результаты будут в виде Графика, по оси «х» кол-во самплов, по оси «у» время

Но метод не очень наглядный при малом кол-ве имитируемых пользователей

Перейти к «Group1»

Увеличить кол-во юзеров с 10 до 500 или 1000чел, и поставить кол-во итераций: бесконечность

Вернуться к «Graph Results»

Там точками будут выстраиваться разные графики.

В это время в браузере сайт начнёт работать хуже и хуже и в конце концов упадёт.

Stop «⯃» – чтобы остановить процесс: прекратить имитировать 100500 пользователей и разгрузить сайт.

**ДОБАВИТЬ АССЁРТ**

Надо добавить саму проверку – Ассёрт, где можно сравнить результаты.

(Предварительно уменьшим в «🟓Group1» пользователей со 100500 до 5, время 1сек, 1 итерация)

В дереве: Group1 🡪 Add 🡪 Assertion 🡪 (выбрать) View Results in Table

Добавилась Проверка, выделить её «🔾Response Assertion»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name: | Response Code - 200 Имя Ассёрта | | | | | | | |
| Comments: | Комменты | | | | | | | |
| Apply to: --------------------------------------------------------------------------------------------------------------------------- | | | | | | | | |
| ⭘Main sample and sub-samples 🞊Main sample only ⭘Sub-samples only ⭘JMeter variable name to use | | | | | | | | |
| Field to Test: ----------------------------------------------------------------------------------------------------------------------- | | | | | | | | |
| ⭘Text Response 🞊 Response Code ⭘ Response Message ⭘ Response Headers | | | | | | | | |
| ⭘Request Headers ⭘URL Sampled ⭘Document (text) ⭘Ignore Status | | | | | | | | |
| ⭘ Request Data | | | | | | | | |
| Pattern Matching Rules---------------------------------------------------------------------------------------------------------- | | | | | | | | |
| ⭘ Contains ⭘Matches 🞊Equals ⭘Substring 🞎 Not 🞎 Or | | | | | | | | |
| Patterns to Test---------------------------------------------------------------------------------------------------------- | | | | | | | | |
| Patterns to Test | | | | | | | | |
| 200 | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | |  | Add | Add form Clipboard | Delete |  |  |  |

В окне «Response Assertion» указать:

Имя и комментарии Проверки – «Response Code – 200»

Будет она применяться только к запросу или к подзапросам – Main sample only

Выбрать поле для теста – Response Code

Правила работы паттернов – Equals

«Add»

Паттерны (ожидаемый результат) – 200

Получаем вводные данные:

Пользователей: 5

Периодичность: 1с

Итераций: 1

Проверка: Ответа сервера

Применить к: Только главному запросу

Что именно проверяем: Статус код

Правило: Полное совпадение

Значение: 200

Start «⯈»

Результаты проверки можно посмотреть в Листенерах, н-р «⯋◗View Results in Table», где непрошедшие запросы будут помечены красным, с описанием фактического результата и несовпадения

Чтобы очистить поле истории Ответов (вверху по центру) нажать «Clear» (1 метёлка) или «Clear All» (2 метёлки)

Добавим Ассёрт на Время отклика

В дереве: Group1 🡪 Add 🡪 Assertion 🡪 (выбрать) Duration Assertions

Cмотрим среднее время отклика в Листенере «View Results in Table» (если нету спеки или опыта)

Добавилась Проверка, выделить её «🔾Duration Assertions»

|  |  |  |
| --- | --- | --- |
| Name: | Duration time less than 500ms Имя Ассёрта | |
| Comments: | Комменты | |
| Apply to: --------------------------------------------------------------------------------------------------------------------------- | | |
| ⭘Main sample and sub-samples 🞊Main sample only ⭘Sub-samples only | | |
| Duration to Assert: ----------------------------------------------------------------------------------------------------------- | | |
| Duration in milliseconds: | | 510 |

Получаем вводные данные:

Пользователей: 5

Периодичность: 1с

Итераций: 1

Проверка: Время отклика сервера

Применить к: Только главному запросу

Время отклика: 510мс

Листенер «View Results in Table»

Start «⯈»

Смотрим результаты

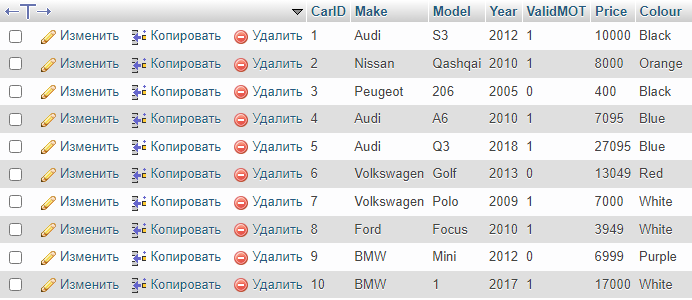
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sample # | Start Time | Hread Name | Label | Sample Time (ms) | Status | Bytes | Sent Bytes | Latency | Connect Time (ms) |
| 1 | 00:20:50.433 | Group1 1-1 | HTTP Request | 516 | ⯂ | 113798 | 116 | 375 | 81 |
| 2 | 00:20:50.648 | Group1 1-2 | HTTP Request | 514 | ⯂ | 113798 | 116 | 365 | 75 |
| 3 | 00:20:50.831 | Group1 1-3 | HTTP Request | 483 | ⯂ | 113798 | 116 | 347 | 71 |
| 4 | 00:20:51.031 | Group1 1-4 | HTTP Request | 502 | ⯂ | 113798 | 116 | 372 | 81 |
| 5 | 00:20:51.230 | Group1 1-5 | HTTP Request | 464 | ⯂ | 113798 | 116 | 345 | 66 |

Там, где время отклика > 500мс – статус с красным щитом – проверки не прошли

**ЗАПРОС К БАЗЕ ДАННЫХ**

Должна быть готова БД на MySQL сайте

(сайт: https://www.db4free.net / БД: andrey1981 / таблица: CARS)



И установлен JDBC-driver:

Link to download jdbc driver: https://dbschema.com/jdbc-driver/MySql.html

Unzip 1 of 1 file: mysql-connector-java-8.0.25

copy it to folder(s): jmeter/extracts or jmeter/bin or jmeter/lib

Restart JMeter

Добавим к Тест-Плану новую Группу «Group2»: Test Plan 🡪 Add 🡪 Thread (Users) 🡪 Thread Group

Добавим Запрос к Базе Данных: Group2 🡪 Add 🡪Sampler 🡪 JDBC Request

Для настройки соединение с БД: Group2 🡪 Add 🡪Config Element 🡪 JDBC Connection Configuration

Добавим Слушателя: Group1 🡪 Add 🡪 Listener 🡪 View Results in Tree

«Символ иконы "Векторная лаборатория" — стоковый вектор TestPlan»

«🟓Group1»

«🎟HTTP Request»

«⯋◗View Results in Table»

«⯋◗View Results in Tree»

«⯋◗Graph Results»

«🔾Response Assertion»

«🔾Duration Assertion»

«🟓Group2»

«🎟JDBC Request»

«🛠JDBC Connection Configuration»

«⯋◗View Results in Tree»

Окно «🛠JDBC Connection Configuration»

|  |  |  |
| --- | --- | --- |
| Name: | JDBC Connection Configuration (по умолчанию) | |
| Comments: |  | |
| Variable Name Bound to Pool: ------------------------------------------------------------------------------------------------- | | |
| Variable Name for created pool: | | var (ввести произвольную переменную) |
| Connection Pool Configuration: ------------------------------------------------------------------------------------------------- | | |
| Max Number of Connections: | | 0 (по умолчанию) |
| Max Wait (ms): | | 10000 (по умолчанию) |
| Time Between Eviction Runs (ms): | | 60000 (по умолчанию) |
| Auto Commit: | | True (по умолчанию) |
| Transaction Isolation: | | DEFAULT (по умолчанию) |
| Preinit Pool: | | False (по умолчанию) |
| Init SQL statements separated by new line | | |
| Connection Validation by Pool: ------------------------------------------------------------------------------------------------- | | |
| Test While Idle: | | True (по умолчанию) |
| Soft Min Evictable Idle Time (ms): | | 5000 (по умолчанию) |
| Validation Query: | |  |
| Database Connection Configuration: ---------------------------------------------------------------------------------------- | | |
| Database URL: | | jdbc:mysql://db4free.net:3306/andrey1981 (запрос:тип://путь:порт/названиеБД) |
| JDBC Driver class: | | com.mysql.jdbc.Driver (выбрать из дроп-листа) |
| Username: | | Фтвкун1981 |
| Password: | | andrey1981 (логин на сайте db4free.net) |
| Connection Properties: | | \*\*\*\*\*\*\*\*\*\* (пароль на сайте db4free.net) |

Окно «🎟JDBC Request»

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | JDBC Request (по умолчанию) | | |
| Comments: |  | | |
| Variable Name Bound to Pool: ------------------------------------------------------------------------------------------------- | | | |
| Variable Name of Pool declared in JDBC Connection Configuration: | | | var (ввести ТУ ЖЕ переменную!) |
| SQL Query: ------------------------------------------------------------------------------------------------- | | | |
| Query Type: | | Select statement (по умолчанию) | |
| Init SQL statements separated by new line  1 SELECT \* FROM cars  2  3 | | | |
|  | | | |
| Parameter values: | |  | |
| Parameter types: | |  | |
| Variable names: | |  | |
| Result variable names: | |  | |
| Query timeout (s): | |  | |
| Limit ResultSet: | |  | |
| Handle ResultSet: | | Store as String (по умолчанию) | |

Должен быть скачан и установлен JDBC драйвер (см. начало этого раздела «ЗАПРОС К БД»)

Слушатель «View Results in Tree»

Start «⯈»

В окне Листенера «View Results in Tree» есть 3 тега:

Sampler Result – Краткая инфа по Ответу

Request – заголовки и тело Запроса

Response Data – тело Ответа

Перейти к «Response Data»

Результат Запроса к БД (SELECT \* FROM cars) выводится в виде таблицы прямо в JMeter:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CarID | Make | Model | Year | ValidMOT | Price | Colour |
| 1 | Audi | S3 | 2012 | true | 10000 | Black |
| 2 | Nissan | Qashqai | 2010 | true | 8000 | Orange |
| 3 | Peugeot | 206 | 2005 | false | 400 | Black |
| 4 | Audi | A6 | 2010 | true | 7095 | Blue |
| 5 | Audi | Q3 | 2018 | true | 27095 | Blue |
| 6 | Volkswagen | Golf | 2013 | false | 13049 | Red |
| 7 | Volkswagen | Polo | 2009 | true | 7000 | White |
| 8 | Ford | Focus | 2010 | true | 3949 | White |
| 9 | BMW | Mini | 2012 | false | 6999 | Purple |
| 10 | BMW | 1 | 2017 | true | 17000 | White |

**ЗАПИСЬ В JMETER СЦЕНАРИЯ ДЕЙСТВИЙ НА САЙТЕ**

С помощью JMeter можно записать некий сценарий (н-р, покупка товара) и потом его проигрывать

Tread Group 🡪 Add 🡪 Config Element 🡪 HTTP Header Manager

Tread Group 🡪 Add 🡪 Logic Controller 🡪 Recording Controller

Tread Group 🡪 Add 🡪 Listener 🡪 View Results Tree

Test Plan 🡪 Add 🡪 Non-Test Elements 🡪 HTTP(S) Test Script Recorder

«Символ иконы "Векторная лаборатория" — стоковый вектор TestPlan»

«🟓Group3»

«🛠HTTP Header Manager»

«🗔Recording Controller»

«⯋◗View Results in Tree»

«🗔HTTP(S) Test Script Recorder»

Перейти к «HTTP(S) Test Script Recorder»: Тут записывается сценарий на выбранном сайте

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name: | | HTTP(S) Test Script Recorder (по умолчанию) | | | | | | | | | | | | | | | | | |
| Comments: | |  | | | | | | | | | | | | | | | | | |
| State: ------------------------------------------------------------------------------------------------------------------------------ | | | | | | | | | | | | | | | | | | | |
|  | | | | |  | | ⯈Start | | ⯃Stop | | | **⭮**Restart | |  | | | |  |  |
| Global Settings: ------------------------------------------------------------------------------------------------------------------- | | | | | | | | | | | | | | | | | | | |
| Port: | 8888 (должен быть одинаков с браузером!) | | | | | | | | |  | HTTP Domains: | | | | | Ab-soft.net | | | |
| Test Plan Creation | | | | Request Filtering | | | | | |  | | | | | | | | | |
| Test Plan Content: --------------------------------------------------------------------------------------------------------------- | | | | | | | | | | | | | | | | | | | |
| Target Controlling: | | | | | | | | | Use Recording Controller (по умолчанию) | | | | | | | | | | |
| Grouping: | | | | | | | | | Do not group samplers (по умолчанию) | | | | | | | | | | |
| 🗹 Capture HTTP Headers | | | | | | | | 🞎 Add Assertions | | | | | | | 🞎 Regex Matching | | | | |
| HTTP Sampler Settings: -------------------------------------------------------------------------------------------------------- | | | | | | | | | | | | | | | | | | | |
| Transaction name | | | | | | | | |  | | | | | | | | | | |
| Naming scheme | | | Prefix | | |  | | | #(counter, number, 000) - #(path) (#(name)) | | | | | | | | | | |
| Counter start value | | | | | | | | |  | | | |  | | | | Set counter | | |
| Create new transaction after request (ms): | | | | | | | | |  | | | | | | | | | | |
| Recording’s default encoding | | | | | | | | |  | | | | | | | | | | |
| 🞎 Retrieve All Embedded Resources | | | | | | | | 🞎 Redirect Automatically | | | | | | | 🗹 Follow Redirects | | | | |
| 🗹 Use Keep Alive | | | | | | | |  | | | | | | |  | | | | |
| Type: | | | | | | | | |  | | | | | | | | | | |
| GraphQL HTTP Sampler settings: ------------------------------------------------------------------------------------------ | | | | | | | | | | | | | | | | | | | |
| 🗹 Detect GraphQL Request | | | | | | | |  | | | | | | |  | | | | |

Запустить дополнительный браузер, (н-р, Mozilla Firefox) для настройки порта, одинакового с JMeter!

Зайти на сайт

Настроить в Firefox прокси-сервер:

Настройки

Проскроллить вниз

Секция «Параметры сети»

[Настроить…]

Окно «Параметры соединения»:

🞊 Ручная настройка прокси

HTTP прокси: localhost Порт: 8888 (должен быть одинаковый с JMeter!)

🗹 Также использовать этот прокси для HTTPS

[ОК]

Вернуться в Firefox на сайт

Обновить страницу

Выйдет сообщение: «Прокси-сервер отказывается принимать соединения»

Вернуться в JMeter

В окне «HTTP(S) Test Script Recorder» нажать «⯈Start»

(может выйти ошибка об отсутствии proxyserver.jks – это библиотека куда сохраняются ключи SSL, и если JMeter запущен из папки, кот расположена в Program Files, то Windows запрещает записывать что-либо в Program Files, и следовательно в подпапку с JMeter и proxyserver.jks. Для решения можно поменять права или переместить подпапку с JMeter на рабочий стол или другой диск).

На короткое время появится и исчезнет сертификат SSL/TLS (сертификат сохраниться в «JMeter/bin»: «ApacheJMeterTemporaryRootCA.crt»).

Появится маленькое окно «Recorder Transaction Control»:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ⯃Stop | HTTP Sampler Settings: ----------------------------------------------------------------------------------- | | | | |
|  | Transaction name | | |  | |
|  | Naming scheme | Prefix |  |  | |
|  | Counter start value | | |  | Set counter |
|  | Create new transaction after request (ms): | | |  | |

В Firefox обновить страницу, вместо сообщения о невозможности отобразить содержимое: «Предупреждение: Вероятная угроза безопасности».

Теперь надо снова настроить Firefox, что сертификату безопасности можно доверять

Настроить в Firefox сертификаты безопасности:

«≡»

Настройки

Поиск в настройках: Сертификаты

Окно «Сертификаты»

🗹 Запрашивать у OCSP-серверов подтверждение текущего статуса сертификатов

[Просмотр сертификатов…]

[Устройства защиты…]

Окно «Управления Сертификатами»

[Импортировать…]

Окно «Загрузка Сертификата»

Как и при открытии обычного файла открыть файл сертификата: …/ApacheJMeter/bin/

«ApacheJMeterTemporaryRootCA.crt»

🗹 Доверять при идентификации веб-сайтов

🗹 Доверять при идентификации пользователей электронной почты

[ОК]

[ОК]

Закрыть тег «Настройки»

В Firefox обновить страницу – сайт заработал.

Теперь маленькое окно «Recorder Transaction Control» записывает наши действия на сайте.

В Firefox на сайте перемещаемся по разделам, под-страницам, вкладкам.

В маленьком окне «Recorder Transaction Control» 🡪 «⯃Stop»

В JMeter в дереве, рядом с «Recording Controller» появился марке подраздела «>»

«Символ иконы "Векторная лаборатория" — стоковый вектор TestPlan»

«🟓Group3»

«🛠HTTP Header Manager»

«>🗔Recording Controller»

«🎟http://mysite.com/….»

«🎟http://mysite.com/….»

«🎟http://mysite.com/….»

«🎟http://mysite.com/….»

«⯋◗View Results in Tree»

«🗔HTTP(S) Test Script Recorder»

Открыть в дереве «Recording Controller» – записались все действия на сайте, в виде «🎟HTTP Request». Причём, кроме них так же все сопутствующие гугл-аналитик, ю-туб, фейсбук и т.д. – их придётся чистить (удалять) вручную (это один из минусов JMeter) – ресурс сайта видно в правой части окна в каждом из «HTTP Request».

Дальше можно нагружать «Tread Group» пользователями, и следить за тем как ведут себя странички нашего сайта.

Но можно использовать BlazeMeter из Google Chrome.

**BlazeMeter**

**ЗАПИСЬ В BLAZEMETER** **СЦЕНАРИЯ ДЕЙСТВИЙ НА САЙТЕ**

BlazeMeter – Расширение Google Chrome, которое записывает посещаемые страницы.

Скачать расширение.

Перейти на выбранный сайт в Chrome.

Запустить расширение в Chrome: Справа-вверху «🏶» 🡪 BlazeMeter

Авторизоваться в нём с помощью Гугл-аккаунта (чтобы была доступна запись в формате JMeter)

Справа вверху появится мини-окно рекордера BlazeMeter

Нажать «🞉»

Пойдёт запись.

В Chrome на сайте перемещаемся по разделам, под-страницам, вкладкам.

Нажать «■»

В Chrome: Справа-вверху, рядом с «🏶», начнёт мерцать «●» с «В» 🡪 нажать «●»

Снова появится мини-окно рекордера BlazeMeter

BlazeMeter мини-окно: «Save»

BlazeMeter мини-окно: Выбрать формат:

🗹 JMeter (JMX)

🞎 Selenium Only (YAML)

🞎 JMeter & Selenium combined (YAML)

BlazeMeter мини-окно: выбрать только тот сайт что нужен:

🗹 mysite.com

🞎 google-analytics.com

🞎 facebook.com

«Save»

JMX-Файл сохранится в загрузки.

Переместить его в папку JMeter/bin

В JMeter: File 🡪 Open 🡪 RECORD-12-22-21-10-23-33-PM.jmx

Загрузится новый Тест-План с названием записи: «RECORD-12-22-21-10-23-33-PM» с «Config Element»-ами, и «Thread Group» с вкладкой «Test»:

«Символ иконы "Векторная лаборатория" — стоковый вектор TestPlan»

«🛠HTTP Header Manager»

«🛠User Defined Variables»

«🛠HTTP Request Defaults»

……

«🟓Thread Group»

«🗔Test»

«🎟http://mysite.com/….»

«🎟http://mysite.com/….»

«🎟http://mysite.com/….»

(Чистить вручную уже ничего не надо, как с записью сценария из самого JMeter – все запросы только по сайту http://mysite.com)

Это уже готовый сценарий

Добавим Листенера «View Results Tree»

В «Thread Group» установим пользователей, частоту, итерации

Start «⯈»

Смотрим на Ответы

Можно добавить Ассёрты.

**Grafana**

**? Grafana** — это мультиплатформенное веб-приложение для аналитики и интерактивной визуализации с открытым исходным кодом. Grafana это пакет для визуализации данных мониторинга. В отличие от Graphite, который и данные хранит, и графики строит, Grafana фокусируется только на визуальной части и дашбордах.

**? Возможности**

Предоставляет диаграммы, графики и предупреждения для Интернета при подключении к поддерживаемым источникам данных. Конечные пользователи могут создавать сложные панели мониторинга с помощью интерактивных запросов. Grafana разделена на фронтенд и бэкенд, написанные на TypeScript и Go соответственно.

Grafana используется для визуализации метрик, собранных с узлов кластера и виртуальных машин.

С помощью информационных панелей (дашбордов) Grafana вы можете:

* настроить отображение метрик в виде графиков и диаграмм;
* отслеживать изменения метрик во времени;
* настроить отправку автоматических уведомлений в мессенджеры. Подробнее см. в статье Grafana. Настройка уведомлений.

**? Источники данных Grafana**

Grafana поддерживает из коробки различные источники данных:

Prometheus,

Graphite,

OpenTSDB,

InfluxDB,

Elasticsearch и другие.

**? Работа с Grafana**

Графана работает только через веб-браузер!

**УСТАНОВКА И ЗАПУСК**

* Установить на компьютер
* Зайти в браузер, в адресной строке набрать: http://localhost:3000
* Появится страница авторизации
* Ввести данные уже установленные по умолчанию:

Email or Username: admin

Password: admin

* Снова появится страница авторизации, с предложением изменить пароль:

Password: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Repeat Password: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Попадаем на Welcome page

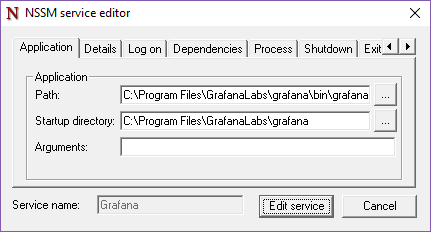
**НАСТРОЙКИ КОНФИГУРАЦИИ**

* Нужно определить собственный конфигурационный файл
* Для Windows версии сервис запускается NSSM (service manager for Windows)
* По умолчанию Графана зависит от конфигурационных файлов, находящихся в папке «Grafana\conf»
* Зайти в папку: C:\Program Files\GrafanaLabs\grafana\conf
* По умолчанию Графана использует содержимое файла defaults.ini
* Этот файл нужно перезаписать, чтобы использовать собственную конфигурацию. Если в дальнейшем с собственной конфигурацией возникнут проблемы, всегда можно вернуться к файлу по умолчанию.
* В Папке «conf»: скопировать файл «defaults.ini» и переименовать в «custom.ini»
* Перейти в папку: C:\Program Files\GrafanaLabs\svc-8.3.3.0
* Там будет файл «nssm.exe»
* В этой папке с файлом «nssm.exe»: Нажать и удерживать «Win+X» и из появившегося меню выбрать: «Командная строка (администратор)»
* Ввести команды (не забыть про кавычки!):

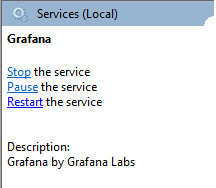
"C:\Program Files\GrafanaLabs\svc-8.3.3.0"

C:\Program Files\GrafanaLabs\svc-8.3.3.0>nssm.exe edit grafana

* Появится окно NMMS Service Editor



* В поле «Arguments» ввести «- - config conf» с названием созданного файла: --config conf\custom.ini
* «Edit service»
* NMMS окно с сообщением об успехе
* «ОК»
* В Windows: Перезапустить (Restart) сервис в поиске по Windows: «Service/Просмотр локальных служб»



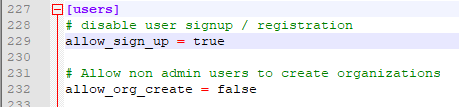
* В браузере убедиться, что Графана работает нормально – обновить страницу: http://localhost:3000 «F5»

**РАЗРЕШИТЬ/ЗАПРЕТИТЬ РЕГИСТРАЦИЮ НОВОГО ПОЛЬЗОВАТЕЛЯ**

По умолчанию опция регистрации нового пользователя запрещена

Чтобы разрешить регистрацию (н-р, для доступа к каким-либо дашбордам):

* Зайти в папку: C:\Program Files\GrafanaLabs\grafana\conf
* Открыть в текстовом редакторе созданный файл «custom.ini»
* Перейти к секции «[users]» (строка ≈220-320)
* В строке «allow\_sign\_up» изменить значение на «true»



* Чтобы убедиться, что всё работает норм – в Браузере перейти на страницу: http://localhost:3000/signup
* Зарегистрировать нового юзера

**НАСТРОЙКА ИСТОЧНИКА ДАННЫХ**

Для Графана нужно настроить один из источников данных, т.к. сама по себе Графана данных не содержит, а только отображает, визуализирует метрики. По умолчанию, один из встроенных источников данных – база данных InfluxDB. Но её надо установить.

Но для самой базы данных тоже нужны сами данные – метрики. Нужно установить контроллер данных. В InfluxDB как источник данных по умолчанию используется Telegraf, который в свою очередь тоже надо устанавливать с того же ресурса (InfluxDB). Но можно подключить и любой другой. Н-р, «JMeter».

То есть, надо настроить связку – стек: «Датчики – База данных – Источник» = «Grafana – InfluxDB – Jmeter»

Связки могут быть абсолютно разными: н-р, «Grafana – InfluxDB – Telegraf» или «Grafana – Prometheus – Jmeter»

**InfluxDB**

**? InfluxDB**

InfluxDB представляет собой программное обеспечение с открытым исходным кодом для хранения временных рядов. БД написана на языке Go и не требует внешних зависимостей. Основным назначением является хранение больших объемов данных с метками времени. Например, данные мониторинга, метрики приложений и данные датчиков IoT.

**? Работа с InfluxDB**

* Скачать «InfluxDB 2.x Open Source Time Series Database» с сайта https://portal.influxdata.com/downloads/
* Распаковать архив
* Копировать содержимое в С:\Program Files\InfluxData
* Запустить из командной строки «influxd.exe» - запуститься сервер «Influx»

C:\Program Files\GrafanaLabs\grafana\bin>grafana-server.exe

* Не закрывать, а просто свернуть командную строку – сервис должен работать!
* В Браузере перейти к порту 8086: http://localhost:8086
* Зарегистрироваться:

Username: MyName

Password: \*\*\*\*\*\*\*\*\*\*\*\*\*

Confirm Password: \*\*\*\*\*\*\*\*\*\*\*\*\*

Initial Organization Name: QAHome (название БД)

Initial Bucket Name: Bucket1 (название таблицы)

«Quick Start»

* Создать источник данных для БД «Bucket1»:
* Меню «Data» 🡪 тег «API Token» 🡪 кнопка справа «+ Generate API Token» 🡪Read/Write API Token
* Окно «Generate Read/Write API Token»:

Description: JMeterToken (ввести название-описание)

Read: Bucket1 (выбрать)

Write: Bucket1 (выбрать)

«Save»

* Тег «API Token»: Выбрать из списка JMeterToken
* Окно «JMeterToken»: JOFqJDii9ZsAXfKlqXbtlUozdhZFJkqtDWplvuL7s1s1iWKraFYZDIi5S6\_0HSc66wkJpn9qPT8TNDSyQKpTg== «Copy to Clipboard»

**JMeter**

Для настройки JMeter надо скачать файл Листенера, который будет отправлять данные бэкенда в БД.

* В Google или GitHub найти джава-файл по названию «JMeter-InfluxDB-Writer»
* Загрузить: «JMeter-InfluxDB-Writer-1.0.jar» и переместить в папку: .\apache-jmeter-5.4.2\lib\ext
* В JMeter создать Test-Plan, Thread Group, HTTP Request, и добавить «Backend Listener»
* В «Backend Listener» прописать настройки:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name: | Backend Listener (по умолчанию) | | | | |
| Comments: |  | | | | |
| Backend Listener implementation: | | | | org.apache.jmeter……InfluxdbBackendListenerClient (выбрать из списка) | 🞃 |
| Async Queue size | | 5000 (по умолчанию) | | | |
|  | | | | | |
| Parameters values: | | | | | |
| influxdbMetricSender | | | org.apache.jmeter.visualizers.backend.influxdb.HttpMetricsSender (по умолчанию) | | |
| influxdbUrl | | | http://localhost:8086/api/v2/write?org=QAHome&bucket=Bucket1 (🡨изменить) | | |
| application | | | Demo | | |
| measurement | | | Jmeter | | |
| summaryOnly | | | false | | |
| samplersRegex | | | .\* (по умолчанию) | | |
| percentiles | | | 99;95;90 (по умолчанию) | | |
| testTitle | | | Demo | | |
| eventTags | | |  | | |
| influxdbToken(🡨добавить! «Add») | | | JOFqJDii9Z….…………ZDIi5S6\_0HSc66wkJpn9qPT8TNDSyQKpTg== (🡨токен вставить из InfluxDB) | | |

* Можно так же добавить Ассёрты и Листенеры по необходимости
* Далее, нагрузить какой-либо сайт.
* Start «⯈»

**InfluxDB**

* Меню «Explore»
* Выбрать тип отображения данных: н-р, «Graph»
* Слева, над фильтрами, вкладка «Query1»: установить значения в секциях-фильтрах (выбрав значения из списков), секции добавлять «+»:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| FROM | Filter | Filter | Filter | Filter | Filter | + |
| *Search* | \_measurement🞃 | \_field🞃 | application🞃 | statut🞃 | transaction🞃 |
| Bucket1 | *Search* | *Search* | *Search* | *Search* | *Search* |
|  | 🞕 jmeter | 🞕 pct95.0 | 🞕 Demo | 🞕 ok | 🞕 all |
|  |  | 🞕 (anyone) |  |  | 🞕 HTTP Request |
|  |  | 🞕 (anyone) |  |  |  |

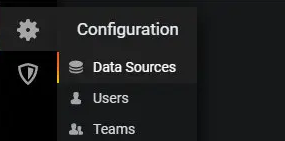
* Справа, над фильтрами, установить время: «Past 5m, 15m, 1h, 3h, 6h, …»
* Над фильтрами, справа: нажать «Submit»

По выбранным параметрам фильтров отобразятся графики запроса из JMeter.

**Grafana**

**ДОБАВЛЕНИЕ ИСТОЧНИКА ДАННЫХ**

* Иконка «Configuration» 🡪 «Data Sources»



* «Add data source» 🡪 «InfluxDB»🡪 «Select»
* Окно «Data Sources / InfluxDB»: задать параметры для подключения к InfluxDB:

Name: InfluxDB-Jmeter (любое) Default: ON (источник данных будет использоваться по умолчанию)

**Query Language**

InfluxQL | Flux (для версии InfluxDB 2.0+ обязательно выбрать «Flux»!)

**HTTP**

URL: http://localhost:8086

Access: Server (default)

…

…

**InfluxDB Details**

Organization: QAHome

Token: JOFqJDii9Z….…………ZDIi5S6\_0HSc66wkJpn9qPT8TNDSyQKpTg== (тот же, что и для JMeter)

Default Bucket: Bucket1

Min time interval: 10s (по умолчанию)

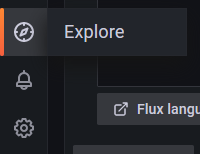
Max series: 1000 (по умолчанию)

« **Save & Test** »

Должно вернуться сообщение: « ✔ 1 buckets found» – значит, всё работает успешно.

**ПРОВЕРИТЬ СТЕК «GRAFANA – INFLUXDB»**

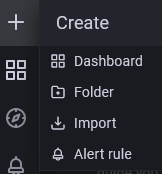
* Перейти в меню «Explore»



* Выбрать из списка источник данных (вверху слева): «InfluxDB-Jmeter»
* Нажать «Sample Query»
* Выбрать «Show buckets»
* Если всё настроено верно – придёт flux-ответ «buckets()» и внизу сформируется служебная таблица, в которой будет Bucket1

**СОЗДАНИЕ ДАШБОРДОВ**

* Дашборды – это панели-индикаторы различных метрик, подгружаемых из баз данных.
* Чтобы создать Дашборд – нажать «Create» 🡪 «Dashboard»



* Выбрать «Add a new panel»:

|  |  |
| --- | --- |
| Add a new panel  (1 панель) | Add a new row  (группа панелей) |
| Add the panel from the panel library (библ. пуста) |  |

* Появится пустая панель
* Справа, будет секция для изменения внешнего вида панели, там же ввести название панели:
* Title: «Jmeter - avg»
* Под Панелью будет окно запроса
* Нажать: «Sample Query»
* Выбрать сниппет фласк-запроса: «Simple Query»
* В сниппете исправить образцы подставив свои данные (пар-ры можно посмотреть в InfluxDB/Explore/Filters):

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | from(bucket: "Bucket1")  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)  |> filter(fn: (r) =>  r.\_measurement == "jmeter" and  r.\_field == "avg"  ) |

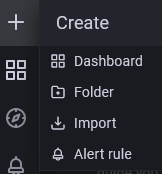
* Справа, над панелью, установить время: «Last 5m, 15m, 30m, 1h, 3h, 6h, …»

В панели отобразится соответствующий график.

**ИМПОРТ ДАШБОРДОВ**

Чтобы не настраивать вручную множество панелей, можно импортировать уже готовые наборы.

* Чтобы создать Дашборд – нажать «Create» 🡪 «Import»



* Выбрать вариант загрузки:

« **Upload JSON file** » – загрузить набор JSON-файлом (со своего компьютера)

Import via grafana.com « **Load** » – загрузить набор с сайта

Import via panel JSON « **Load** » – загрузить набор JSON-кодом прямо здесь-же, во фрейм

* Для загрузки с сайта надо вставить URL или ID набора
* Перейти на сайт http://grafana.com

Тег Grafana 🡪 Dashboards 🡪 Search Dashboards: Jmeter (or: InfluxDB Metrics) «Enter»

Выбрать из найденных

На странице определённого Дашборда посмотреть ID

(либо на странице, либо в конце адресной строки браузера):

InfluxDB Metrics – 317

Apache JMeter Dashboard using Core InfluxdbBackendListenerClient – 5496

…

* Вернуться в Grafana, к «Import» и ввести ID:

Import via grafana.com: 317 « **Load** » – загрузить набор «InfluxDB Metrics» с сайта

* Появится новая версия окна «Import»:

…

**Options**

Name

InfluxDB Metrics (по умолчанию, можно изменить)

Folder

General (по умолчанию, можно изменить)

…

Internal influxdb

InfluxDB-Jmeter (выбрать из списка)

« **Import** »

* Справа, над панелями, установить время: «Last 5m, 15m, 30m, 1h, 3h, 6h, …»

В панели отобразятся соответствующие графики.

**Telegraf**

Influx Telegraf – инструмент для сбора источников данных, для мониторинга данных о производительности компьютера. Стек «Telegraf–Influxdb–Grafana»: Telegraf выступает в качестве инструмента для сбора источников данных; Influxdb – для хранения данных; Grafana – отображает диаграммы для создания платформы мониторинга.

**УСТАНОВКА**

* На сайте influxdata.com перейти в раздел загрузки: https://portal.influxdata.com/downloads/
* Выбрать секцию Telegraf и платформу Windows
* Использовать появившуюся инструкцию для скачивания в PowerShell:

wget https://dl.influxdata.com/telegraf/releases/telegraf-1.21.1\_windows\_amd64.zip -UseBasicParsing -OutFile telegraf-1.21.1\_windows\_amd64.zip

Expand-Archive .\telegraf-1.21.1\_windows\_amd64.zip -DestinationPath 'C:\Program Files\InfluxData\telegraf'

ИЛИ

* В браузере: ввести в адресную строку часть команды:

<https://dl.influxdata.com/telegraf/releases/telegraf-1.21.1_windows_amd64.zip>

* Windows окно скачивания: выбрать открыть/сохранить
* Windows: из загрузок переместить архив и распаковать
* Папку «telegraf-1.21.1» лучше переместить на Desktop (т.к. в Program Files будет проблематично запустить Telegraf из-за безопасности Windows)
* В папке «telegraf-1.21.1» будут два файла: telegraf.conf и telegraf.exe – их надо настроить
* В файле telegraf.conf плагин для снятия основных системных метрик уже активирован. Телеграф мониторит следующие объекты:

Processor

Logical Disk

Physical Disk

Network Interface

System

Memory

Paging File

Но можно настроить и другие метрики.

**НАСТРОЙКА – I СПОСОБ**

* Windows: Открыть telegraf.conf в текстовом редакторе
* Деактивировать плагин InfluxDB v1, обозначив его как коммент: добавить «#» перед [[outputs.influxdb]].
* Активировать плагин InfluxDB v2, удалив обозначение коммент: удалить «#» перед [[outputs.influxdb\_v2]].
* Также в секции [[outputs.influxdb\_v2]] надо прописать URL, токен, организацию и таблицу (Бакет), также удалив обозначение комментария «#».
* Токен нужно сгенерировать в InfluxDB в «Data», тег «API Tokens».

[[outputs.influxdb\_v2]]

urls = ["http://localhost:8086"]

token = "iVer00JQHEEb1c-v-…….\_m7--227HNTKSA=="

organization = "QAHome"

bucket = "Bucket2"

**НАСТРОЙКА – II СПОСОБ**

* InfluxDB: Меню «Data» 🡪 тег «API Token» 🡪 кнопка справа «+ Generate API Token» 🡪 Read/Write API Token
* Окно «Generate Read/Write API Token»:

Description: TelegrafToken (ввести название-описание)

Read: Bucket2 (выбрать)

Write: Bucket2 (выбрать)

«Save»

* InfluxDB: Меню «Data» 🡪 тег «Telegraf» 🡪 кнопка справа «+ Create Configuration»

Bucket: Bucket2

Выбрать «System»: System | Docker | Kubernetes | NGINX | Redis

«Continue»

Telegraf Configuration Name: MyTelegraf

«Create and Verify»

(Окно с инструкцией по настройке)

«Finish»

* InfluxDB: Меню «Data» 🡪 тег «Telegraf» 🡪 выбрать из списка «MyTelegraf» 🡪 скопировать весь скрипт
* Windows: в текстовом редакторе изменить telegraf.conf, удалить всё, вставить из буфера скрипт
* InfluxDB: Меню «Data» 🡪 тег «API Token» 🡪 TelegrafToken 🡪 скопировать токен
* Windows: в текстовом редакторе в изменённый telegraf.conf, подставить токен вместо «$INFLUX\_TOKEN»:

~86 | token = "$INFLUX\_TOKEN" 🡪 ~86 | token = "iVer00JQHEEb1c-v-…….\_m7--227HNTKSA=="

* Windows: поиск 🡪 Просмотр Локальных Служб 🡪 «Службы»
* Windows окно «Службы»: если появилась строка «Telegraf Data Collector Server» 🡪 колонка тип запуска 🡪 «Автоматически» поменять на «Вручную»
* Windows: «Win+X» 🡪 Открыть командную строку (Администратор)
* CLI: cd C:\Users\Admin1\Desktop\telegraf-1.21.1
* CLI: C:\Users\Admin1\Desktop\telegraf-1.21.1>telegraf.exe --config telegraf.conf
* CLI: Запустится служба Telegraf по сбору системных данных, настроенная под Организацию и Бакет
* CLI: Не закрывать (а просто свернуть) окно командной строки, а то служба остановится!

**InfluxDB**

* Меню «Explore»
* Выбрать тип отображения данных: н-р, «Graph»
* Слева, над фильтрами, вкладка «Query1»: Установить значения в секциях-фильтрах (выбрав значения из списков), секции добавлять «+»:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| FROM | Filter | Filter | Filter | Filter | Filter | + |
| *Search* | \_measurement🞃 | \_field🞃 | cpu🞃 | host🞃 | …🞃 |
| Bucket2 | *Search* | *Search* | *Search* | *Search* | *Search* |
|  | 🞕 cpu | 🞕 (anyone) | 🞕 cpu-total | 🞕 Admin |  |
|  | 🞕 disk | 🞕 (anyone) | 🞕 cpu0 |  |  |
|  | 🞕 (anyone) | 🞕 usage\_user | 🞕 (anyone) |  |  |

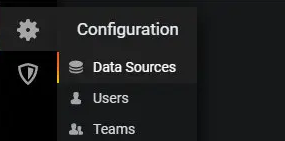
* Справа, над фильтрами, установить время: «Past 5m, 15m, 1h, 3h, 6h, …»
* Над фильтрами, справа нажать «Submit»

По выбранным параметрам фильтров отобразятся графики запроса из Telegraf.

**Grafana**

**ДОБАВЛЕНИЕ ИСТОЧНИКА ДАННЫХ**

* Иконка «Configuration» 🡪 «Data Sources»



* «Add data source» 🡪 «InfluxDB»🡪 «Select»
* Окно «Data Sources / InfluxDB»
* Задать параметры для подключения к InfluxDB:

Name: InfluxDB-Telegraf (любое) Default: ON (источник данных будет использоваться по умолчанию)

**Query Language**

InfluxQL | Flux (для версии InfluxDB 2.0+ обязательно выбрать «Flux»!)

**HTTP**

URL: http://localhost:8086

Access: Server (default)

…

…

**InfluxDB Details**

Organization: QAHome

Token: iVer00JQHEEb1c-v-…….\_m7--227HNTKSA== (тот же, что и для Telegraf)

Default Bucket: Bucket2

Min time interval: 10s (по умолчанию)

Max series: 1000 (по умолчанию)

« **Save & Test** »

* Должно вернуться сообщение: « ✔ 1 buckets found» – значит, всё работает успешно.
* Можно скопировать из InfluxDB токен для UserName, тогда Графана будет следить за всеми Buckets, и тогда должно вернуться сообщение: « ✔ … buckets found».

**ПРОВЕРИТЬ СТЕК «GRAFANA – INFLUXDB»**

* Перейти в меню «Explore»
* Выбрать из списка источник данных (вверху слева): «InfluxDB-Telegraf»
* «Sample Query»
* Выбрать «Show buckets»
* Если всё настроено верно – придёт flux-ответ «buckets()» и внизу сформируется служебная таблица, в которой будет Bucket2

**СОЗДАНИЕ ДАШБОРДОВ**

* «Create» 🡪 «Dashboard»
* Выбрать «Add a new panel»:
* Появится пустая панель
* Справа, будет секция для изменения внешнего вида панели, там же ввести название панели:
* Title: «Telegraf - cpu»
* Под Панелью будет окно запроса
* Нажать: «Sample Query»
* Выбрать сниппет фласк-запроса: «Simple Query»
* В сниппете исправить образцы по умолчанию на свои (пар-ры можно посмотреть в InfluxDB/Explore/Filters):

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | from(bucket: "Bucket2")  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)  |> filter(fn: (r) =>  r.\_measurement == "cpu" and  r.\_field == "usage\_system"  ) |

Или

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from(bucket: "Bucket2")  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)  |> filter(fn: (r) =>  r.\_measurement == "net" and  r.\_field == "packets\_recv" or  r.\_field == "packets\_sent"  ) |

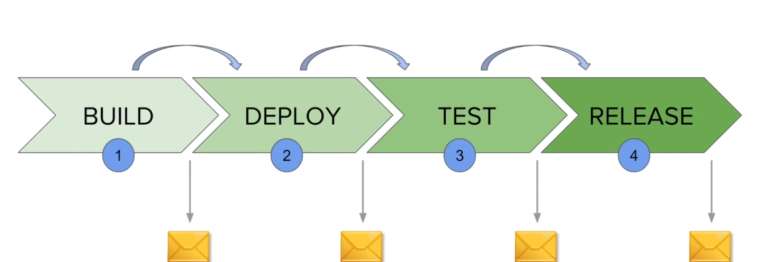
* Справа, над панелью, установить время: «Last 5m, 15m, 30m, 1h, 3h, 6h, …»

В панели отобразится соответствующий график.

**CI/CD**

**? CI/CD (Continuous Integration / Continuous Delivery OR Continuous Deployment)** — это комбинация непрерывной интеграции и непрерывного развёртывания программного обеспечения в процессе разработки.

CI/CD объединяет разработку, тестирование и развёртывание приложения.



**? CI (Continuous Integration)**

Основные этапы:

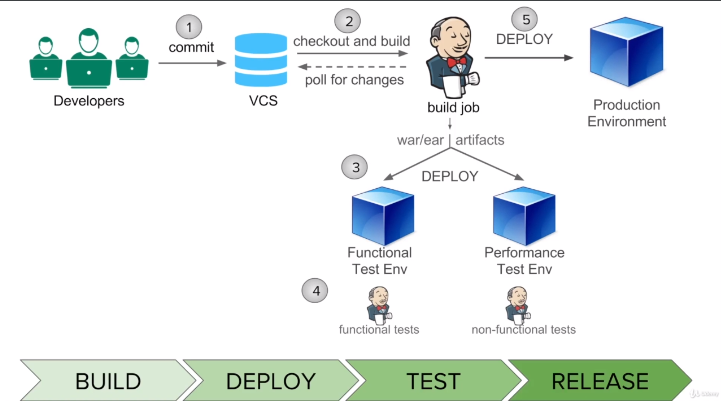
* Сборка
* Развёртывание
* Тестирование
* Релиз

Каждому из этапов соответствует свой ряд работ. Эти работы переходят друг в друга и связаны на пример цепочки: пока один ряд работ не закончится – другой не начнётся.

При завершении работ на каждом из этапов можно настроить нотификацию.

Реальный проект выглядит сложнее:

1. Программисты пишут код и коммитят его в Git или другую VCS.
2. Jenkins мониторит Git и когда появляются изменения – стягивает их.
3. Jenkins производит сборку и создаёт «.war» или «.ear» файлы-артефакты для последующего развёртывания в средах Функционального тестирования и тестирования Производительности.
4. Jenkins производит функциональное и нефункциональное тестирование.
5. Далее Jenkins производит развёртывания в производственной среде.



Тут шаги (3) и (5) оба относятся к этапу «Deploy», однако, в аспекте Автоматизированного «Deployment» имеется в виду все шаги в целом (1-2-3-4-5).

Таким образом, Автоматизированное Развёртывание – существенная часть CD (непрерывного развёртывания)

Автоматизированное Развёртывание – это процесс автоматизации процесса развёртывания в системе Непрерывного Развёртывания (CD), которая представляет из себя: Build🡪Deploy🡪Test🡪Release.

**? Pipeline**

Pipeline – рабочий процесс с группой событий или работ, которые связаны и интегрированы друг с другом в последовательность. Каждая работа в Pipeline имеет зависимость от одной и более других работ.

**Jenkins**

**? Jenkins** — программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения.

Кратко: Jenkins – это Java-приложение, необходимое для CI/CD.

**? Зачем нужен Jenkins**

Позволяет автоматизировать часть процесса разработки программного обеспечения, в котором не обязательно участие человека, обеспечивая функции непрерывной интеграции. Работает в сервлет-контейнере, например, Apache Tomcat.

**? Когда может быть запущена сборка.**

Сборка может быть запущена разными способами, например, по событию фиксации изменений в системе управления версиями, по расписанию, по запросу на определённый URL, после завершения другой сборки в очереди.

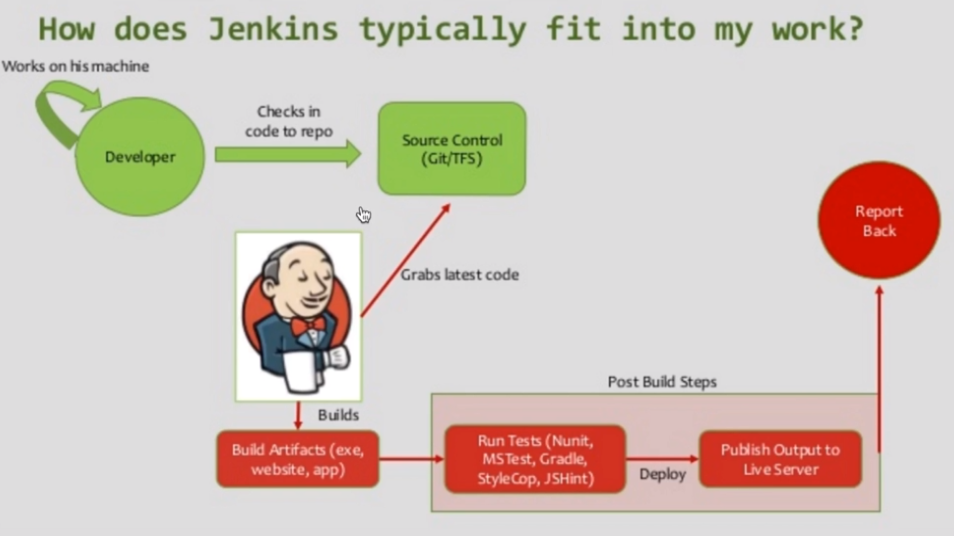
**? Преимущество использования Дженкинса в контейнере, перед использованием без него.**

В контейнере можно запустить кроме Дженкинса другие приложения, как будто на отдельном сервере.

**? Сервлет** – Сервлет является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ.

**? Принцип работы Jenkins**

* Программисты разрабатывают код
* Готовый билд кладут в репозиторий
* Дженкинс стягивает билды из репозитория
* Собирает коды в экзешник, сайт, прилагу
* Запускает и проводит тесты
* Публикует полученное на сервере
* Уведомляет

****

**? Работа с Jenkins**

**УСТАНОВКА**

Сайт «https://www.jenkins.io/» 🡪 «Download» 🡪 скачать стабильный релиз «Stable (LTS)» 🡪

«Generic Java Package (.war)»

Переместить файл jenkins.war, н-р: .\Desktop\Tools\jenkins-2.319.1

Запустить CLI

cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1

C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1>java -jar jenkins.war

Начнётся процесс запуска Дженкинса

В командной строке скопировать сгенерированный пароль (также пароль можно будет найти в папке «jenkins\secrets\initialAdminPassword»)

0bb2754dbafc460ca8cfc0db4ff40f7a

This may also be found at: C:\Users\Admin1\.jenkins\secrets\initialAdminPassword

Windows: Окно «Оповещение системы безопасности»: «Разрешить доступ»

Открыть браузер 🡪 перейти к http://localhost:8080

Окно «Getting Started»: Unlock Jenkins – Password: 0bb2754dbafc460ca8cfc0db4ff40f7a (скопированный)

«Выбрать плагины для установки»: Install suggested plugins | Select plugins to install

Выбрать плагины, или установить предложенные: All | None | Suggested 🡪 «Install»

Начнётся процесс установки

Окно «Create First Admin User»: создать нового пользователя ИЛИ нажать «Skip and continue as admin»

Окно «Instance Configuration»: Jenkins URL: <http://localhost:8080> 🡪 «Save and Finish»

Окно «Jenkins is ready!»: «Start using Jenkins»

Windows: в папке C:\Users\Admin1\.jenkins – будут все системные файлы и плагины

**ИЗМЕНЕНИЕ ДОМАШНЕЙ ДИРЕКТОРИИ (.JENKINS)**

Домашняя директория «.jenkins» устанавливается в директорию профайла юзера «.\Users\Admin1»

Для работы с Дженкинсом нужно будет много места, поэтому лучше эту системную директорию переместить

Запустить Дженкинс: Запустить CLI

cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1

C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1>java -jar jenkins.war

C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1>java -jar jenkins.war --httpPort=9190 \*будет работать на порту 9190

Открыть браузер 🡪 перейти к http://localhost:8080 (или localhost:9190)🡪 откроется Дженкинс

Панель слева: «Manage Jenkins» 🡪 выбрать «System Configuration»

Вверху будет указана домашняя директория и путь к ней.

Windows: в папке C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1 создать папку, н-р: JenkinsHome

Скопировать всё из C:\Users\Admin1\.jenkins 🡪 в C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome

Поменять переменную среды в Windows: My Computer (right click) 🡪 Properties 🡪 Additional System Parameters (left-hand) 🡪 «Environment Variables»

Windows окно «Environment Variables», секция «System Variables»: «Create»

Variable Name: JENKINS\_HOME

Variable Value: C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome

«ОК»

«ОК»

«ОК»

Перезапустить Дженкинс: в браузере в адресной строке: http://localhost:8080/restsrt ИЛИ закрыть окно командной строки со службой Дженкинс и запустить его опять: java -jar jenkins.war

Перезагрузить страницу в браузере

Login: admin

Password: 0bb2754dbafc460ca8cfc0db4ff40f7a

Проверить, что домашняя директория изменилась: «Manage Jenkins» 🡪 выбрать «System Configuration» 🡪

Home directory: C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome

**ИЗМЕНЕНИЕ ПРАВ**

Панель слева: «Manage Jenkins» 🡪 выбрать «Security Configuration»

Секция «Authorization» 🡪 🞊 Anyone can do anything 🡪 «Save»

**ИСПОЛЬЗОВАНИЕ CLI**

cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1 – перейти в папку с файлом jenkins.war

java -jar jenkins.war – запуск Дженкинс через порт 8080 (по умолчанию)

java -jar jenkins.war --httpPort=9000 – запуск Дженкинс через любой порт, н-р, 9000

Открыть браузер 🡪 перейти к http://localhost:8080 🡪 откроется Дженкинс

Панель слева: «Manage Jenkins» 🡪 выбрать «Configure Global Security» 🡪 🗹 Enable Security

В адресной строке браузера: http://localhost:8080/cli/

Страница «Jenkins CLI»: гиперссылка jenkins-cli.jar

Скачается файл jenkins-cli.jar

Создать папку и переместить файл в неё: C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsCLI

Открыть командную строку в папке C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsCLI

java -jar jenkins-cli.jar -auth admin: 0bb2754dbafc460ca8cfc0db4ff40f7a -s http:// localhost:8080/ help

Если всё ОК – в консоли выведется список доступных команд Jenkins CLI

В адресной строке браузера: http://localhost:8080/cli/

Выбрать из списка команду, н-р «restart» 🡪 появится образец CLI-команды

Скопировать и вставить в CLI:

java -jar jenkins-cli.jar -s http://localhost:8080/ -webSocket restart

**СОЗДАНИЕ НОВОГО ПОЛЬЗОВАТЕЛЯ**

«Manage Jenkins» 🡪 «Manage Users» 🡪 Окно «Manage Users» с одним юзером «admin»

Окно «Manage Users» 🡪 (справа) «Create User»

Name: User1

Password: 123456

Confirm Password: 123456

Full Name:

E-mail: user1@gmail.com

«Create User»

Окно «Manage Users»: таблица с 2 юзерами: admin и User1

**КОНФИГУРАЦИЯ НОВОГО ПОЛЬЗОВАТЕЛЯ**

Главная страница: Справа вверху выбрать User«🞃» 🡪 «Configure»

Окно «Configure»: тут можно изменить имя пользователя и пароль, сгенерировать API-токен, установить ключ.

**УПРАВЛЕНИЕ РОЛЯМИ ПОЛЬЗОВАТЕЛЕЙ**

В Дженкинсе должны быть распределены роли по правам использования функционала.

Предварительно нужно скачать плагин «Roles Strategy»

Google: jenkins roles strategy plugin🡪 https://plugins.jenkins.io/role-strategy/🡪 Releases🡪 3.2.0(Latest)🡪 Download

Файл role-strategy.hpi переместить в C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\plugins Перезапустить Дженкинс

ИЛИ

«Manage Jenkins» 🡪 «Manage Plugins» 🡪 тег «Advanced» 🡪 секция «Load Plugin» 🡪 «open» 🡪 role-strategy.hpi

Перезапустить Дженкинс

ИЛИ

«Manage Jenkins» 🡪 «Manage Plugins» 🡪 тег «Available» 🡪 search: role 🡪 🗹 Role-based Authorization Strategy

«Download now and install after restart»

«Manage Jenkins» 🡪 «Configure Global Security»

🗹 Enable Security

секция «Authorization»:

🞊 Role-Based Strategy 🡪 «Save»

«Manage Jenkins» 🡪 «Manage and Assign Roles» 🡪 «Manage Roles»

Окно «Manage Roles»:

**Global roles** (Роль на уровне Jenkins)

admin: … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹

Role to add: employee

«Add»

admin: … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹

employee: … 🞎, … 🞎, … 🞎, … 🗹, … 🞎, … 🞎, … 🗹 (предоставить те или иные права)

«Apply»

**Item roles** (Роль на уровне Item)

Role to add: developer

Pattern: Dev.\* (юзер будет иметь доступ ко всем проектам, которые начинаются на «Dev…»)

«Add»

developer “Dev.\*”: … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹 (все права проектов «Dev…»)

«Apply»

Role to add: tester

Pattern: Test.\* (юзер будет иметь доступ ко всем проектам, которые начинаются на «Test…»)

developer “Dev.\*”: … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹 (все права проектов «Dev…»)

tester “Test.\*”: … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹, … 🗹 (все права проектов «Test…»)

«Apply»

**Node roles** (Роль на ур-не Нодов) (None)

«Save»

«Manage Jenkins» 🡪 «Manage and Assign Roles» 🡪 «Assign Roles»

Окно «Assign Roles» (Global roles): Снять права у Анонимуса, добавить пользователей «admin», «user1», «user2» и предоставить им соответствующие права (администратор/работник)

Окно «Assign Roles» (Item roles): Снять права у Анонимуса, добавить пользователей «user1», «user2» и предоставить им соответствующие права (разработчик/тестировщик)

**Global roles**

|  |  |  |
| --- | --- | --- |
| User/group | admin | employee |
| Anonymous | 🗹 | 🗹 |

↓

User/group to add: admin «Add»

User/group to add: user1 «Add»

User/group to add: user2 «Add»

↓

|  |  |  |
| --- | --- | --- |
| User/group | admin | employee |
| Anonymous | 🞎 | 🞎 |
| admin | 🗹 | 🞎 |
| user1 | 🞎 | 🗹 |
| user2 | 🞎 | 🗹 |

**Item roles**

|  |  |  |
| --- | --- | --- |
| User/group | developer | tester |
| Anonymous | 🗹 | 🗹 |

↓

User/group to add: user1 «Add»

User/group to add: user2 «Add»

↓

|  |  |  |
| --- | --- | --- |
| User/group | admin | employee |
| Anonymous | 🞎 | 🞎 |
| admin | 🗹 | 🞎 |
| user1 | 🞎 | 🗹 |
| user2 | 🞎 | 🗹 |

**Node roles**

|  |
| --- |
| User/group |
| Anonymous |

«Apply» «Save»

Теперь User1 и User2, когда залогинятся, будут видеть только проекты, начинающиеся на «Dev…» и «Test…», соответственно, и не смогут менять настройки.

**БАЗОВАЯ КОНФИГУРАЦИЯ**

«Manage Jenkins» 🡪 «Configure System» (общее описание, конкретный пример – ниже)

**Home Directory** – посмотреть путь к системной папке

**System Message** – краткое сообщение, которое будет выводится вверху окна с проектами,   
для нотификации других пользователей Дженкинса, (можно применять html-теги для выделения этого сообщения: <h1>message</h1>, также можно вставить картинку или гиперссылку; НО надо разрешить Дженкинсу использовать разметку: «Manage Jenkins» 🡪 «Configure Global Security» 🡪 секция «Markup Formatter» 🡪 изменить с «Plain text» на «Safe HTML»).

**# of executors** – максимальное число работ по сборке, которое может одновременно вести Дженкинс

**Labels** – метки, которые присваиваются каждому Ноду (среде сборки), добавленной в «Manage Jenkins» 🡪 «Manage Nodes» (Кроме нашего Нода «Мастер», можно создать отдельный Нод, со своим числом работ по сборке, и проставить в нём Label, а потом по имени Label, ссылаться на этот Нод, с его настройками и числом работ)

**Quiet Period** – время задержки перед следующей сборкой, после срабатывания триггера (в секундах) (нужно > 0, т.к. бывает коммит не происходит мгновенно, а Дженкинс уже начнёт работать с недокомитченными файлами)

**SCM checkout retry count** – количество попыток обращения к репозиторию для выгрузки проектов

**Restrict project naming** – стратегия наименований (ограничивает наименования проектов):

⭘ Default

🞊 Pattern: Test.\* – названия проектов могут начинаться только на «Test…..»

⭘ Role-Based Strategy

**Global Properties** – глобальные настройки

🞎 Disable deferred wipeout on this node

🞎 Environment Variable – глобальные переменные «ключ – значение»

Name: key1

Value: value1

(можно обращаться к значению «value1» переменной «key1» из любого места) (синтаксис: $(key1) ИЛИ $key1)

🞎 Tool Locations – месторасположение разных тулзов, при сборке

**Jenkins URL** – просит в адресе http://localhost:8080 вместо «localhost» подставить IP-адрес:

http://127.0.0.1:8080, но можно и не делать

**System Admin Email Address** – мейл администратора

…

**БАЗОВАЯ КОНФИГУРАЦИЯ – ПРИМЕР**

«Manage Jenkins» 🡪 «Configure System»

**Home Directory:** C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome

**System Message:** <strong>This an example system message</strong>

**# of executors:** 5 – одновременно могут работать max 5 сборок

**Labels:** (None)

**Quiet Period:** 10 – задержка между проектами = 10с

**SCM checkout retry count:** 5 – кол-во попыток обращения к репозиторию = 5

**Restrict project naming:** 🞊 Pattern: Test.\* – названия проектов могут начинаться только на «Test»

**Global Properties:** (None)

**Jenkins URL:** http://localhost:8080 (ignore)

**System Admin Email Address:** mymail@gmail.com

**ОБЗОР – БОКОВЫЕ ПАНЕЛИ**

|  |  |  |
| --- | --- | --- |
| Боковая панель Дашборда:   * New Item * People * Build History * Manage Jenkins * My Views | Боковая панель Проекта:   * Back to Dashboards * Status * Changes * Workspace * Build Now * Delete * Configure   Builds History  #20 01-01-2022 01:21  #19 01-01-2022 01:10 | Боковая панель Сборки:   * Back to Project * Status * Changes * Console Output * Edit Build Information * Delete Build * Previous Build * Next Build |

**ОБЗОР – ДАШБОРД**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **All** | |  |  |  |  |  |
| **S** | **W** | **Name** | **Last Success** | **Last Failure** | **Last Duration** |  |
| ⮿ | 🌧 | **Test1** 🞃 | 1 min 56 sec - #10 | 4 min 11 sec - #7 | 0.62 sec | ▶🕑 |
| ✓ | 🌣 | **Test2** 🞃 | 2 min 07 sec - #9 | 3 min 12 sec - #8 | 0.71 sec | ▶🕑 |

*\* S – Status – статус последней сборки: Success/Failure*

*\* W – Weather – агрегированное состояние последних 5 сборок (провалились 5/5 – дождь , 4/5 – морось, …, 0/5 – ясно)*

*\* Name – Названия проектов, при наведении на название рядом появляется «🞃»– дроплист с действиями из Панели Проекта*

*\* # – Порядковый номер сборки*

*\* Last Success – Время и номер последней успешной сборки*

*\* Last Failure – Время и номер последней неуспешной сборки*

*\* Last Duration – Время продолжительности последней сборки*

*\* 🕑 – Schedule – Расписание*

**РАБОТА – СОЗДАНИЕ НОВОГО ПРОЕКТА**

«New Item» (общее описание, конкретный пример – ниже)

Ввести имя Item (начинающееся на «Test…», как было установлено в паттерне базовой конфигурации): Test1

Выбрать «Freestyle Project»

«ОК»

Окно «Test1»:

**General:**

Description: <em>This is test project #1</em> (описание проекта)

**Source Code Management:** (использование систем контроля версий)

🞊 None

⭘ Git

**Build Triggers:**

🞎 Trigger builds remotely (e.g., from scripts)

🞎 …

🞎 Build periodically – периодичность запуска сборки

Schedule: (ввести период по шаблону, чтобы посмотреть шаблон – нажать «(?)»)

🞎 Poll SCM – периодичность запроса в SCM (Git) об изменениях

Schedule: (ввести период по шаблону, чтобы посмотреть шаблон – нажать «(?)»)

**Build:** (сами сборки – без них Дженкинс будет работать в холостую со статусом «ОК»)

«Add build step»

⭘ Invoke Gradle Script

⭘ …

🞊 Execute Windows batch command – запускает командный сценарий Windows для сборки

Command: (ввести команду Windows: dir / date / … для начала сборки)

⭘ Execute Shell – выполняет запуск сценария оболочки для сборки проекта

Command: (ввести команду Shell для начала сборки)

**Post Build Actions:** (действия, после окончания сборки: уведомление / отчёт / начало новой сборки / …)

«Apply» «Save»

«Back to Dashboard» – вернуться на главную страницу

Произвести сборку: на Дашборде нажать Test1«🞃» 🡪 «Build now» ИЛИ зайти в «Test1» 🡪 «Build now» (справа)

**РАБОТА – СОЗДАНИЕ НОВОГО ПРОЕКТА – ПРИМЕР**

«New Item»

Ввести имя Item (начинающееся на «Test…», как было установлено в паттерне базовой конфигурации): Test1

Выбрать «Freestyle Project»

«ОК»

Окно «Test1»:

**General:**

Description: <em>This is test project #1</em>

**Source Code Management:**

🞊 None

**Build Triggers**

(None)

**Build:**

«Add build step»

🞊 Execute Windows batch command

Command: chcp 65001 (\*) (в CLI задаётся кодировка UTF-8)

winver (из CLI вызывается окно с версией Windows)

\* обязательно вводить перед командой «chcp 65001», иначе возникнет конфликт кодировки латиница-кириллица / Jenkins-Java-WinCLI и тесты будут FAILURE, а в выводе ответа консоли в Jenkins – «кракозяблы»

**Post Build Actions**

(None)

«Apply» «Save»

«Back to Dashboard» – вернуться на главную страницу

Произвести сборку: на Дашборде нажать Test1«🞃» 🡪 «Build now» **ИЛИ** зайти в «Test1» 🡪 «Build now» (справа)

На панели выведется результат:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **All** | |  |  |  |  |  |
| **S** | **W** | **Name** | **Last Success** | **Last Failure** | **Last Duration** | **Schedule** |
| ✓ | 🌣 | **Test1** 🞃 | 12 sec - #20 | 36 min - #15 | 5.2 sec | ▶🕑 |

Проект «Test1» можно корректировать – настраивать, исправляя ошибки: зайти в «Test1» 🡪 «Config» (справа)

Вывод в консоль можно посмотреть: «Test1» 🡪 (слева посредине) Builds History «#20» 🡪 «Console Outputs»

✓ **Console Outputs**

|  |
| --- |
| Started by user unknown or anonymous  Running as SYSTEM  Building in workspace C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\Test1  [Test1] $ cmd /c call C:\Users\Admin1\AppData\Local\Temp\jenkins809434390690511893.bat  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\Test1>chcp 65001  Active code page: 65001  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\Test1>winver  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\Test1>exit 0  Finished: SUCCESS |

**РАБОТА – ТРИГГЕР ПРОЕКТА ИЗВНЕ**

«Test1» 🡪 «Configure»

…

**Build Triggers:**

🗹 Trigger builds remotely (e.g., from scripts)

Authentication Token

|  |
| --- |
| 1234 (создать любой токен) |

Use the following URL to trigger build remotely: JENKINS\_URL/job/Test1/build?token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME

Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

Под строкой токена: Скопировать образец URL «JENKINS\_URL/job/Test1/build?token=TOKEN\_NAME»

«Apply» «Save»

Перейти в другой браузер, на другую машину.

В другом браузере: вставить и изменить URL «http://127.0.0.1:8080/job/Test1/build?token=1234» 🡪 «Enter»

В своём браузере, в Дженкинсе произойдёт новая сборка (н-р, #21)

**РАБОТА – СВЯЗКА НЕСКОЛЬКИХ ПРОЕКТОВ**

Создать +2 проекта: «Test2» и «Test3»

«New Item» 🡪 Name: Test2 🡪 «Freestyle Project» 🡪 «ОК»

Окно «Test2» 🡪

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: chcp 65001 (в CLI задаётся кодировка UTF-8)

dir (CLI показывает содержание папки)

**Build Triggers:** 🗹 Build after other projects are built (Запустить проект только после другого)

Projects to watch: Test1 (после какого именно проекта – Test1)

🞊Trigger only if build is stable (запустить проект Test2, только если Test1 прошёл ОК)

**Post Build Actions:** «Add Step» 🡪 Build another project (После сборки Test2 запустить др. проект)

Build projects: Test3 (какой именно проект – Test3)

🞊Trigger only if build is stable (запустить проект Test3, только если Test2 прошёл ОК)

«Apply» «Save»

«New Item» 🡪 Name: Test3 🡪 «Freestyle Project» 🡪 «ОК»

Окно «Test3» 🡪

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: chcp 65001 (в CLI задаётся кодировка UTF-8)

systeminfo (CLI показывает информацию о системе)

**Build Triggers:** 🗹 Build after other projects are built (Запустить проект только после другого)

Projects to watch: Test2 (после какого именно проекта – Test2)

🞊Trigger only if build is stable (запустить проект Test3, только если Test2 прошёл ОК)

**Post Build Actions:** (None) (После сборки Test3 ничего не запускать)

«Apply» «Save»

Запустить сборку «Test1»:

«🞃» 🡪 «Build now» 🡪 если Test1 пройдёт ОК – запустится Test2 🡪 если Test2 пройдёт ОК – запустится Test3

Получилась цепочка: «Test1 – Test2 – Test3»

«Build Queue» (информационная секция слева посредине): По мере окончания сборки проекта будет появляться следующий проект в очереди (проекты будут запускаться с задержкой 10с, установленной в «Quiet Period» в Базовой Конфигурации Дженкинс).

**РАБОТА – ТРИГГЕР JENKINS НА КОММИТ В GIT**

**Windows**

Создать файл с кодом (н-р, на Java): Hello.java

Написать программу: н-р, вывод в консоли «Hello World» 10 раз:

|  |
| --- |
| public class Hello { /\*создание класса\*/  public static void main(String[] args) { /\*создание метода\*/  for(int i=1;i<=10;i++) { /\*создание цикла: из 1, 2, …, 10\*/  System.out.println(“Hello World...”+i); /\*вывод на экран «Hello World» 10 раз\*/  }  }  } |

Сохранить этот файл в новой папке, н-р: Jenkins-Git-Java

В этой папке открыть Командную Строку

Скомпилировать файл «Hello.java»: javac Hello.java

В папке появится файл «Hello.class»

(Командная строка должна знать путь к файлу «javac.exe» для компиляции – задать путь, как системную переменную: Control Panel 🡪 System 🡪 Additional System Parameters 🡪 tag «Additional» 🡪 «Env variables» 🡪 section «System Variables» 🡪«Create» 🡪 Variable Name: PATH | Variable Value: .;C:\Program Files\Java\jdk1.8.0\_301\bin («.;»+Путь к папке с файлом «javac.exe») 🡪 «ОК»)

Запустить файл «Hello.java» в консоли: C:\Users\Admin1\Desktop\Jenkins-Git-Java>java Hello

Hello World...1

Hello World...2

………………………

Hello World...10

**Jenkins**

(Удалить паттерн названия в базовой конфигурации, чтобы называть проекты как угодно, а не только «Test…»:  
«Manage Jenkins» 🡪 выбрать «System Configuration» 🡪 🗹 Restrict project naming: ⭘Pattern 🞊Default 🡪 «Save»)

«New Item» 🡪 Name: HelloWorld 🡪 «Freestyle Project» 🡪 «ОК»

Окно «HelloWorld»:

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: chcp 65001 (в CLI задаётся кодировка UTF-8)

cd C:\Users\Admin1\Desktop\...\Jenkins-Git-Java (перейти в папку с Hello.java)

javac Hello.java (скомпилировать файл Hello.java)

java Hello (запустить файл Hello.java)

«Apply» «Save»

Запустить сборку «HelloWorld»

✓ **Console Output**

|  |
| --- |
| Started by user admin  Running as SYSTEM  Building in workspace C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld  [HelloWorld] $ cmd /c call C:\Users\Admin1\AppData\Local\Temp\jenkins6648657467756164965.bat  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld>chcp 65001  Active code page: 65001  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld>cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>javac Hello.java  'javac' is not recognized as an internal or external command,  operable program or batch file.  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>java Hello  Hello World...1  Hello World...2  Hello World...3  Hello World...4  Hello World...5  Hello World...6  Hello World...7  Hello World...8  Hello World...9  Hello World...10  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>exit 0  Finished: SUCCESS |

**GitHub**

Создать новый удалённый репозиторий «Jenkins-Git-Java»

**GitBash**

В папке «Jenkins-Git-Java» (с файлами «Hello.java» и «Hello.class») запустить GitBash

git init – создать локальный репозиторий

git status

git add .

git commit -m “added HelloWorld Project for Jenkins”

git remote add origin https://github.com/UserName/Jenkins-Git-Java.git (создать по локальному удалённый)

git branch -M main (переименование ветки «master» в «main»)

git push -u origin main (запустить ветку «main» с коммитом в удалённый репозиторий)

**Jenkins**

Проверить наличие Git-плагина: «Manage Jenkins» 🡪 «Manage Plugins» 🡪 Search: Git plugin

«Dashboards» 🡪 «HelloWorld🞃» 🡪 «Configure»

**Source Code Management:**

⭘ None

🞊 Git

Repository URL:

|  |
| --- |
| https://github.com/UserName/Jenkins-Git-Java.git (скопировать в GitHub) |

Branch Specifier (blank for 'any')

|  |
| --- |
| \*/main (изменить ветку «master» (по умолчанию) на «main») |

**Build Triggers:**

🗹 Poll SCM

Schedule:

|  |
| --- |
| \* \* \* \* \* (MINUTE HOUR DOM MONTH DOW) (будет запрашивать репо каждую минуту) |

«Apply» «Save»

**GitBash**

Сделать изменения в репозитории «Jenkins-Git-Java»

git echo “This is file for make changes in the Git repo” >> Changes.txt

git add .

git commit -m “add file for make changes in the Git repo”

git push

**Jenkins**

Дженкинс мониторит изменения в репозитории раз в минуту   
Как только произошли изменения (появился файл «Changes.txt») – на Дашборде начнёт мигать «▶🕑» –запустится триггер и пройдёт сборка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **All** | |  |  |  |  |  |
| **S** | **W** | **Name** | **Last Success** | **Last Failure** | **Last Duration** | **Schedule** |
| ✓ | 🌣 | **HelloWorld** 🞃 | 25 min - #5 | N/A | 1 min 9 sec | ▶🕑 |

Результаты обращения к Git и срабатывание программы можно посмотреть в консоли: «#5🞃» 🡪 «Console»

✓**Console Output**

|  |
| --- |
| Запущен изменением в SCM  Running as SYSTEM  Building in workspace C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld  The recommended git tool is: NONE  No credentials specified  > git.exe rev-parse --resolve-git-dir C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld\.git # timeout=10  Fetching changes from the remote Git repository  > git.exe config remote.origin.url https://github.com/AndreyGlabay/Jenkins-Git-Java.git # timeout=10  Fetching upstream changes from https://github.com/AndreyGlabay/Jenkins-Git-Java.git  > git.exe --version # timeout=10  > git --version # 'git version 2.33.0.windows.2'  > git.exe fetch --tags --force --progress -- https://github.com/AndreyGlabay/Jenkins-Git-Java.git +refs/heads/\*:refs/remotes/origin/\* # timeout=10  > git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10  Checking out Revision 008128a0fa2a0334c8acf3bea12e819dcfa97f6e (refs/remotes/origin/main)  > git.exe config core.sparsecheckout # timeout=10  > git.exe checkout -f 008128a0fa2a0334c8acf3bea12e819dcfa97f6e # timeout=10  Commit message: "new file for changes in Git repo"  > git.exe rev-list --no-walk 683959584b18a4497680155f989ba663b6da2c4f # timeout=10  [HelloWorld] $ cmd /c call C:\Users\Admin1\AppData\Local\Temp\jenkins356615379134911759.bat  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld>chcp 65001  Active code page: 65001  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\HelloWorld>cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>javac Hello.java  'javac' is not recognized as an internal or external command,  operable program or batch file.  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>java Hello  Hello World...1  Hello World...2  Hello World...3  Hello World...4  Hello World...5  Hello World...6  Hello World...7  Hello World...8  Hello World...9  Hello World...10  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>exit 0  Finished: SUCCESS |

**РАБОТА – СОЗДАНИЕ PIPELINE**

Установить Delivery Pipeline-plugin: «Manage Jenkins» 🡪 «Manage Plugins» 🡪 Available tag 🡪 Search: Delivery 🡪

🡪 🗹 Delivery Pipeline 🡪 «Download now and install after restart»

**Install/Update Plugins:** Downloaded Successfully. Will be activated during the next boot

Перезапустить Jenkins: закрыть-открыть CL 🡪 java -jar Jenkins.war --httpPort=9394

Создать 3 Проекта:

«New Item» 🡪 Name: SampleBuildJob 🡪 «Freestyle Project» 🡪 «ОК»

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: dir

Echo “Build Job done successfully!”

«Apply» «Save»

«New Item» 🡪 Name: SampleDeployJob 🡪 «Freestyle Project» 🡪 «ОК»

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: dir

Echo “Deploy Job done successfully!”

«Apply» «Save»

«New Item» 🡪 Name: SampleTestJob 🡪 «Freestyle Project» 🡪 «ОК»

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: dir

Echo “Test Job done successfully!”

«Apply» «Save»

Связать проекты в цепочку:

Окно «SampleDeployJob» 🡪

**Build Triggers:** 🗹 Build after other projects are built

Projects to watch: SampleBuildJob

🞊Trigger only if build is stable

«Apply» «Save»

Окно «SampleTestJob» 🡪

**Build Triggers:** 🗹 Build after other projects are built

Projects to watch: SampleDeployJob

🞊Trigger only if build is stable

«Apply» «Save»

Проверить связку: запустить «SampleBuildJob» запустится по цепочке «SampleDeployJob» и «SampleTestJob»

Настроить отображение Pipeline: «Dashboard» 🡪 tag «+» 🡪

View name: DeliveryPipelineTest

🞊Delivery Pipeline View

«OK»

Окно «DeliveryPipelineTest»:

**Pipelines**

Components 🡪 «Add»

Name: BuildJob (любое)

Initial Job: «🞃» SampleBuildJob (выбрать)

«Apply» «Save»

На Дашборде новым тегом «TestDeliveryPipeline» отобразится Pipeline:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Delivery Pipeline** | | **All** | **+** |  | | | |
| **BuildJob** ▶🕑 | | | | | | | |
| **#3 triggered by user admin started 37 minutes ago** | | | | | | | |
| **SampleBuildJob** |  | | | | **SampleDeployJob** |  | **SampleTestJob** |
| SampleBuildJob |  | | | | SampleDeployJob 🗘 |  | SampleTestJob 🗘 |
| 37 minutes ago 1 sec |  | | | | 37 minutes ago 1 sec |  | 36 minutes ago 1 sec |
|  | | | | | | | |
| **#2 triggered by user admin started an hour ago** | | | | | | | |
| **SampleBuildJob** |  | | | | **SampleDeployJob** |  | **SampleTestJob** |
| SampleBuildJob |  | | | | SampleDeployJob 🗘 |  | SampleTestJob 🗘 |
| an hour ago 1 sec |  | | | |  |  |  |
|  | | | | | | | |
| **#1 triggered by user admin started an hour ago** | | | | | | | |
| **SampleBuildJob** |  | | | | **SampleDeployJob** |  | **SampleTestJob** |
| SampleBuildJob |  | | | | SampleDeployJob 🗘 |  | SampleTestJob 🗘 |
| an hour ago 8 sec |  | | | |  |  |  |
|  | | | | | | | |

*\* Серые линии – работы, которые ещё не были соединены между собой на время сборки или сборка не производилась.*

*\* Синие линии – работы, у которых сборка в процессе.*

*\* Зелёные линии – работы, у которых сборка ОК.*

*\* Красные линии – работы, у которых сборка провалилась.*

Слева меню «Edit view» - настроить отображение Pipeline.

Number of pipeline instances per pipeline: 3 «🞃» выбрать кол-во отображаемых запусков

🗹 Enable start of new pipeline build – позволяет запускать Pipeline из Дашборда, из тега – появится кнопка «▶🕑»

🞎 Enable manual triggers – определённые шаги запускать только вручную

🗹 Enable rebuild – в углу каждого шага (кроме первого) появится «🗘» – отдельно запустить сборку шага

🗹 Show total build time – показывает общее время работы Pipeline.

**РАБОТА – УСТАНОВКА СБОРКИ PIPELINE**

Установить Build Pipeline-plugin: «Manage Jenkins» 🡪 «Manage Plugins» 🡪 Available tag 🡪 Search: Build 🡪

🡪 🗹 Build Pipeline 🡪 «Download now and install after restart»

**Install/Update Plugins:** Downloaded Successfully. Will be activated during the next boot

Перезапустить Jenkins: закрыть-открыть CL 🡪 java -jar Jenkins.war --httpPort=9394

Настроить отображение Pipeline: «Dashboard» 🡪 tag «+» 🡪

View name: BuildPipelineTest

🞊 Build Pipeline View

«OK»

Окно «BuildPipelineTest»:

**Displayed Options**

No Of Displayed Builds: 5 «🞃»(выбрать кол-во отображаемых Pipeline, по умолчанию – 1)

**Upstream / downstream config**

Select Initial Job: «🞃» SampleBuildJob (выбрать)

Console Output Link Style: «🞃» New Window (выбрать, чтоб пользоваться мини-иконками «🗅 консоль»)

«Apply» «Save»

Окно «Build Pipeline»:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | | | | | | | | | | | | |  |
|  | **Build Pipeline** | | | | | | | | | | | | | | | |  |
|  |  | | | | 🕑 | | 🗏 | | 🛠 | 🗀 | ⦸ | 🛠 | |  | | |  |
|  |  | | | | Run | | History | | Configure | Add Step | Delete | Manage | |  | | |  |
|  |  | | | | | | | | | | | | | | | |  |
|  |  |  | | | | | | | | | | | | | |  |  |
|  | Pipeline |  | #4 SampleBuildJob | | 🡆 | | #3 SampleDeployJob | | | | | 🡆 | | #3 SampleTestJob |  |
| #4 | 05.01.2022 16:49:15  8.4 sec  admin  🗅⮫ | | 05.01.2022 16:49:35  1.1 sec  admin  🗅⮫ | | | | | 05.01.2022 16:49:50  1.1 sec  admin  🗅⮫ |
|  |  | | | | | | | | | | | | | |  |
|  |  | | | | | | | | | | | | | | | |  |

*\* Run – запускает Пайплайн*

*\* History – история сборок Пайплайна*

*\* Configure – настройки Пайплайна*

*\*Add Step – добавляет новый Item*

*\* Delete – удаляет Пайплайн*

*\* Manage – настройки Дженкинс = Manage Jenkins*

*\* #4 SampleBuildJob – ссылка на сборку*

*\* «*🗅*» –просмотр консоли*

*\* «*⮫*» –перезапустить сборку именно этого шага*

**РАБОТА – ПАРАМЕТРИЗАЦИЯ ПРОЕКТА**

«New Item» 🡪 Name: ParametrizedJob 🡪 «Freestyle Project» 🡪 «ОК»

🗹 This project is parametrized 🡪 «Add Parameter» 🡪 «🞃String Parameter»

Name: name

Default Value: Automation

Description: (blank)

«Add Parameter» 🡪 «🞃Choice Parameter»

Name: choice

Variants: A B C D E

Description: (blank)

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: echo %name% (Win: %...% выведет значение пар-ра «name», Mac/Lnx: $... или ${…})

echo %choice% (Win: %...% выведет значение пар-ра «name», Mac/Lnx: $... или ${…})

«Apply» «Save»

В меню слева вместо «Build now», появится «Build with Parameters» 🡪 Окно проекта с параметрами:

параметром строки, значение которого можно изменить: name: Automation

параметром выбора, значение которого нужно выбрать: choice: «🞃» A (или B или C …) 🡪 «Build»

✓ **Console Output**

|  |
| --- |
| …  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\ParametrizedJob>echo Automation  Automation  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\ParametrizedJob>echo B  B  …  Finished: SUCCESS |

Установить плагин использования checkbox, radio button, …: «Manage Jenkins» 🡪 «Manage Plugins» 🡪 Available tag 🡪 Search: extend 🡪 🗹 Extensible Choice Parameter 🡪 «Download now and install after restart»

«New Item» 🡪 Name: ParametrizedJob 🡪 «Freestyle Project» 🡪 «ОК»

🗹 This project is parametrized 🡪 «Add Parameter» 🡪 «🞃Extended Choice Parameter»

Name: message

Description: (blank)

🞊 Basic Parameter Types

Parameter Type: «🞃Check Boxes»

N of visible items: 4 (сколько чек-боксов по вертикали будет видно без прокрутки)

Delimiter: (blank) (разделитель при перечислении, по умолчанию – запятая)

**Choose Source for Value**

🞊 Value 🡪 Value: A,B,C,D,E,F,G

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: echo %message% (Win: %...% выведет значение пар-ра «name», Mac/Lnx: $... или ${…})

«Apply» «Save»

В меню слева вместо «Build now», появится «Build with Parameters» 🡪 Окно проекта с параметрами:

параметром строки, значение которого выбрать из списка чек-боксов: message: 🗹 A, 🞏 B, …, 🗹 E 🡪 «Build»

✓ **Console Output**

|  |
| --- |
| …  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\ParametrizedJob2>echo A,E  A,C  …  Finished: SUCCESS |

Также можно настроить и переключатели (🞊)

**РАБОТА – УПРАВЛЕНИЕ ИЗ КОНСОЛИ**

«New Item» 🡪 Name: ConsoleJob 🡪 «Freestyle Project» 🡪 «ОК»

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: echo First steps in Jenkins (вывод на консоль)

«Apply» «Save»

Должен быть установлен файл «jenkins-cli.jar» – описано в разделе «ИСПОЛЬЗОВАНИЕ CLI» вначале.

«Manage Jenkins» 🡪 выбрать «Configure Global Security» 🡪 🞊 Everyone can do anything

«Manage Jenkins» 🡪 выбрать «Jenkins CLI» или в адресной строке ввести: http://localhost:9394/cli

Выбрать нужную команду, н-р, «Build» 🡪 окно с образцом и инструкциями и опциями команды для CLI

java -jar jenkins-cli.jar -s http://localhost:8080/ -webSocket build JOB [-c] [-f] [-p] [-r N] [-s] [-v] [-w]

В CL перейти к папке с файлом «jenkins-cli.jar» и подставить команду, заменив «JOB» на «ConsoleJob» и порт

cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsCLI

java -jar jenkins-cli.jar -s http://localhost:9394/ -webSocket build ConsoleJob

В UI можно видеть, что произошла новая сборка.

Флаг «-f» – Вывести в консоль номер сборки и статус (SUCCESS/FAILED)

Флаги «-s -v» – Вывести в консоль подробности сборки + результат Windows-команды

Если включена опция «Configure Global Security» 🞊 Authorized users can do anything – в CL надо авторизоваться

«-auth admin:admin» и набирать эту команду при каждом запросе

java -jar jenkins-cli.jar -s http://localhost:9394/ -auth admin:admin -webSocket build ConsoleJob -f

Добавить параметр к проекту:

«ConsoleJob» 🡪 «Configuration»

🗹 This project is parametrized 🡪 «Add Parameter» 🡪 «🞃String Parameter»

Name: MESSAGE

Default Value: (blank)

Description:

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: echo %MESSAGE% (Win: %...% выведет значение пар-ра, Mac/Lnx: $... или ${…})

«Apply» «Save»

Меню справа «Build with Parameters» 🡪 Окно проекта с параметрами: MESSAGE: Hello, CLI !!! 🡪 «Build»

✓ **Console Output**

|  |
| --- |
| …………  ……….  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\ConsoleJob>echo Hello, CLI !!!  Hello, CLI !!!  ………. |

java -jar jenkins-cli.jar -s http://localhost:9394/ -webSocket build ConsoleJob -s -v

↓

…

echo is on (вместо значения параметра консоль просто сообщит, что ф-ция «echo» вкл, т.к. не указан пар-р)

….

SUCCESS

Флаг «-p» – ввести установленное название пар-ра: «MESSAGE», и любое значение:

java -jar jenkins-cli.jar -s http://localhost:9394/ -webSocket build ConsoleJob -s -v -p MESSAGE=CLI,HelloAgain!

↓

…

CLI,HelloAgain

….

SUCCESS

Чтобы сообщение было с пробелом – заключить значение пар-ра в двойные кавычки

java -jar jenkins-cli.jar -s http://localhost:9394/ -webSocket build ConsoleJob -s -v -p MESSAGE=”CLI, Hello Again!”

↓

…

CLI, Hello Again

….

SUCCESS

**BUILD MONITOR VIEW**

Установить Build Monitor View plugin: «Manage Jenkins» 🡪 «Manage Plugins» 🡪 Available 🡪 Search: Build Mon 🡪

🡪 🗹 Build Monitor View 🡪 «Download now and install after restart»

Настроить отображение Build Monitor View: «Dashboard» 🡪 tag «+» 🡪

View name: BuildMonitor1

🞊 Build Monitor View

«OK»

Окно «BuildMonitor1»:

**Job Filters**

Jobs (выбрать проекты для отображения)

🗹 AutoDeploymentTest

🗹 ConsoleJob

🗹 ParametrizedJob

🗹 ParametrizedJob2

🗹 SampleBuildJob

🗹 SampleDeployJob

🗹 SampleTestJob

**Build Monitor - View Settings**

Title: BuildMonitor1 (заголовок)

Ordered by: Status «🞃» (сортировка по: Name / Full Name / Status / Duration / …)

«Apply» «Save»

Отобразится монитор. Поля будут отображаться зелёным/красным в зависимости от статуса.

Размер шрифта, кол-во и расположение полей можно менять: справа вверху «шестерёнка» 🡪 Configure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ⬤  LOGO | **BuildMonitor1** | | | | ☼  SETTINGS |
|  | **AutoDeploymentTest** |  |  | **ConsoleJob** |  |
| **#10** |  | **22 h ago** | **#13** |  | **2 h ago** |
|  | **ParametrizedJob** |  |  | **ParametrizedJob2** |  |
| **#4** |  | **5 h ago** | **#5** |  | **1 h ago** |
|  | **SampleBuildJob** |  |  | **SampleDeployJob** |  |
| **#5** |  | **6 h ago** | **#5** |  | **6 h ago** |
|  | **SampleTestJob** | | | |  |
| **#5** |  | | | | **6 h ago** |
|  | | | | | |

**РАБОТА – НАСТРОЙКА E-MAIL НОТИФИКАЦИИ**

«Manage Jenkins» 🡪 «System Configuration»

**Extended E-mail Notification**

SMTP server: smtp.gmail.com (для Gmail)

Default user e-mail suffix: (blank)

SMTP Port: 465 (465 или 587, или 25 – для Gmail)

🗹 Use SMTP Authentication:

Username: mail1@gmail.com (E-mail с которого уведомления будут отправляться)

Password: 12345

🗹 Use TLS

Reply-To Address: mymail@gmail.com

Charset: UTF-8

Test e-mail recipient: mymail@gmail.com

«Test Configuration»

«Apply» «Save»

«AnyProject» 🡪 «Configuration»

**Post Build Action**

«Add Post Build Action» 🡪 E-mail notification

Recipients: recipient-1@gmail.com, recipient-2@gmail.com

**РАБОТА – АВТОМАТИЧЕСКОЕ РАЗВЁРТЫВАНИЕ**

Установить Deploy-plugin: «Manage Jenkins» 🡪 «Manage Plugins» 🡪 Available tag 🡪 Search: Deploy 🡪

🡪 🗹 Deploy to container 🡪 «Download now and install after restart»

**Install/Update Plugins:** Downloaded Successfully. Will be activated during the next boot

Перезапустить Jenkins: закрыть-открыть CL 🡪 java -jar Jenkins.war --httpPort=9394 (на 8080 будет работать Tomcat)

Браузер: http://localhost:9394 🡪 Login: admin 🡪 Password: 0bb2754dbafc460ca8cfc0db4ff40f7a

(если пароль забыт – можно посмотреть в файле: «.\JenkinsHome\secrets» 🡪 initialAdminPassword)

Создать пользователя Tomcat:

Windows: C:\Users\Admin1\Desktop\Tools\apache-tomcat-8.5.73\conf 🡪 tomcat-users.xml 🡪 Edit

Вставить код по образцу и инструкции (приведенным тут же в файле в виде комментов <!-->) или удалить <!-->

|  |
| --- |
| 36 | <role rolename="manager-gui"/>  37 | <role rolename="manager-script"/>  38 | <role rolename="manager-jmx"/>  39 | <role rolename="manager-status"/>  40 | <user username="admin" password="admin" roles="manager-gui,manager-script,manager-jmx,manager-status"/> |

«Save»

Google: sample war file download 🡪 https://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/ 🡪 «here» 🡪 загрузится файл «sample.war» переместить его в папку .\JenkinsHome\workspace\AutoDeploymentTest

Создать новую работу по сборке, которая произведёт сборку проекта и в конце создаст war/ear-файл

«New Item» 🡪 Name: AutoDeploymentTest 🡪 «Freestyle Project» 🡪 «ОК»

Окно «AutoDeploymentTest» 🡪

**Build:** «Add build step» 🡪 🞊 Execute Windows batch command

Command: cd C:\...\Projects\Jenkins-Git-Java (перейти в папку с текстовым файлом)

type Changes-new.txt (вывести содержание текстового файла в консоль)

**Post Build Actions:** «Add Step» 🡪 Deploy war/ear file to container (после – war/ear – в контейнер)

WAR/EAR files: \*\*/\*.war (какой именно файл создастся, посмотреть маску – «?»)

Context path: sample.war (контекстный путь к файлу в пределах проекта)

Containers: «Add Container» 🡪 Tomcat 7.x

Login-Pass настроенные в Tomcat Credentials: «Add» 🡪 Login: admin Password: admin «OK» 🡪 «-none-🞃» 🡪 «admin/\*\*\*\*\*»

Tomcat URL: http://localhost:8080

«Apply» «Save»

Запустить сборку «AutoDeploymentTest»

✓ **Console Output**

|  |
| --- |
| Started by user admin  Running as SYSTEM  Building in workspace C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\AutoDeploymentTest  [AutoDeploymentTest] $ cmd /c call C:\Users\Admin1\AppData\Local\Temp\jenkins8226016441021187928.bat  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\AutoDeploymentTest>cd C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>type Changes-new.txt  This is Test file for new Git repo changes  And follow changes are the newest!  a. Change #1  b. Change #2  c. Change #3  d. Change #4  e. Change #5  C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\Projects\Jenkins-Git-Java>exit 0  [DeployPublisher][INFO] Attempting to deploy 1 war file(s)  [DeployPublisher][INFO] Deploying C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\AutoDeploymentTest\sample.war to container Tomcat 8.x Remote with context sample.war  [C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\AutoDeploymentTest\sample.war] is not deployed. Doing a fresh deployment.  Deploying [C:\Users\Admin1\Desktop\Tools\jenkins-2.319.1\JenkinsHome\workspace\AutoDeploymentTest\sample.war]  Finished: SUCCESS |

В Tomcat убедиться, что war-файл развёрнут: http://localhost:8080/sample.war – страница «Sample "Hello, World" Application»

**Tomcat**

**? Tomcat (в старых версиях — Catalina)** — контейнер сервлетов с открытым исходным кодом.

**? Для чего используется Tomcat** — Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server, а также в качестве контейнера сервлетов в серверах приложений JBoss и GlassFish.

**СКАЧАТЬ TOMCAT**

* Сайт: https://tomcat.apache.org/download-80.cgi
* Справа, в секции «Download» выбрать версию: н-р, «Tomcat 8»
* В основном контекстном поле выбрать архив для скачивания: «64-bit Windows zip (pgp, sha512)»
* Распаковать и переместить папку «apache-tomcat-8.5.73»: н-р, в C:\Users\Admin1\Desktop\Tools
* Скопировать и переместить файл «Jenkins.war» в папку «.\apache-tomcat-8.5.73\webapps»
* Запустить Tomcat: CL 🡪 перейти в папку «bin» с файлом «startup.sh»

cd C:\Users\Admin1\Desktop\Tools\apache-tomcat-8.5.73\bin

C:\Users\Admin1\Desktop\Tools\apache-tomcat-8.5.73\bin>startup.sh

* Браузер: http://localhost:8080 🡪 проверить что Tomcat работает
* (Командная строка и браузер – Дженкинс остановлен!)
* Внутри Tomcat перейти в Дженкинс: http://localhost:8080/jenkins
* В браузере отобразится Дженкинс, но он работает внутри Tomcat-сервера
* Создать внутри Томкэт-Дженкинса проект «Test4»
* Запустить Дженкинс отдельно на другом порту (н-р, 9090): java -jar jenkins.war --httpPort=9090
* Проект «Test4» отобразится и в Независимом-Дженкинсе (т.к. он стартовал уже после создания «Test4»)
* Одновременно работают Томкэт-Дженкинс (порт 8080) и Независимый-Дженкинс (порт 9090)
* Томкэт-Дженкинс (порт 8080): создать проект «Test8»
* Независимый-Дженкинс (порт 9090): создать проект «Test9»
* Теперь Томкэт-Дженкинс не видит проект «Test9», а Независимый-Дженкинс не видит проект «Test8»
* Чтобы синхронизировать Дженкинс-в-контейнере и Дженкинс-без – перезапустить оба сервиса.
* После перезапуска Томкэт и Дженкинс в обоих оболочках появятся и «Test8», и «Test9»
* Для остановки Tomcat ОБЯЗАТЕЛЬНО! в командной строке, находясь в папке «bin» набрать: shutdown.sh

**Nets, OSI, TCP/IP**

? Классификация сетей

* Тип коммутации
  + Коммутация каналов (телефонная сеть) – надо установить связь между отправителем и получателем (недостаток таких сетей – при выходе из строя одного из промежуточных узлов, соединение разрывается)
  + Коммутация пакетов (компьютерная сеть) – сообщение разбивается на части – пакеты –каждый из которых передаётся отдельным путём (преимущество таких сетей – отказоустойчивость: при выходе из строя одного из промежуточных узлов, пакеты идут обходным путём – на каждом промежуточном узле, для каждого пакета решается задача маршрутизации)

|  |  |  |
| --- | --- | --- |
| Коммутация каналов | Коммутация пакетов | (задача маршрутизации) |
|  |  |  |

* Технологии передачи
  + Широковещательные сети – данные, которые передаются в сеть 1 компьютером – доступны всем компам в сети
  + Точка-точка – данные передаются от 1 компа другому, от другого – третьему, приходится преодолевать несколько промежуточных узлов

|  |  |
| --- | --- |
| Широковещательные | Точка-точка |
|  |  |

* Протяжённость

|  |  |  |
| --- | --- | --- |
| Название | Протяжённость | Расположение |
| Персональные | 1 м | Стол |
| Локальные | 10 м – 1 км | Комната, здание, кампус |
| Муниципальные | 10 км | Город |
| Глобальная | 100 – 1000 км | Страна, Континент |
| Объединение сетей | 10 000 км | Мир (Интернет) |

? Топология – способ соединения компьютеров между собой для образования сети.

Топология – это граф:

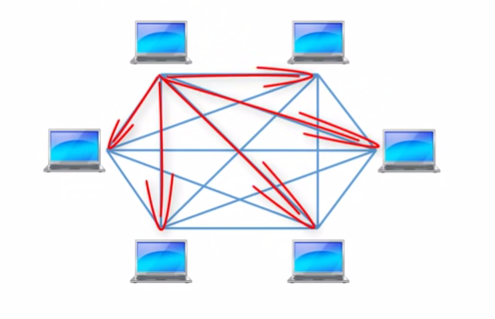
* Вершины – узлы сети (компьютеры и сетевое оборуд-ие)
* Рёбра – связи между узлами (физические или информационные)

? Базовые топологии

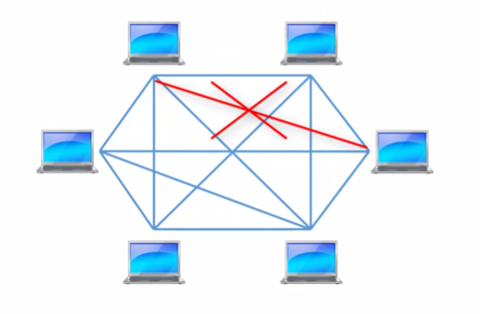
Полносвязная – каждое у-во в сети имеет прямое соединение со всеми другими у-вами

«+» Прямой канал каждого у-ва в сети – всегда можно передавать данные

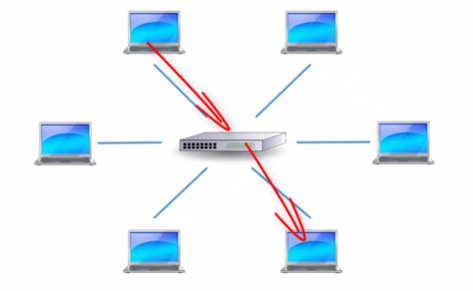
«–» Нужно много соединений и много адаптеров (в сети на 1000 компов эта топология не возможна)



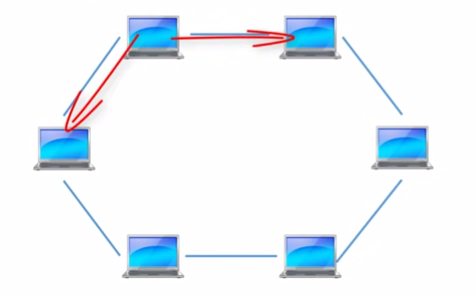
Ячеистая – частный случай полносвязной, когда из полносвязной удалены некоторые соединения



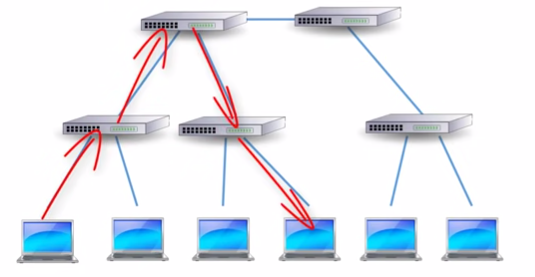
Звезда – компьютеры подключаются не друг к другу, а к центральному у-ву (концентратор, коммутатор, маршрутизатор, точка доступа Wi-Fi) – и данные передаются через это центральное у-во



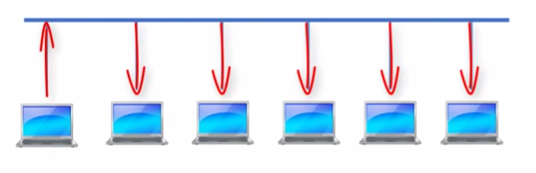
Кольцо – каждый компьютер соединяется с двумя соседними компьютерами



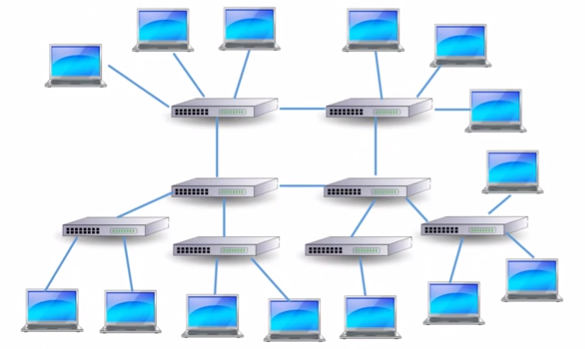
Дерево – компьютеры и сетевое оборудование образуют древо – для передачи данных нужно пройти 1 или несколько промежуточных ус-в.



Общая шина – все компьютеры в сети подключены к некоей среде передачи данных (медный кабель) – данные, передающиеся в эту среду, доступны всем компьютерам, подключённым к общей шине



**? Смешенная топология** – на практике базовые топологии не используются, а используется смешенная топология (н-р, центральные у-ва – топология кольцо, к которым в свою очередь подключаются у-ва по топологии звезда и дерево)

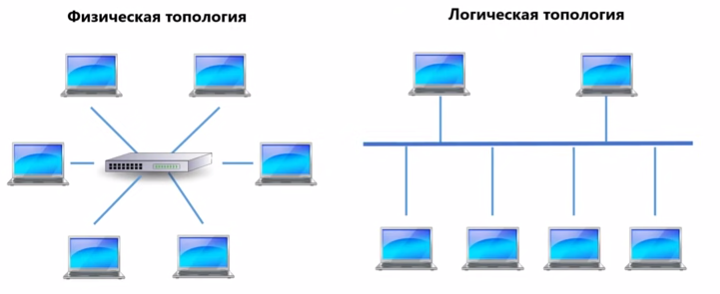


**? Физическая / логическая топологии**

* Физическая топология – соединение у-в в сети
* Логическая топология – правила распространения сигналов в сети

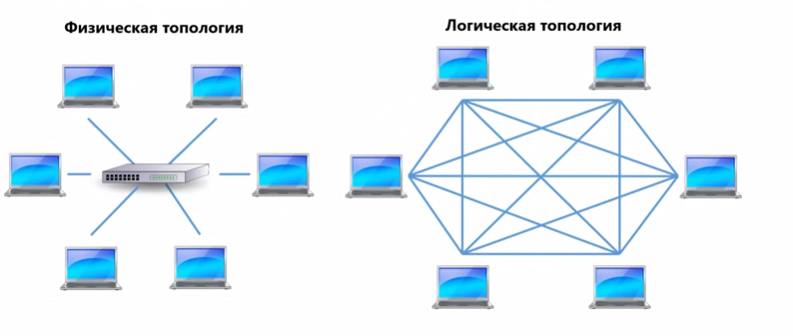
Классический Ethernet:

* Физическая – звезда – все компьютеры подключаются к промежуточному у-во – концентратору
* Логическая – общая шина – концентратор передаёт данные поступившие на 1 порт передаёт на все порты



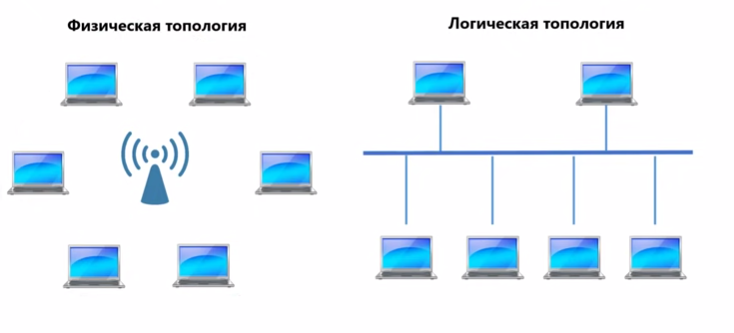
Коммутируемый Ethernet

* Физическая – звезда – все компьютеры подключаются к промежуточному у-во – концентратору
* Логическая – полносвязная – концентратор может соединить каждый компьютер с любым другим



Технология Wi-Fi

* Физическая – отсутствует – нет физических соединений
* Логическая – общая шина – всё что 1 комп передаёт через эфир могут принять все компы в зоне сигнала



**? Зачем нужны стандарты**

* На раннем этапе развития сетей (1960–1970) стандартизации не было.
* Оборудование разных производителей не могло взаимодействовать по сети:
  + Несовместимость сетевого оборудования
  + Несовместимость ПО
  + Разные протоколы
* Решение – стандарты, которые играют огромную роль в работе сетей:
  + Оборудование разных поставщиков
  + ПО разных производителей
  + Разные ОС и платформы
  + Разные у-ва

**? Типы стандартов**

* De-jure (формальные, юридические) – стандарты, принятые по формальным законам стандартизации
* De-facto (фактические) – установившиеся сами-собой (н-р, новая технология, кот быстро распространилась и стала популярной – стек протоколов TCP/IP)

**? Стандарты для сетей**

* ISO (Международная Организация по Стандартизации) – Эталонная модель взаимодействия открытых систем
* IEEE (Институт Инженеров по Электронике и Электротехнике) – Технологии передачи данных
  + 802.3 – Ethernet
  + 802.11 – Беспроводные локальные сети (Wi-Fi)
  + 802.15 – Персональные сети (BlueTooth)
  + 802.16 – Широкополосные беспроводные сети (WiMAX)
  + …
* IAB (Совет по Архитектуре Интернета) – Протоколы Интернет
  + IRTF (Internet Research Task Force) – Долгосрочные перспективные исследования
  + IETF (Internet Engineering Task Force) – Выпускает стандарты на сетевые протоколы
    - RFC (Request for Comments) – Док-ты с описанием работы протоколов (не называется «стандарты», но являются таковыми)
    - RFC 793 – протокол TCP
    - RFC 791 – протокол IP
    - RFC 826 – протокол ARP
    - RFC 792 – протокол ICMP
    - RFC 2131 – протокол DHCP
    - …
* W3C (Консорциум Всемирной Паутины) – Стандарты Web
  + Рекомендации W3C – документы по стандартам Web
  + Рекомендация HTML (Hypertext Markup Language)
  + Рекомендация CSS (Cascading Style Sheets)
  + Рекомендация WSA (Web Service Architecture)
  + Рекомендация XML (Extensible Markup Language)
  + …

**? Сложность создания сетей**

* Многообразие оборудования и ПО – как работать с большим разнообразием оборудования и ПО
* Надёжность – сеть должна работать, даже если выйдут из строя некоторые из её у-в
* Развитие сети – добавлять в сеть новые компьютеры и сети отдельных стран и континентов
* Распределение ресурсов – для всех пользователей одновременно всех ресурсов не хватает (дни пик)
* Качество обслуживания – данные передавать без искажения или с большой скоростью
* Безопасность – чтобы злоумышленники не украли персональные данные и т.п.

**? Решение проблемы сложности создания компьютерных сетей**

* Декомпозиция на отдельные подзадачи
* Шаблон «Уровни»

**? Декомпозиция на отдельные подзадачи: Шаблон «Уровни»**

* Компьютерные сети строятся в виде ур-ней, организованных один над другим.
* Каждый ур-нь решает одну или несколько тесно связанных между собой задач.
* Ур-нь предоставляет сервис вышестоящему ур-ню.
* Вышестоящие ур-ни могут решать уже более сложные задачи.

Среда передачи данных

Уровень 3

Компьютер 1

Уровень 2

Уровень 1

Уровень 3

Компьютер 2

Уровень 2

Уровень 1

«+» Ур-нь решает одну конкретную или несколько тесно связанных задач.

«+» Выполняется изоляция решений друг от друга – если в сети происходит изменение, нужно поменять оборудование, только соответствующего ур-ня, а не всех ур-ней сразу.

**? Сервис / Протокол / Интерфейс**

* Сервис – что делает уровень
* Протокол – как уровень это делает
* Интерфейс – как получить доступ к сервису уровня
* Сервис – описывает какие функции реализует уровень
* Интерфейс – набор примитивных операций, которые нижний уровень предоставляет верхнему
* Протокол – правила и соглашения, используемые для связи уровня N одного компьютера с уровнем N другого компьютера

Протокол ур-ня N

Интерфейс ур-ня N

Уровень N

Интерфейс ур-ня N–1

Уровень N

Интерфейс ур-ня N

Интерфейс ур-ня N–1

Пользователи или вышестоящие уровни взаимодействуют с интерфейсом уровня. Они понимают, что должен делать этот уровень в описании его сервиса и вызывают некоторые функции интерфейса. Протокол является реализацией этого взаимодействия, и он скрыт от вышестоящих уровней и от пользователей. Если заменить один протокол другим, то в работе вышестоящих уровней ничего менять не придётся. С другой стороны, можно вносить изменения в интерфейсы одного компьютера, но он всё равно будет взаимодействовать с другими компьютерами, используя один и тот же протокол. Благодаря этому по сети успешно взаимодействуют компьютеры, работающие на разных платформах, например, Windows и Linux.

Интерфейс – реальное взаимодействие внутри одного компьютера, где ур-нь N вызывает функции ур-ня N–1

Протокол – виртуальное взаимодействие между компьютерами – реально соединяются только ур-ни, работающие с физической средой; единственный способ передать информацию – использовать заголовок соответствующего ур-ня.

**? Архитектура сети** – набор ур-ней и протоколов сети (интерфейсы в архитектуру не входят, т.к. они могут быть разными на разных аппаратных платформах).

**? Стек протоколов** – иерархически организованный набор протоколов, достаточный для организации взаимодействия по сети.

**? Эталонные модели организации сетей** – какие ур-ни должны быть для объединения + ф-ции уровней

* Модель ISO OSI (The Open Systems Interconnection model) – модель взаимодействия открытых систем
  + Юридический стандарт Международной Организации Стандартизации
  + 7 уровней (протоколы не входят в модель)
  + Хорошая теоретическая проработка
  + На практике не используется
* Модель TCP/IP
  + Фактический стандарт на основе популярного стека протоколов TCP/IP
  + 4 уровня
  + Протоколы TCP/IP хорошо используются на практике
  + Основа Интернет

**? Инкапсуляция** – включение сообщения вышестоящего уровня в сообщение нижестоящего уровня (сообщение = заголовок + данные (+ концевик))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Компьютер A** | | |  | **Компьютер B** | | |
| Ур-нь 3  (L3) | Формирует из данных от приложения msg и передаёт его на L2 | ██████ |  | ██████ | Извлекает из msg данные и передаёт его приложению | Ур-нь 3  (L3) |
| Ур-нь 2  (L2) | L2 разделяет одно большое msg на две части и добавляет к ним заголовок L2 и отправляет их на L1 | ████ ████ |  | ████ ████ | L1 удаляет заголовки, соединяет два msg в одно большое и передаёт на L3 | Ур-нь 2  (L2) |
| Ур-нь 1  (L1) | L1 добавляет заголовок и концевик (т.к. L1 взаимодействует со средой передачи). | █████▌ █████▌ |  | █████▌ █████▌ | L1 удаляет заголовки и концевики, и передаёт на L2 | Ур-нь 1  (L1) |
| Среда | **Msg1 = заголовок-L1 + данные-L1 (= заголовок-L2 + данные-L2 (= I-часть-L3)) + концевик-L1 + Msg2 = заголовок-L1…** | | | | | Среда |

**OSI Model**

**? Модель OSI (The Open Systems Interconnection model)** – модель взаимодействия открытых систем – юридический стандарт, принятый в 1983. Открытая система – в терминологии системы – система, построенная в соответствиями с открытыми спецификациями – которые доступны всем и соответствуют стандартам (а не открытая система – в смысле система с открытыми исходными кодами, которая распространяется бесплатно).

**? Преимущества открытых систем:**

* Возможность строить сети из оборудований разных производителей
* «Безболезненная» замена отдельных компонентов сети
* Лёгкость объединения нескольких сетей

**? Модель OSI:**

* Описывает:
  + 7 уровней организации сети
  + Назначение каждого уровня
* Не является сетевой архитектурой!
* Протоколы описаны в отдельных стандартах, и не входят в Модель
* На практике Модель OSI не используется
* Модель OSI имеет хорошую теоретическую проработку вопросов сетевого взаимодействия.
* Модель OSI используется в качестве «общего языка» для описания разных сетей

🡪 🡪🡪 Среда передачи данных 🡪 🡪 🡪

Сетевой

Канальный

Физический

Транспортный

Компьютер 1

Сеансовый

Представления

Прикладной

Протоколы

Интерфейсы

Интерфейсы

Сетевой

Канальный

Физический

Транспортный

Компьютер 2

Сеансовый

Представления

Прикладной

🡪 Data Transmission Environment 🡪

Network

Data link

Physical

Transport

Host 1

Session

Presentation

Application

Protocols

Interfaces

Interfaces

Network

Data link

Physical

Transport

Host 2

Session

Presentation

Application

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Модель OSI** | | | | | | | | |
| **Уровень (Layer)** | | | **Тип данных (PDU)** | | **Функции** | | **Примеры** | |
| Host  layers | 7. Прикладной (Application) | | Данные (data) | | Доступ к сетевым  службам | | HTTP, FTP, POP3, WebSocket | |
| 6. Представления (Presentation) | | Представление  и шифрование данных | | ASCII, EBCDIC | |
| 5. Сеансовый (Session) | | Управление  сеансом связи | | RPC, PAP, L2TP | |
| 4. Транспортный (Transport) | | Сегменты (segment) /Дейтаграммы (datagram) | | Прямая связь между конечными пунктами и надежность | | TCP, UDP, SCTP, PORTS | |
| Media  layers | 3. Сетевой (Network) | | Пакеты (packet) | | Определение маршрута и логическая адресация | | IPv4, IPv6, IPsec, AppleTalk | |
| 2. Канальный (Data Link) | | Кадры (frame) | | Физическая  адресация | | PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта | |
| 1. Физический (Physical) | | Биты (bit) | | Работа со средой передачи, сигналами и двоичными данными | | USB, кабель («витая пара», коакс., оптовол.), р/канал | |
| **OSI Model** | | | | | | | | |
| **Layer** | | **PDU** | | **Function** | | **Protocols** | | **Technics** |
| Host  layers | 7. Application | Data | | User App Layer | | HTTP, FTP, POP3, WebSocket | | Application, Firewall, VPN, Proxy, SSL/TLS |
| 6. Presentation | Data encrypting, conversion | | ASCII, EBCDIC | |
| 5. Session | Open & Close Session | | RPC, PAP, L2TP | |
| 4. Transport | Segment /Datagram | | Ensure Delivery | | TCP, UDP, SCTP, PORTS | | Balancer |
| Media  layers | 3. Network | Packet | | Control Routing | | IPv4, IPv6, IPsec, AppleTalk | | Router |
| 2. Data Link | Frame | | Error free data transfer | | PPP, IEEE 802.22, Ethernet, DSL, ARP, NIC | | Switch |
| 1. Physical | Bit | | Passing bits | | USB, cable, radio channel | | Hub, RJ45 |

|  |  |  |
| --- | --- | --- |
| **Модель OSI** | | |
| **Уровень / Тип данных** | | **Функции** |
| Host  layers | 7. Прикладной  *Данные/Сообщение* | Прикладной уровень – то ради чего строится сеть – набор приложений, которые могут использовать пользователи: веб-страницы, соцсети, видео/аудио, email, файлы и др. |
| 6. Представления  *Данные/Сообщение* | Согласование синтаксиса и семантики (смысл) передаваемых данных (формат символов и чисел). Шифрование/дешифрование (TLS/SSL). |
| 5. Сеансовый  *Данные/Сообщение* | Установка сеансов связи: управление диалогом (очерёдность передачи) и маркерами (предотвращение одновременного выполнения критической операции), синхронизация (метки в сообщениях). |
| 4. Транспортный  *Сегмент/Дейтаграмма* | Передача данных между **процессами** на хостах: предоставление надёжного, защищённого от ошибок канала с гарантированным порядком следования; сквозной ур-нь: сообщения изолированы от обор-ния |
| Media  layers | 3. Сетевой  *Пакет* | Объединение сетей, построенных на основе различных технологий: создание составной сети, адресация (какой именно комп, независимо от ур-ня 2), маршрутизация – определение маршрута пакетов |
| 2. Канальный  *Кадр* | Передача сообщения по каналу. Начало-конец сообщения в потоке бит. Обнаружение и коррекция ошибок. В широковещательной сети: управление доступом к среде (в один момент данные передавал только 1 комп), физическая адресация (на какой именно комп из множества передать инфу). |
| 1. Физический  *Бит* | Передача бит по физическому каналу связи. Не вникает в смысл передаваемой информации.  Задача: как передать биты информации в виде сигналов, передаваемых по среде. |

**? Пример работы Модели OSI** – военный штаб ☺

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  | INTELLIGENCE: «ENEMY!» | | |  |  |  |  |  |  | «ENEMY!» |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | BLD 105 |  |  |  |  |  | BLD 106 |  |
|  | **HQ #1** | |  |  |  |  | **HQ #2** | |  |  |
| FL 7 | APPLICATION | |  | GEN в Штабе №1 получает от разведки инфу о том, что враг близко. Ему надо предупредить Штаб №2. Он составляет message [ENEMY IS NEAR] и передаёт msg на FL 6. |  | FL 7 | APPLICATION | |  | GEN в Штабе №2 получает сообщение [ENEMY IS NEAR], интерпретирует его и понимает, что враг близко. |
| GEN | Interpretation | |  |  | GEN | Interpretation | |  |
|  |  |  |  |  |  |  |  |  |
| FL 6 | PRESENTATION | |  | BRG получает msg [ENEMY IS NEAR], 🡪 шифрует его:  🡪 [EAGLES IS NAPPING] и передаёт msg целиком, но в зашифрованном виде на FL 5. |  | FL 6 | PRESENTATION | |  | BRG дешифрует [EAGLES IS NAPPING] 🡪 [ENEMY IS NEAR] и передаёт сообщение на FL 7 |
| BRG | Coding | |  |  | BRG | Coding | |  |
|  |  |  |  |  |  |  |  |  |
| FL 5 | SESSION | |  | CNL по радио связывается со Штабом №2 и говорит: «К вам будет доставлено сообщение. Откройте дверь! Это Штаб №1, пароль dg34fcj8ks0pVtu5ElvvB6» и передаёт msg на FL 4. |  | FL 5 | SESSION | |  | CNL подтверждает, что да, мы ждали это сообщение, как и было сообщено из Штаба №1, и передаёт msg на FL 6 |
| CNL | Info | |  |  | CNL | Info | |  |
|  |  |  |  |  |  |  |  |  |
| FL 4 | TRANSPORT | |  | MAJ разделяет одно сообщение на части и к каждой добавляет адрес Штаба №2:  [106-EAGLES] [106-IS] [106-NAPPING] и передаёт msg на FL 3 |  | FL 4 | TRANSPORT | |  | MAJ удаляет адрес своего Штаба и собирает все части в одно [EAGLES IS NAPPING], и передаёт msg на FL 5 |
| MAJ | Defragm+Address | |  |  | MAJ | Defragm+Address | |  |
|  |  |  |  |  |  |  |  |  |
| FL 3 | NETWORK | |  | CPT выбирает из многих путей самый свободный и короткий, добавляет к каждой части номер дороги [55-106-EAGLES] [55-106-IS] [55-106-NAPPING], передаёт msg на FL 2 |  | FL 3 | NETWORK | |  | CPT преобразует msg – удаляет № пути: [106-EAGLES] [106-IS] [106-NAPPING], и передаёт msg на FL 4 |
| CPT | Route | |  |  | CPT | Route | |  |
|  |  |  |  |  |  |  |  |  |
| FL 2 | DATA LINK | |  | LTN выбирает дверь-шлюз, которая выведет на путь №55, добавляет № шлюза к каждой части [#5-55-106-EAGLES] [#5-55-106-IS] [#5-55-106-NAPPING], передаёт msg на FL 1 |  | FL 2 | DATA LINK | |  | LTN преобразует msg – удаляет № шлюза [55-106-EAGLES] [55-106-IS] [55-106-NAPPING], передаёт msg на FL 3 |
| LTN | Gateway | |  |  | LTN | Gateway | |  |
|  |  |  |  |  |  |  |  |  |
| FL 1 | PHYSICAL | |  | CAD преобразует части в «0» и «1» [000110100011001011] [010110101001001011] [111110101000101011] и передаёт по проводам – через шлюз #5 по пути #55 в здание #106 |  | FL 1 | PHYSICAL | |  | CAD преобразует: [00011] [01101] [11101] 🡪 [#5-55-106-EAGLES] [#5-55-106-IS] [#5-55-106-NAPPING], передаёт msg на FL 2 |
| CAD | Transmission | |  |  | CAD | Transmission | |  |
|  | ∏⁴ | ∏⁵ | ∏⁶ |  |  | ∏⁶ | ∏⁵ | ∏⁴ |
|  | **🗶** | **✓** | **🗶** |  |  |  | **🗶** | **✓** | **🗶** |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **🗶** |  | **🗶** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | **✓** |  |  |  |  |  | **✓** |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **🗶** |  |
|  | **🗶** |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

**TCP/IP Model**

**?** Модель TCP/IP – никто отдельно не принимал стандартов модели TCP/IP, просто стек протоколов TCP/IP оказался настолько популярным, что все стали использовать этот стек. Стек TCP/IP создавался для объединения больших компьютеров в университетах по линиям телефонной связи соединениям точка-точка. Когда появились новые технологии (широковещательные, спутниковые) просто стека протоколов оказалось недостаточно, необходима была модель, которая бы говорила как строить сети на орснове технологий, чтобы в них мог работать стек протоколов TCP/IP.

* Фактический стандарт на основе популярного стека протоколов TCP/IP
* 4 уровня
* Протоколы TCP/IP хорошо используются на практике
* Основа Интернет

|  |  |  |  |
| --- | --- | --- | --- |
| **Модель OSI** | **5 ур OSI-TCP/IP** | **Модель TCP/IP** | |
| **Уровень (Layer)** | **Уровень (Layer)** | **Уровень (Layer)** | **Функции** |
| 7. Прикладной (Application) | 5. Прикладной | 4. Прикладной (Application) | Сочетает в себе 3 ур-ня OSI – на практике, если приложению (ур.7) нужны ф-ции ур.5 или ур.6, то оно само должно их реализовывать. |
| 6. Представления (Presentation) |
| 5. Сеансовый (Session) |
| 4. Транспортный (Transport) | 4. Транспортный | 3. Транспортный (Transport) | Связь между двумя процессами на разных хостах |
| 3. Сетевой (Network) | 3. Сетевой | 2. Интернет (Internet) | Поиск маршрута в составной сети |
| 2. Канальный (Data Link) | 2. Канальный | 1. Сетевых интерфейсов (Network Access) | Интерфейс взаимодействия с разными сетевыми технологиями (Ethernet, Wi-Fi) |
| 1. Физический (Physical) | 1. Физический |

**? Сравнение моделей OSI и TCP/IP**

|  |  |  |
| --- | --- | --- |
| **Характеристика** | **Модель OSI** | **Модель TCP/IP** |
| Достоинства | Хорошая теоретическая проработка. Разделено понятие интерфейса и реализации. | Протоколы широко применяются |
| Недостатки | Протоколы не используются | Плохая теоретическая проработка, подходит только для сетей на основе стека TCP/IP |
| Применение | Модель для описания разных типов сетей (Fibre Channel, Infiniband, SS7 – телефонная сигнализация) | Сети на основе стека TCP/IP – Интернет |

**? Стек протоколов TCP/IP**

Прикладной

Транспортный

Сетевой

Сетевых интерфейсов

**HTTP**

**SMTP**

**FTP**

**DNS**

**TCP**

**UDP**

**IP**

**ICMP**

**ARP**

**DHCP**

**Ethernet**

**Wi-Fi**

**DSL**

|  |  |
| --- | --- |
| Прикладной | HTTP – Веб  SMTP – Передача почты  FTP – Передача файлов  DNS – Назначение IP-адресам более понятных для людей имён |
| Транспортный | TCP – Медленная передача данных с гарантией доставки  UDP – Быстрая передача данных без гарантии доставки |
| Сетевой | IP – Обеспечивает доставку пакетов, тем самым объединяет сегменты сети в одну сеть  ICMP – Передача сообщений об ошибках: услуга не доступна, хост/маршрутизатор не отвечают  ARP – Протокол для определения MAC-адреса по IP-адресу другого компьютера  DHCP – Протокол для автоматического получения IP-адреса сетевыми устройствами |
| Сетевых интерфейсов | Ethernet – Кабельный  Wi-Fi – Радиоэфир  DSL – Модем через телефонные линии |

**Уровень 1 – Физический уровень**

**? Основные задачи Физического уровня:**

* Передача потока бит по среде передачи данных (единица передачи информации – бит)

*\* Не вникает в смысл передаваемой информации*

**? Представление сигналов**

Задача: как передать биты информации в виде сигналов, передаваемых по среде.

Теоретически: 0 и 1 легко преобразуются в сигналы. Но на практике сигнал искажается. Получатель должен определить: что же именно передал ему отправитель, где тут 0, а где – 1?



**? Модель канала связи**

Не важно, как Физический ур-нь передаёт сообщение, важно, что у нас есть канал связи, по которому можно отправить сообщение от отправителя к получателю.

Сообщение

Отправитель

Получатель

Характеристики канала связи:

* Пропускная способность (бит/с) – сколько данных можно передать за единицу времени
* Задержка – сколько времени пройдёт, пока сообщение от отправителя дойдёт к получателю.
* Количество ошибок:
  + Если ошибки возникают часто – протоколы и сетевые технологии должно исправлять их
  + Если ошибки возникают редко – они могут исправляться на Транспортном ур-не, а сетевое оборудование может не обеспечивать гарантию доставки данных и отсутствие ошибок

**? В зависимости от направления, какие бывают каналы – Типы каналов связи:**

* Симплексный
* Дуплексный
* Полудуплексный

**? Среды передачи данных**

* **Кабели:**
  + Телефонный кабель
  + Коаксиальный кабель
  + Витая пара (часто применяется) – обычно 4 витых изолированных медных провода внутри 1 кабеля
  + Оптический кабель (часто применяется) – тонкие световоды в оболочке, соединённые в 1 кабель
  + Провода электропитания 220V
* **Беспроводные:**
  + Радиоволны (часто) – много направлений, много приёмников, несколько источн. искажают друг друга
    - Сотовая связь – GSM 900MHz, требуется лицензирование (чтобы др. не работали на этой частоте)
    - Wi-Fi – 2.4GHz & 5GHz, лицензирование не требуется (др. приборы также работают на этих частотах)
  + Инфракрасное излучение
* **Спутниковые каналы**
* **Беспроводная оптика (лазеры)**

**? Ошибки в каналах связи**

* Оптические кабели – Очень редко
* Медные кабели – Редко
* Радиоволны – Часто

**? Представление информации**

* Прямоугольные импульсы: Представление инфы – кодирование (baseband mod) (медные провода)
* Синусоидальные волны: Представление инфы – модуляция (passband mod) (оптоволокно, беспров.)

**Уровень 2 – Канальный уровень**

**? Основные задачи Канального уровня:**

* Передача сообщений про каналу связи – кадров
  + Определение начала/конца кадра в потоке бит
* Обнаружение и коррекция ошибок
* Множественный доступ к каналу связи:
  + Адресация
  + Согласованный доступ к каналу

**? Формирование кадра**

Пакет

Пакет

Заголовок

Концевик

**КАДР**

…0111000101011101001000101201…

Пакет

Пакет

Заголовок

Концевик

**КАДР**

…0111000101011101001000101201…

**Хост 1**

**Хост 2**

***Сетевой***

***Канальный***

***Физический***

* Хост-1: Пакет с Сетевого ур-ня поступает на Канальный.
* Хост-1: На канальном ур-не к пакету добавляются заголовок и концевик, кадр поступает на Физический ур.
* Хост-1: На Физическом уровне биты преобразуется в сигнал.
* Сигнал передаётся по среде передачи данных.
* Хост-2: Сигнал поступает на Физический ур-нь 🡪 биты 🡪 преобразуются в кадр и поступают на Канальный.
* Хост-2: На Канальном ур-не от кадра отсекаются заголовок и концевик 🡪 выделяется пакет.
* Хост-2: Пакет поступает на Сетевой ур-нь.

? Методы выделения кадров:

* Указатель количества байт – в начало каждого кадра добавлять его длину в байтах  
  «+» Очень просто в реализации  
  «–» При искажении 1 байте длины в 1 кадре – нарушится вся цепочка: определение начал всех кадров  
  
* Вставка байтов (Протокол BSC) – начало и конец кадра отмечаются спец последовательностями байтов.
  + В начале кадра – буквы «DLE STX» (Start of Text)
  + В конце кадра – буквы «DLE ETX» (End of Text)
  + В самом сообщении, если встречается комбинация STX или ETX, перед ними ставится «DLE» (Data Link Escape), чтобы не путать с началом и концом сообщения
* Вставка битов (Протоколы HDLS и PPP) – начало и конец кадра отмечаются битами
  + В начале кадра – 01111110 (ноль – 6 единиц – ноль)
  + В конце кадра – 01111110 (ноль – 6 единиц – ноль)
  + В самом сообщении, если встречается 5 единиц подряд, к ним добавляется ноль (чтобы не получилось 6 единиц подряд), получатель игнорирует 0 после каждых 5 единиц подряд
* Средства Физического уровня
  + Классический Ethernet – Преамбула общей длиной 8 байт
    - Начало кадра – Первые 7 байт – 10101010
    - Конец кадра – Последний 1 байт – 10101011 (указатель начала нового сообщения)
  + Fast Ethernet – передача неиспользуемых символов избыточного кода
    - Начало кадра – символы «J» (11000) и «K» (10001)
    - Конец кадра – символ «T» (01101)

**? Обнаружение ошибок** – контрольная сумма – кадр, в котором ошибка, просто отсекается, не исправляется

**? Исправление ошибок** – коды исправления ошибок – и обнаруживают и исправляют ошибки

**? Повторная отправка данных** – если в кадре ошибка или кадр вообще не дошёл до получателя – получатель не отправляет подтверждение, отправитель по истечению определённого времени не получив подтверждение, понимает, что кадр не доставился или доставился с ошибкой и отправляет кадр повторно

Отправитель

Получатель

Сообщение

Подтверждение

Подтверждение

Сообщение

Сообщение

🕙

**? Методы повторной отправки данных:**

* Остановка и ожидание:
  + Отправитель передаёт кадр и останавливается
  + Получатель отправляет подтверждение
  + Отправитель передаёт новый кадр
* Скользящее окно (**ТОЛЬКО** **на Транспортном уровне!**)  
  (Размер окна – кол-во кадров, которые можно отправить одновременно):
  + Отправитель передаёт несколько кадров один за другим, не дожидаясь подтверждения
  + Получатель подтверждает получение кадров
  + Отправитель передаёт новую порцию кадров

**? Множественный доступ к каналам**

Модель OSI разрабатывалась для каналов точка-точка (последовательные линии связи для больших компов)

Когда получили распространение разделяемые каналы связи –пришлось разбить Канальный ур-нь на два.

* Канальный Уровень:
  + LLC – Подуровень управления логическим каналом (Logical Link Control)
    - Отвечает за передачу данных – создание кадров, обработка ошибок и т.д.
    - Мультиплексирование – передача данных через одну технологию Канального ур-ня нескольких типов протоколов вышестоящего уровня
    - Управление потоком – предотвращение «затопления» медленного оборудования получателя, быстрым потоком от мощного оборудования отправителя
    - Общий для различных технологий
  + MAC – Подуровень управления доступа к среде (Media Access Control) – Только при технологии с разделяемым доступом, при соединении точка-точка подуровень MAC не нужен
    - Корректное совместное использование разделяемой среды – управление доступом – предотвращение «коллизии» - одновременной передачи данных многими отправителями в одну среду и данные исказятся – обеспечение использование канала только 1 отправителем. Методы управления доступом:
      * Рандомизированный – из N компов выбирается один с вероятностью 1/N (WF, EN)
      * На основе правил использования (Token Ring)
    - Адресация (какому у-ву из множества предназначается информация)
    - Специфический для разных технологий

**Технология Ethernet**

**? Ethernet** – самая популярная в настоящее время технология для создания проводных компьютерных сетей.

Работает на Физическом и Канальном уровне с подуровнями LLC и MAC

**? История Ethernet**

* 1973 – Сеть на разделяемом кабеле придумал Роберт Меткалф, Xerox (The Ether Network, Cable-Tree Ether)
* Xerox, DEC, Intel решают использовать Ethernet в кач-ве сетевого решения для своих компаний (Ethernet II)
* 1982 – создан проект IEEE 802.3 для стандартизации Ethernet
* Конец 1990-ых – Ethernet становится доминирующей технологией в локальных сетях.

**? Типы Ethernet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Скорость** | **Кабель** | **Стандарт IEEE** |
| Ethernet | 10 Mb/s | Толстый, тонкий коаксиал, Витая пара, Оптика | 802.3 |
| Fast Ethernet | 100 Mb/s | Витая пара, Оптика | 802.3u |
| Gigabit Ethernet | 1 Gb/s | Витая пара, Оптика | 802.3z, 802.3ab |
| 5G Ethernet | 2.5 Gb/s, 5 Gb/s | Витая пара | 802.3bz |
| 10G Ethernet | 10 Gb/s | Витая пара, Оптика | 802.3ae, 802.3an |
| 100G Ethernet | 40 Gb/s, 100 Gb/s | Оптика | 802.3ba |

**? Две технологии Ethernet:**

* Классический Ethernet
  + Разделяемая среда
  + Ethernet – Gigabit Ethernet
* Коммутируемый Ethernet
  + Точка-точка
  + Появился в Fast Ethernet
  + 10G Ethernet – 100G Ethernet

**? Варианты технологии Ethernet**

Классический Ethernet (появился самым первым)

* Физическая Топология – общая шина
* Логическая Топология – общая шина
* Технология – коаксиальный кабель и сетевые адаптеры
* «–» – если в каком-то месте происходил разрыв кабеля или повреждение адаптера – переставала работать вся сеть
* «–» – трудно найти место разрыва кабеля

II вариант технологии Ethernet

* Физическая Топология – звезда
* Логическая Топология – общая шина
* Технология – Концентратор (Hub)
* «+» – если в каком-то месте происходил разрыв кабеля – переставала работать часть сети
* «+» – легко найти место разрыва кабеля – по индикатору на Хабе

**? Физический и Канальный уровни Ethernet**

* Физический уровень Ethernet – технология Ethernet содержит описание по разным типам кабелей:
  + Коаксиальный
  + Витая пара
  + Оптоволокно
* Канальный уровень Ethernet – содержатся методы доступа и протоколы, одинаковые для любых кабелей  
  (В классическом Ethernet смешаны подуровни LLC и MAC)

**? Формат кадра Ethernet**

Стандарты

* Экспериментальная реализация Ethernet в Xerox
* Ethernet II (Ethernet DIX) – фирменный стандарт Ethernet компаний DEC, Intel, Xerox
* IEEE 802.3 – юридический стандарт Ethernet принимался очень долго и стандарт Ethernet II успел широко распространится; отличаются незначительно; поддерживаются оба стандарта: Ethernet II и IEEE 802.3

Стандарт кадра Ethernet II

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заголовок | | | Данные | Концевик |
| 6 байт | 6 байт | 2 байта | 46–1500 байт | 4 байта |
| Адрес получателя | Адрес отправителя | Тип | Данные | Контрольная сумма |

* Адрес получателя – MAC-адрес получателя
* Адрес отправителя – MAC-адрес отправителя
* Тип – тип протокола следующего уровня – содержится код протокола от которого получены данные:
  + 0800 – Ipv4
  + 86DD – Ipv6
  + 0806 – ARP
* Данные – данные, полученные от протокола вышестоящего ур-ня
  + Min длина 46 байт – продиктована самим Ethernet – контекст коллизий
  + Max длина 1500 байт – произвольно выбрана создателями (при создании это был большой объём)
  + Расширение JumboFrame – размер до 9000 байт
* Контрольная сумма – если она не совпадает – кадр отбрасывается, отправитель никак не уведомляется

Стандарт кадра IEEE 802.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заголовок | | | Данные | Концевик |
| 6 байт | 6 байт | 2 байта | 46–1500 байт | 4 байта |
| Адрес получателя | Адрес отправителя | Длина | Данные | Контрольная сумма |

**MAC-адреса**

**? MAC-адреса**

* Служат для идентификации сетевых интерфейсов узлов сети Ethernet (IEEE 802.3) и Wi-Fi (IEEE 802.11) = для того, что бы на канальном ур-не знать какому именно у-ву предназначается сообщение
* Регламентированы IEEE в группе 802
* Длина – 6 байт = 48 бит
* Форма записи – 6 шестнадцатеричных чисел (1С-75-08-D2-49-45 = 1С:75:08:D2:49:45)
* Должны быть уникальными

**? Типы MAC-адресов**

* Индивидуальный (unicast) (**30-9C-23-15-E8-8C**) – получает 1 компьютер
* Групповой (multicast) – первый бит старшего байта = 1 (**01-80-C2-00-00-08**) – получают компы в группе
* Широковещательный (broadcast) – все биты = 1 (**FF-FF-FF-FF-FF-FF**) – получают все компы в сети

**? Уникальность MAC-адресов**

* В одном сегменте сети не должно быть одинаковых MAC-адресов
  + одна широковещательная среда Ethernet или Wi-Fi
  + один VLAN в коммутируемом Ethernet
* Если будет два компьютера с одним MAC-адресом, то один из них не будет работать (какой именно – не регламентируется)

**? Способы назначения MAC-адресов**

* Централизованный (по умолчанию) (второй бит старшего байта = **0**):
  + Адреса назначаются производителем оборудования (записываются в сетевой адаптер)
  + Правила назначения описываются стандартом IEEE 802
* Локальный (второй бит старшего байта = **1**):
  + Адреса назначаются администратором сети
  + Администратор должен обеспечить уникальность

**? Структура MAC-адресов**

Для того, чтобы реализовать уникальность MAC-адресов во всём мире – вводится структура MAC-адреса:

* Первые 3 байта – OUI, Organizationally Unique Identifier – уникальный идентификатор организации – выдаются IEEE производителям оборудования.   
  Примеры OUI:
  + 00:00:0C – Cisco (ещё 6C:50:4D, 70:81:05 и др.)
  + 00:02:B3 – Intel
  + 00:04:AC – IBM
* Последние 3 байта – назначает производитель оборудования, отвечающий за уникальность

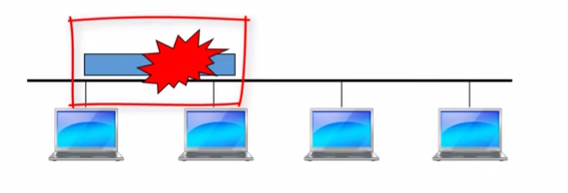
**?Посмотреть MAC-адрес своего компьютера**

* Windows GUI – Просмотр свойств сети
* Windows CLI – ipconfig/all
* Linux – ifconfig или ip link

**Ethernet. Метод доступа CSMA/CD**

**? Коллизия сообщений**

* Один компьютер начал и передаёт данные в разделяемую среду.
* Второй компьютер начал передавать данные, хоть первый комп не закончил передачу.
* Произойдёт коллизия – столкновение сигналов, наложение сигналов друг на друга и потеря сообщения



**? CSMA/CD**

* В сетях с разделённым доступом надо ввести метод управление доступом к среде, чтобы в одно и то же время только один компьютер передавал данные
* В классическом Ethernet используется метод CSMA/CD (Carrier Sense Multiple Access with Collision Detection) – Множественный доступ с прослушиванием несущей частоты и обнаружением коллизий =   
  = Множественный доступ + Прослушивание несущей частоты + Обнаружением коллизий
  + Множественный доступ – разделяемая среда с подключением множества компьютеров
  + Прослушивание несущей частоты – компы передают данные, только если среда свободна; способ определить свободна ли среда – прослушивание основной гармоники сигнала (несущей частоты): все компьютеры смотрят – изменяется ли сигнал с заданной частотой, если изменяется – то значит какой-то компьютер передаёт сообщение, и передавать нельзя в этот момент нельзя
  + Обнаружением коллизий – компьютер одновременно передаёт и тут же принимает собственный сигнал, если принятый сигнал отличается от переданного – значит он был искажён и произошла коллизия; компьютер останавливает передачу и отправляет в сеть Jam-последовательность – это сигнал который сильно искажает все данные, которые передаются по сети, усиливает коллизию, для того, чтобы остальные компьютеры гарантировано поняли, что произошла коллизия и остановили передачу

**? Устройство метода CSMA/CD**

Кадр

Кадр

Кадр

Кадр

Период передачи

Период конкуренции

Период простоя

Работа по этому методу состоит из трёх периодов:

* Период передачи – какому-то компьютеру удалось захватить доступ к среде и он передаёт свои данные
* Период простоя – никому данные передавать не нужно, и среда свободна
* Период конкуренции – несколько компьютеров пытаются передавать данные, возникает коллизия, передача данных останавливается, всё повторяется, пока одному из компов не удастся захватить среду

**? Период передачи**

Если в среде нет несущей частоты, то компьютер может начинать передачу данных

Схема передачи:

* Преамбула – помогает отправителю и получателю синхронизироваться и выделить кадр
  + Преамбула общей длиной 8 байт (Классический Ethernet)
  + Начало кадра – Первые 7 байт – 10101010
  + Конец кадра – Последний 1 байт – 10101011 (указатель начала нового сообщения)
* Передача самого кадра:
  + После окончания преамбулы компьютер начинает передавать кадр
  + Все остальные компьютеры начинают принимать кадр и записывают его в свой буфер
  + Первые 6 байт кадра содержат адрес получателя
  + Компьютер, который узнал свой адрес, продолжает записывать кадр
  + Остальные компьютеры удаляют кадр из буфера
  + Неразборчивый режим – адаптер принимает все кадры не зависимо от MAC-адреса
* Выдерживается пауза – Межкадровый интервал
  + Продолжительность 9,6 с (Классический Ethernet)
  + Нужно для того, чтобы 1 комп не захватил канал монопольно и не стал передавать кадр за кадром
  + Приведение сетевых адаптеров в исходное состояние

Преамбула

Кадр

Межкадровый

интервал

**? Период конкуренции**

После того как компьютер закончил передавать кадр и выдерживается Межкадровый интервал – остальные компьютеры могут попытаться захватить канал – начинается период конкуренции

Схема конкуренции:

* Компьютеры пытаются начать передачу: передают и принимают своё же сообщение
* Происходит коллизия.
* Компьютеры обнаруживают коллизию и выдерживают отсрочку
* Если длина отсрочки будет одинаковая, то компьютеры продолжат попытку передачи одновременно, снова произойдёт коллизия, снова отсрочка, получится зацикленность
* Длительность отсрочки выбирается по определённому алгоритму:  
  Длительность отсрочки = L × 512 битовых интервалов, где  
  Битовый интервал – время между появлениями двух последовательных битов (0,1мкс в классическом Ethernet)  
  L – случайно выбирается из диапазона [0, 2N–1], где N – номер попытки  
  Алгоритм называется – Экспоненциальный двоичный алгоритм отсрочки.  
  Диапазоны L:  
  1 попытка: [0, 1]  
  2 попытка: [0, 3]  
  3 попытка: [0, 7]  
  5 попытка: [0, 31]  
  10 попытка: [0, 1023]
* После 10 попыток интервал не увеличивается
* После 16 попыток передача прекращается
* Экспоненциальный двоичный алгоритм отсрочки хорошо работает при низкой нагрузке:
  + В сети мало компьютеров
  + Компьютеры редко передают данные
* Алгоритм отсрочки плохо работает при высокой нагрузке – коллизии возникают чаще:
  + Растёт число попыток передачи
  + Растёт интервал, из которого выбирается L, и длительность пауз
  + Экспоненциально увеличивается задержка

**? Недостатки классического Ethernet**

* Плохая масштабируемость – сеть неработоспособна при загрузке среды >30%; кол-во компьютеров ≤ 30
* Низкая безопасность – Данные в разделяемой среде доступны всем
* Разное время доставки кадра – Причина – коллизии; плохо для трафика реального времени

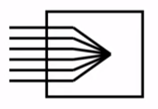
**Коммутируемый Ethernet**

**? Две технологии Ethernet:**

* Классический Ethernet:
  + Исторически появился первым (1973)
  + Разделяемая среда (Общая шина) 🡪 Коллизии
  + Метод CSMA/CD
  + «–»: Плохая масштабируемость, низкая безопасность и т.д.
* Коммутируемый Ethernet:
  + Новая усовершенствованная технология (1995, Fast Ethernet)
  + Нет разделяемой среды (Точка-точка) 🡪 Нет коллизий
  + Новые у-ва – Коммутаторы
  + «+»: Высокая производительность и масштабируемость, высокая безопасность и т.д.

**? Концентратор и коммутатор**

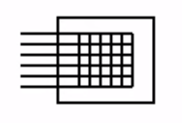
* **Концентратор (Hub)**



Топология – общая шина

Работает на Физическом ур-не – сигнал, поступивший на один порт, передаётся на все порты.

* **Коммутатор (Switch)**



Топология – полносвязная – обеспечивает соединение всех портов напрямую: точка-точка

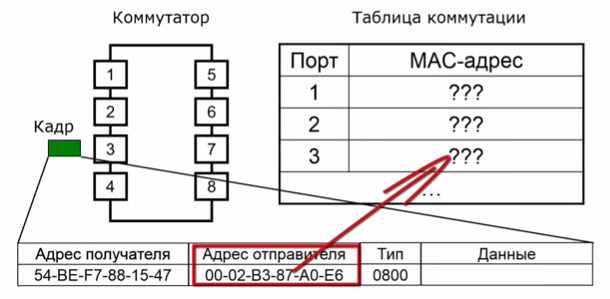
Работает на Канальном ур-не – сигнал, поступивший на один порт, анализируется 🡪 извлекается адрес получателя 🡪 сигнал передаётся только на тот порт, к которому подключен получатель

**? Особенности работы коммутаторов**

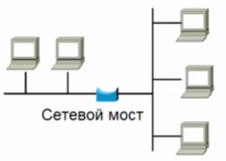
* Таблица коммутации – соответствие MAC-адресов портам коммутатора

|  |  |
| --- | --- |
| Порт Коммутатора | MAC-адрес |
| 1 | 1C-75-08-D2-49-45 |
| 2 | 00-02-B3-A7-49-D1 |
| 3 | 00-04-AC-85-E7-03 |

* Алгоритм обратного обучения – Заполнение таблицы коммутации
  + Коммутатор, только подключился, и ничего не знает о компьютерах, подключенным к его портам 🡪 На порт приходит кадр 🡪 коммутатор извлекает Адрес отправителя 🡪 записывает его в таблицу



* Алгоритм прозрачного моста – Передача кадров коммутатором



* + Мост – у-во для объединения сетей (сеть на десятки компов дробилась на под сети из 2-3 компов,   
    эти подсети соединялись мостами).
  + Прозрачный мост – незаметен для сетевых у-в (у него нет MAC-адреса); не требует настройки.
  + Алгоритм прозрачного моста: кадр пришёл на коммутатор 🡪 он извлёк из заголовка MAC-адрес   
    получателя 🡪 по таблице маршрутизации посмотрел номер порта 🡪 переслал кадр на этот порт.
  + Если в таблице нет такого адреса – коммутатор отработает как концентратор – перешлёт кадр всем

**VLAN**

**? VLAN (Virtual Local Area Network)** – Виртуальная локальная сеть – технология разделения отдельной сети на несколько логических сетей, изолированных друг от друга.

**? Место VLAN в модели OSI** – Канальный ур-нь, коммутаторы

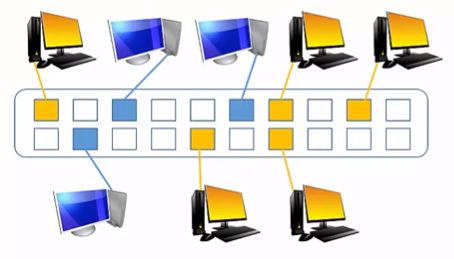
**? Зачем разделять сеть на виртуальные подсети**:

* Разные отделы внутри крупной организации
* Разные компании в бизнес-центре – нельзя создать отдельные физические сети, т.к. неизвестно сколько будет арендаторов

**? Преимущества разделения сети на виртуальные подсети:**

* Безопасность
* Распределение нагрузки
* Ограничение широковещательного трафика – если коммутатор не знает MAC-адреса, он отработает как концентратор – отправит сообщение на все порты – широковещательный трафик – в крупной сети широковещательного трафика, со всеми недостатками, достаточно много

**? Маркировка виртуальных сетей – Нетегированный VLAN**



Разные VLAN отмечают разными цветами. К коммутатору подключены 2 VLAN: синий и жёлтый. Компьютеры отмаркированные жёлтым и подключённые к жёлтым портам могут взаимодействовать только друг с другом, и не могут отправить никакие данные компьютерам из синего VLAN.

В жизни VLAN отмечают порядковыми номерами. Эти номера записываются в таблицу маршрутизации.

Компьютер с порта-1 (VLAN №2 «жёлтый») не сможет отправить кадр на порт-3 (VLAN №3, «синий»), даже если ему известен MAC-адрес получателя, т.к. VLAN у них разные: №2 и №3.

|  |  |  |
| --- | --- | --- |
| Порт Коммутатора | MAC-адрес | VLAN |
| 1 | 1C-75-08-D2-49-45 | 2 |
| 3 | 00-02-B3-A7-49-D1 | 3 |
| 6 | 00-04-AC-85-E7-03 | 3 |
| 7 | 54-BE-F7-88-15-47 | 2 |
| 9 | 00-40-D0-C0-08-BA | 2 |

**? VLAN между коммутаторов – Тегированный VLAN**

VLAN хорошо работает если в сети 1 коммутатор. Но в больших сетях коммутатор может подключаться к другим коммутаторам. Когда кадр передаётся от одного коммутатора другому необходима информация к какому VLAN принадлежит этот кадр. С другой стороны – формат кадра регламентирован стандартом IEEE:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заголовок | | | Данные | Концевик |
| 6 байт | 6 байт | 2 байта | 46–1500 байт | 4 байта |
| Адрес получателя | Адрес отправителя | Тип | Данные | Контрольная сумма |

При этом нельзя просто добавить новое поле «VLAN» - совместимость с существующим оборудованием.

Было внесено изменение формата и введён новый стандарт IEEE 802.1Q

* В поле «Тип кадра», вместо протокола вышестоящего уровня, вставляется специальное значение «0х8100» – указатель, что кадр с VLAN.
* Поле «Данные» увеличивалось на 4 байта – добавлялись 2 поля: «Тег» и «Тип»
* Поле «Тег» – номер VLAN
* Поле «Тип» – код протокола уровня выше (вместо первоначального поля заголовка «Тип»)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Заголовок | | | Данные | | | Концевик |
| 6 байт | 6 байт | 2 байта | 2 байта | 2 байта | 46–1500 байт | 4 байта |
| Адрес получателя | Адрес отправителя | Тип | Тег | Тип | Данные | Контрольная сумма |

* Сетевой адаптер внутри компьютера генерирует обычный Ethernet-кадр, внутри поля заголовка «Тип»: 0х0800 (код протокола следующего ур-ня, 0х0800 = IP-протокол)
* Коммутатор получает кадр, понимает, что он получен с компа MAC-адрес … , входящим в VLAN №2
* Коммутатор добавляет 2 дополнительных поля и изменяет поля:
  + Тип: ~~0х0800~~ 0х8100 – изменяет, старое значение переносит в новое поле «Тип»
  + Тег: 2 – добавляет и проставляет номер VLAN
  + Тип: 0х0800 – код протокола следующего ур-ня переносится из старого поля «Тип» (0х0800=IP)
* Отправляющий Коммутатор передаёт кадр на Принимающий Коммутатор
* Принимающий Коммутатор по значению «0х8100» в поле «Тип» понимает, что кадр содержит информацию о VLAN, понимает что дальше в поле «Тег» информация о номере VLAN, а дальше в поле «Тип» - протокол вышестоящего уровня.
* Принимающий Коммутатор считывает эту информацию и удаляет 2 поля:
  + ~~Тип: 0х8100~~
  + ~~Тег: 2~~
  + Тип: 0х0800
* В таком виде (без 2ух дополнительных полей) принимающий коммутатор передаёт кадр на комп получателя VLAN №2.

Комп-A, VLAN-2 🡪 Коммутатор-A

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заголовок | | | Данные | Концевик |
| Адрес получателя | Адрес отправителя | Тип | Данные | Контрольная сумма |
| … | … | 0х0800 | … | … |

Коммутатор-A

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Заголовок | | | Данные | | | Концевик |
| Адрес получателя | Адрес отправителя | Тип | Тег | Тип | Данные | Контрольная сумма |
| … | … | 0х8100 | 2 | 0х0800 | … | … |

Коммутатор-A 🡪 Коммутатор-B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Заголовок | | | Данные | | | Концевик |
| Адрес получателя | Адрес отправителя | Тип | Тег | Тип | Данные | Контрольная сумма |
| … | … | 0х8100 | 2 | 0х0800 | … | … |

Коммутатор-B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Заголовок | | | Данные | | | Концевик |
| Адрес получателя | Адрес отправителя | Тип | Тег | Тип | Данные | Контрольная сумма |
| … | … | 0х8100 | 2 | 0х0800 | … | … |

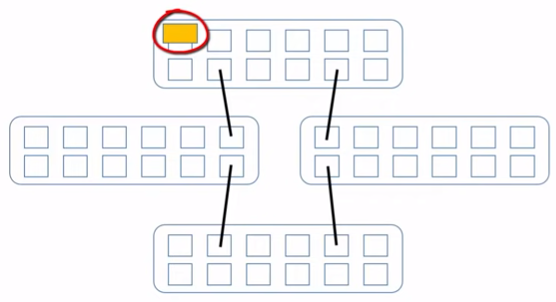
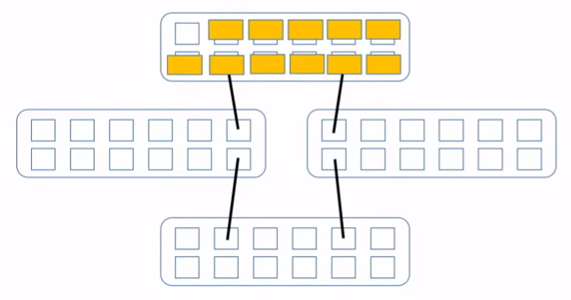
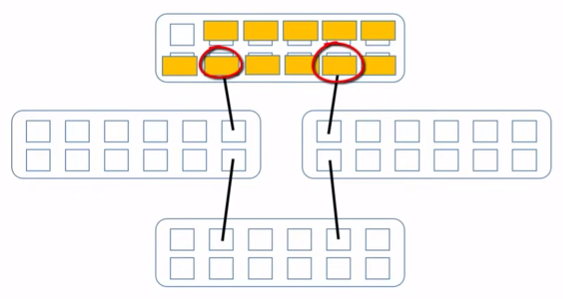
Коммутатор-B 🡪 Комп-B, VLAN-2

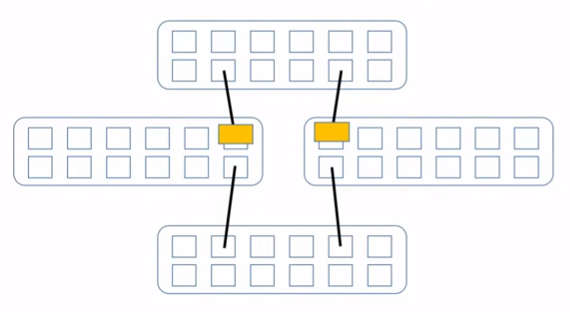
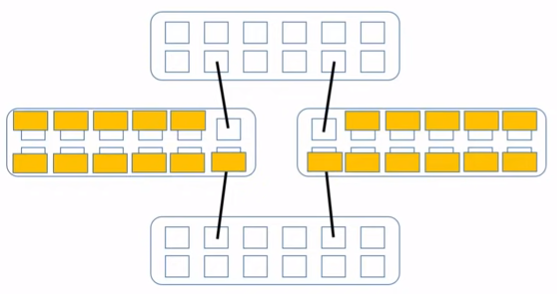
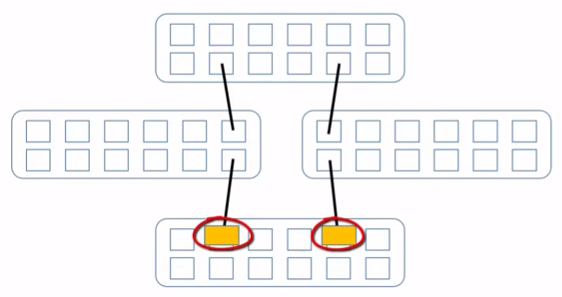
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Заголовок | | | Данные | Концевик |
| Адрес получателя | Адрес отправителя | Тип | Данные | Контрольная сумма |
| … | … | 0х0800 | … | … |

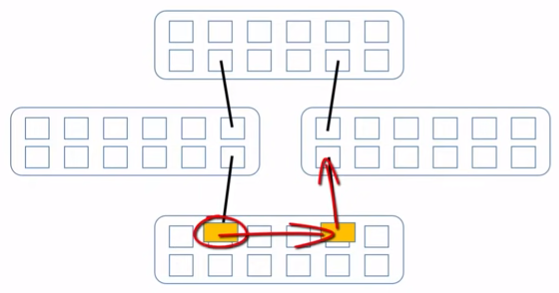
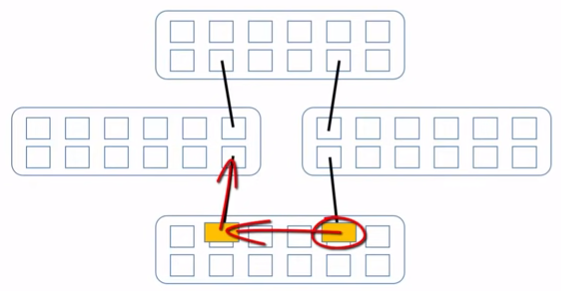
**Протокол STP**

**? Широковещательный шторм**

Соединение коммутаторов в кольцо невозможно – возникает широковещательный шторм – кадры бесконечно дублируются, отправляются в сеть, множатся и сеть выходит из строя (т.к. если коммутатор не знает для кого предназначен кадр, он передаст его на все порты, кроме порта получения, в конечном итоге до конечного коммутатора на 2 разных порта А и В поступят 2 одинаковых кадра А и В, но он не знает, что кадры одинаковы и отправит каждый из них на все порты, кроме полученного, кадр А снова улетит в сеть через порт В, а кадр В через порт А – зацикленность – Широковещательный шторм).

🡪🡪🡪

🡪🡪🡪

🡪

**? STP (Spanning Tree Protocol) – Протокол связующего (остовного) дерева** – Автоматической отключение дублирующих соединений между коммутаторами, чтобы не образовывалось кольцо. Связующее дерево – подграф без циклов, содержащий все вершины исходного графа (Стандарт IEEE 802.1D).

**? STP обеспечивает:**

* Надёжность соединений между коммутаторами – если разорвётся одно соединение – можно исп другое
* Защита от ошибок конфигурации – если случайно подключили коммутатор не в тот порт и образовалось кольцо

**? Этапы работы протокола STP:**

* Выбор корневого коммутатора
* Определение кратчайших путей до корневого коммутатора
* Отключение всех остальных коммутаторов

**? Сообщения протокола STP:**

* Для того чтобы реализовать протокол STP коммутаторы обмениваются друг с другом сообщениями BPDU (Bridge Protocol Data Units)
* BPDU отправляются каждые 2 секунды
* BPDU рассылаются на групповой адрес (**01:80:С2:00:00:00**), который есть у всех коммутаторов, поддерживающих STP

**? Этап I – Выбор корневого коммутатора:**

Выбор по умолчанию – сравнение MAC-адресов – коммутатор с наименьшим MAC-адресом (MAC-адрес – 6 шестнадцатеричных чисел) будет корневым. Но можно настроить вручную «BID – Bridge ID», чтобы сделать корневым коммутатор, не тот у кого ID меньше, а самый мощный. Сперва, коммутаторы подключённые в кольцо не знают ничего о своих 2ух соседях и каждый считает себя корневым. 🡪 Они обмениваются BID с соседями. 🡪 Сравнивают полученный BID со своим. 🡪 Выбирают меньший BID. 🡪 Опять обмениваются сообщениями с 2-мя соседями, но уже указывают не свой BID, а наименьший. 🡪 Таким образом, наименьший BID расходится по кольцу и коммутаторы выбирают корневым коммутатор с этим BID.

**? Этап II – Расчёт кратчайших путей (для разрыва кольца):**

Коммутаторы рассылают на все порты BPDU с минимальным расстоянием до корневого коммутатора. 🡪 Расстояние («Стоимость») определено в стандарте IEEE 802.1D по скорости соединения (4Mb/s=250, 1Gb/s=4, …) 🡪

Коммутаторы определяют суммарное состояние до корневого коммутатора (через своих и других соседей) 🡪 Протокол отключает соединение с наибольшим расстоянием до корневого коммутатора (если расстояния равны, отключится подсоединение в порт с наибольшим номером). 🡪 Кольцо разрывается, получается дерево 🡪 При разрыве соединения Протокол может подключить отключенное соединение, для работы сети.

**? Состояние портов в STP:**

* Listening – порт обрабатывает BPDU, но не передаёт данные
* Learning – порт не передаёт кадры, но изучает MAC-адреса в поступающих кадрах и формирует таблицу
* Forwarding – порт принимает и передаёт кадры данных и BPDU
* Blocking – порт заблокирован, чтобы избежать кольцевого соединения
* Disabled – порт выключен администратором

**? Развитие STP, RSTP:**

Проблемы STP:

1. Переход от состояния «Listening» к «Forwarding» занимает 30 сек, что для современных сетей долго
2. Проблемы при построении сети с несколькими коммутаторами, принадлежащих разным VLAN – соединения между коммутаторами из разных VLAN будут отключены, т.к. STP ничего не знает про VLAN

* Для решения 1ой проблемы – разработан RSTP (Rapid Spanning Tree Protocol) / Стандарт IEEE 802.1w:
* Улучшенная версия STP
* Срабатывает быстрее при подключении оборудования и изменении конфигурации сети (несколько сек)

Для решения 2ой проблемы – разработан MSTP (Multiple Spanning Tree Protocol) / Стандарт IEEE 802.1s:

* Отдельное связующее дерево для каждого VLAN

?

?

?

? **Сокеты / Адреса / Порты**

**Со́кет** (*socket* — разъём) — программный интерфейс для обеспечения обмена данными между [процессами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). Процессы при таком обмене могут исполняться как на одной [ЭВМ](https://ru.wikipedia.org/wiki/%D0%AD%D0%92%D0%9C), так и на различных ЭВМ, связанных между собой [сетью](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). Сокет — [абстрактный](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) объект, представляющий конечную точку соединения

Для взаимодействия между машинами с помощью стека протоколов [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP) используются адреса и порты. Адрес представляет собой 32-битную структуру для протокола [IPv4](https://ru.wikipedia.org/wiki/IPv4), 128-битную для [IPv6](https://ru.wikipedia.org/wiki/IPv6). Номер порта — целое число в диапазоне от 0 до 65535 (для протокола [TCP](https://ru.wikipedia.org/wiki/TCP)).

Эта пара определяет сокет («гнездо», соответствующее [адресу](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%B0%D0%B4%D1%80%D0%B5%D1%81) и [порту](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8)))

**IP-адрес** ([Internet Protocol](https://ru.wikipedia.org/wiki/Internet_Protocol" \o "Internet Protocol)) — уникальный числовой [идентификатор](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80) устройства в компьютерной [сети](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), работающий по протоколу [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP).

**Порт** (*port*) — целое неотрицательное число, записываемое в заголовках [протоколов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [транспортного уровня](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D1%8B%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) сетевой [модели OSI](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI) ([TCP](https://ru.wikipedia.org/wiki/TCP), [UDP](https://ru.wikipedia.org/wiki/UDP), [SCTP](https://ru.wikipedia.org/wiki/SCTP), [DCCP](https://ru.wikipedia.org/wiki/DCCP)).

Обычно на [хосте](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82) под управлением [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) в [пространстве пользователя](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) исполняется несколько процессов, в каждом из которых выполняется какая-либо [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). В случае если несколько программ используют компьютерную [сеть](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), то ОС периодически получает по сети [IP-пакет](https://ru.wikipedia.org/wiki/IP-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82), предназначенный для одной из программ.

**Сетевой шлюз** (*Gateway*) — аппаратный [маршрутизатор](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80) или [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для сопряжения [компьютерных сетей](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8), использующих разные [протоколы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) (например, локальной и глобальной).

Основная задача сетевого шлюза — конвертировать [протокол](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) между сетями.

В сети [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) узлом или конечной точкой может быть или сетевой шлюз, или [хост](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82). Интернет-пользователи и компьютеры, которые доставляют [веб-страницы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) пользователям — это хосты, а узлы между различными сетями — это сетевые шлюзы. Например, [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%B0%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)), контролирующий [трафик](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D1%82%D1%80%D0%B0%D1%84%D0%B8%D0%BA) между локальной сетью компании и сетью Интернет — это сетевой шлюз.

? **Классификация сетей:**

- тип коммутации – коммутация каналов (телефонная сеть) / коммутация пакетов (комп сеть)

- технологии передачи – широковещательная, точка-точка

- протяжённость - персональная, локальная, муниципальная, глобальная, международная

? **Топология комп сетей** – «схема» (граф) соединения компьютеров (вершины – компьютеры, рёбра – соединения между компьютерами).

**? Физическая и Логическая топология**

Физическая топология – как у-ва соединяются между собой

Логическая топология – как сигнал распределяется по сети

? структура заголовка IP

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | № бита |
| 4 бита | | | | 4 бита | | | | 8 бит | | | | | | | | 16 бит | | | | | | | | | | | | | | | |  |
| Номер Версии | | | | Длина Заголовка | | | | Тип Сервиса | | | | PR | D | T | R | Общая длина | | | | | | | | | | | | | | | |
| 16 бит | | | | | | | | | | | | | | | | 3 бита Флаги | | | 13 бит | | | | | | | | | | | | |  |
| Идентификатор Пакета | | | | | | | | | | | | | | | |  | D | M | Смещение Фрагмента | | | | | | | | | | | | |
| 8 бит | | | | | | | | 8 бит | | | | | | | | 16 бит | | | | | | | | | | | | | | | |  |
| Время жизни (TTL) | | | | | | | | Протокол верхнего уровня | | | | | | | | Контрольная сумма | | | | | | | | | | | | | | | |
| 32 бита | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
| IP-Адрес Источника | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 бита | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
| IP-Адрес Назначения | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Параметры и | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
| Выравнивание | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Поле Тип Сервиса**

Бит **PR** – Precedence – приоритет от 0 (мин) до 7 (макс)

Бит **D** – Delay – говорит о том, что маршрут должен выбираться для минимизации задержки доставки данного пакета

Бит **T** – для максимизации пропускной способности

Бит **R** – для максимизации надежности доставки

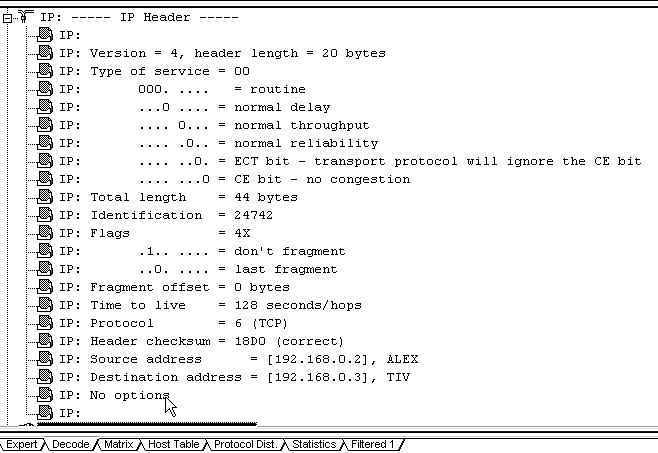
**Поле Флаги (Flags)** занимает 3 бита и содержит признаки, связанные с фрагментацией:

бит **DF (Do not Fragment)** запрещает маршрутизатору фрагментировать данный пакет

бит**MF (More Fragments)** говорит о том, что данный пакет является промежуточным (не последним) фрагментом. Оставшийся бит зарезервирован.

**Поле Смещение фрагмента (Fragment Offset)** занимает 13 бит и задает смещение в байтах поля данных этого пакета от начала общего поля данных исходного пакета, подвергнутого фрагментации. Используется при сборке/разборке фрагментов пакетов при передачах их между сетями с различными величинами [MTU](https://wiki.dieg.info/mtu). Смещение должно быть кратно 8 байт.

**Протокол верхнего ур-ня -** Значения идентификаторов для различных протоколов приводятся в документе [RFC](https://wiki.dieg.info/rfc) 3232 - Assigned Numbers.



**? Список статус кодов HTTP ответа сервера**

Коды ответа ( для тестировщика) – это просто удобное понимание, как именно отреагировал сервер на web или API запрос.

Разделяются на 5 групп:

* 1xx – Info – Информационные (100-105)
* 2xx – Success – Успешные (200-226)
* 3xx – Redirect – Перенаправление (300-307)
* 4xx – Client error – Ошибка клиента (400-499)
* 5xx – Server error – Ошибка сервера (500-510)

**1xx – Info**

100 – Continue

101 – Switching protocols

102 – Processing

103 – Early hints

**2xx – Success**

200 – OK

201 – Created

202 – Accepted – обработка запроса не окончена, ждать не обязательно

203 – Not Authoritative info

204 – No content – нету тела сообщения

205 – Reset content

206 – Partial content

207 – Multi-Status — сервер передаёт результаты выполнения сразу нескольких независимых операций.

**3xx – Redirect**

300 – Multiply choise

301 – Moved perm – документ, новый URI

302 – Found, Moved temp – документ, новый URI

303 – See other – надо запросить по адресу в location GETом, даже если запрашивался другим методом

304 – Not modified

305 – Use proxy

306 – (reserved)

307 – Temp redirect – ресурс, новый URI , метод GET/POST менять нельзя

308 – Perman redirect – ресурс, новый URI , метод GET/POST менять нельзя

**4xx – Client error**

400 – Bad request

401 – Not authorized

402 – (Payment required)

403 – Forbidden

404 – Not found - самая распространённая ошибка при пользовании Интернетом, основная причина — ошибка в написании адреса Web-страницы. Сервер понял запрос, но не нашёл соответствующего ресурса по указанному URL. Если серверу известно, что по этому адресу был документ, то ему желательно использовать код [410](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#410) (~410)*. Ответ 404 может использоваться вместо*[*403*](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#403)*, если требуется тщательно скрыть от посторонних глаз определённые ресурсы.*

405 – Method not allowed – **метод** серверу известен, но **применить нельзя** (~501)

406 – Not Acceptable — запрошенный URI не может удовлетворить переданным в заголовке **характеристикам**. Если метод был не HEAD, то сервер должен вернуть список допустимых характеристик для данного ресурса. Появился в HTTP/1.1.

407 – Proxy Authentication Required — ответ аналогичен коду [401](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP#401) за исключением того, что аутентификация производится для прокси-сервера. Механизм аналогичен идентификации на исходном сервере. Появился в HTTP/1.1.

408 – Request Timeout — время ожидания сервером передачи от клиента истекло. Клиент может повторить аналогичный предыдущему запрос в любое время. Например, такая ситуация может возникнуть при загрузке на сервер объёмного файла методом [POST](https://ru.wikipedia.org/wiki/HTTP#POST) или [PUT](https://ru.wikipedia.org/wiki/HTTP#PUT).

409 – Conflict – запрос не может быть выполнен из-за конфликтного обращения к ресурсу. Такое возможно, например, когда два клиента пытаются изменить ресурс с помощью метода [PUT](https://ru.wikipedia.org/wiki/HTTP#PUT).

410 – Gone – такой ответ сервер посылает, если ресурс раньше был по указанному URL, но был удалён и теперь недоступен. Серверу в этом случае неизвестно и местоположение альтернативного документа (например копии). (~404)

411 – Length required - для указанного ресурса клиент должен указать Content-Length в заголовке запроса. Без указания этого поля не стоит делать повторную попытку запроса к серверу по данному URI. Такой ответ естественен для запросов типа POST и PUT. Например, если по указанному URI производится загрузка файлов, а на сервере стоит ограничение на их объём.

412 Precondition Failed — возвращается, если ни одно из условных полей заголовка запроса не было выполнено.

413 Payload Too Large — возвращается в случае, если сервер отказывается обработать запрос по причине слишком большого размера тела запроса. Сервер может закрыть соединение, чтобы прекратить дальнейшую передачу запроса.

414 URI Too Long — сервер не может обработать запрос из-за слишком длинного указанного URI. Такую ошибку можно спровоцировать, например, когда клиент пытается передать длинные параметры через метод GET, а не POST.

418 – I’m a teapot —код был введен как одна из первоапрельских шуток, не ожидается, что данный код будет поддерживаться реальными серверами

499 Client Closed Request — нестандартный код, предложенный и используемый nginx для случаев, когда клиент закрыл соединение, пока nginx обрабатывал запрос

**5xx – Server error**

500 – Internal Server Error

501 – Not implemented – **метод** серверу **неизвестен** (~405)

502 – Bad Gateway

503 – Service unavailable

504 – Gateway timeout

505 – HTTP ver. not supported

? Заголовки HTTP

1. [General Headers](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%BE%D0%B2_HTTP#%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8) (*Основные заголовки*) — должны включаться в любое сообщение клиента и сервера.
2. [Request Headers](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%BE%D0%B2_HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B0) (*Заголовки запроса*) — используются только в запросах клиента.
3. [Response Headers](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%BE%D0%B2_HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0) (*Заголовки ответа*) — только для ответов от сервера.
4. [Entity Headers](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B7%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%BE%D0%B2_HTTP#%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8_%D1%81%D1%83%D1%89%D0%BD%D0%BE%D1%81%D1%82%D0%B8) (*Заголовки сущности*) — сопровождают каждую [сущность](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D1%89%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) сообщения.

* Host
* User-Agent
* Accept
* Accept-Encoding
* Connection
* Content-Type
* Token
* + Content-Length (POST)
* + Allow (как ответ на 405 и 501)

**? WebDAV (Web Distributed Authoring and Versioning)** или просто **DAV** — набор расширений и дополнений к протоколу [HTTP](https://ru.wikipedia.org/wiki/HTTP), поддерживающих совместную работу пользователей над редактированием файлов и управление файлами на удаленных веб-серверах.

? кэш - **Кэш** (cache) – **это** совокупность временных копий файлов программ, а также специально отведенное место их хранения для оперативного доступа.

? куки - **Cookies** — **это** небольшие текстовые файлы у нас на компьютерах, в которых хранится информация о наших предыдущих действиях на сайтах.

Главное отличие **куки** от **кэша** заключается в том, что каждый раз, когда вы повторно заходите на конкретный сайт, с которого вам был когда-то отправлен конкретный **куки**, веб-клиент (обычно, это ваш веб-браузер) пересылает этот фрагмент данных web-серверу в составе HTTP-запроса.

? в какой части HTTP запроса будут сидеть куки – в заголовке

? виды кукис –

Сессионные cookie. Сессионные cookie, также известные как временные cookie, существуют только во временной памяти, пока пользователь находится на странице веб-сайта. ...

Постоянные cookie. ...

Сторонние cookie. ...

Супер-cookie. ...

Зомби-cookie. ...

Установка cookie. ...

Аутентификация ...

Неточная идентификация

? Отличие куки от сессии

Куки хранятся на клиенте

Сессия – не хранится, это временной промежуток

? разница между кэшированием GET и POST – GET кэшируется, а POST – нет, не кэшируется, не сохраняется в истории браузера.

**? GTPR** – соглашение по которому приложения не могут отправлять личные данные без уведомления и разрешения пользователя.

**? Отличие HTTP/1.1 от HTTP/2**

* Протокол HTTP/2 является бинарным. Изменены способы разбиения данных на фрагменты и транспортирования их между сервером и клиентом.
* В HTTP/2 сервер имеет право послать то содержимое, которое ещё не было запрошено клиентом. Это позволит серверу сразу выслать дополнительные файлы, которые потребуются браузеру для отображения страниц, без необходимости анализа браузером основной страницы и запрашивания необходимых дополнений.
* Также часть улучшений получена за счёт мультиплексирования запросов и ответов, а также за счёт сжатия передаваемых заголовков и введения явной приоритизации запросов.

**Proxy (Charles, Fiddler)**

? Proxy (снифферы трафика):

Charles — инструмент для мониторинга HTTP/HTTPS трафика. Программа работает как прокси-сервер между приложением и сервером этого приложения. Charles записывает и сохраняет все запросы, которые проходят через него и позволяет их редактировать.

?

**Linux**

? Отличие Win от Linux – Безопасность, свобода, бесплатность, открытый код, популярность, количество программного обеспечения, все это основные отличия Linux от Windows

? Разделы команд Linux

• Навигация по каталогам и файлам: cd, ls, pwd.

• Работа с файлами и каталогами: rm, mv, cp, mkdir, cat, more, grep, sort, touch, tail, head, less, find.

• Повышение привилегий: su, sudo.

• Управление правами: chmod, chown, chgrp.

• Текстовые редакторы: vi, vim, nano.

• Архивация и разархивирование: tar, unzip, zip.

• Установка программ: apt, yum.

• Информация о командах: man, опция -h (--help).

• История ранее выполняемых команд: history.

• Работа с сетью: curl, ping, nslookup, netstat, wget, telnet, ifconfig, ip, ss.

• Информация о системе и процессах: top, du, df, ps.

• Управление процессами: kill.

? Команды подробно

cd

(Change directory)

Перейти к папке

clear

Очистить терминал

history

История команд в терминале

man

Справка (полная) в терминале по той или иной команде

whatis

Справка (краткая, 1 строка) в терминале по той или иной команде

sudo

(Substitute User Do)

Передаёт юзеру права Супер Юзера (‘Root’ = Superuser) на выполнение (разово) той или иной команды команды.

su

(Substitute User)

Переключает юзера на другого юзера со всеми правами. Не только на Супер Юзера (‘Root’ = Superuser), но и на любого другого.

(apt – Advanced Packing Tool – программа для установки, обновления и удаления программных пакетов, Настройка системных репозиториев)

sudo apt install

Устанавливает пакет.

sudo apt upgrade

Обновляет пакеты уже установленные в системе.

sudo apt update

Обновляет список пакетов.

sudo apt-add-repository

Добавляет (удаляет) записи в репозиторий Ubuntu: sourses.list APT

sudo apt remove

Удаляет пакет

kill

$ kill -9 SIGTERM PID – делает запрос на остановку программы # (Program ID)

$ kill -9 SIGKILL PID – принудительно останавливает программу # (Program ID)

$ ps ux – показывает список процессов с PID (Program ID)

cat

(concatenate)

Выводит содержимое файла в терминал

chmod

(Change Mode)

Изменяет права доступа к файлу/каталогу: чтение, запись, выполнение.

chown

(Change Owner)

Изменяет владельца файла

file

Узнать тип данных, которые находятся внутри файла

rename

Переименовать множество файлов.

touch

Создание нового пустого файла.

zip/unzip

Добавление/извлечение файлов, директорий в/из архива.

mkdir

(make directory)

Создать новую директорию.

rmdir

(remove directory)

Удалить директорию, ТОЛЬКО если она пустая

rm

(Remove)

Удалить файлы НО! полностью, мимо корзины.

mv

(Move)

Переместить файлы из каталога в каталог ИЛИ переименовать файл в текущем каталоге.

locate

Поиск файлов и каталогов. ЧУВСТВИТЕЛЕН к регистру.

find

Поиск внутри каталога со спец.установками: по имени, расширению, владельцу и т.д.

lsblk

Подробная информация о дисках.

df

(disk free)

Информация о дисках.

free

Информация о оперативной памяти.

uptime

Время работы системы.

whoami

Вывод имени пользователя.

userdel

Удаление пользователяю

passwd

Смена пароля пользователя.

ip address show

Информация об IP адресе

ping

Информация о состоянии соединения.

Работа с сетью: curl, ping, nslookup, netstat, wget, telnet, ifconfig, ip, ss

curl

Это набор библиотек, в которых реализуются базовые возможности работы с URL страницами и передачи файлов. Она отлично подходит для имитации действий пользователя на страницах и других операций с URL адресами.

$ curl опции ссылка

curl https://raw.githubusercontent.com/curl/curl/master/README.md - загрузка файлов

curl -o readme.txt https://raw.githubusercontent.com/curl/curl/master/README.md - запись в файл readme.txt

curl -O https://raw.githubusercontent.com/curl/curl/master/README.md - запись в файл с назв как на сервере

netstat

Сетевые подключения, статистику интерфейсов, таблицы маршрутизации, masquerade, multicast, и т.п..

Перечислить все порты: netstat –a

Перечислить все TCP порты: netstat –at

Перечислить все UDP порты: netstat –au

Перечислить все прослушиваемые порты: netstat –l

Перечислить прослушиваемые TCP порты: netstat –lt

Перечислить прослушиваемые UDP порты: netstat –lu

Перечислить прослушиваемые UNIX сокеты: netstat –lx

Показать статистику всех портов: netstat –s

Показать статистику только TCP портов: netstat –st

Показать статистику только UDP портов: netstat –su

Опция netstat -p добавит «PID/Program Name» в вывод netstat, и может быть совмещена с любым другим набором опций

wget

Скачивание веб-страниц и файлов

$ wget опции аддресс\_ссылки

wget http://ftp.gnu.org/gnu/wget/wget-1.5.3.tar.gz - скачает один файл и сохранит его в текущей директории

wget -O wget.zip http://ftp.gnu.org/gnu/wget/wget-1.5.3.tar.gz - например, скачать файл wget с именем wget.zip

telnet

Telnet - это сетевая утилита, которая позволяет соединиться с удаленным портом любого компьютера и установить интерактивный канал связи, например, для передачи команд или получения информации. Можно сказать, что это универсальный браузер в терминале, который умеет работать со множеством сетевых протоколов.

$ telnet опции хост порт

telnet localhost 123 - проверить доступность порта на узле

telnet opennet.ru 80 - это тестирование сайта из консоли.

ifconfig

Настройки сети - включать или выключать сетевые интерфейсы, настраивать их параметры, переключать режимы и многое другое.

sudo apt install net-tools

$ ifconfig опции интерфейс команда параметры адрес

Параметры и адрес необязательны и зависят от используемой команды. А опции влияют на поведение утилиты более глобально. Опций всего несколько, рассмотрим их

 -a - применять команду ко всем интерфейсам, например, полезно, если вы хотите отключить или включить все сетевые интерфейсы;

 -s - вывести краткий список интерфейсов.

sudo ifconfig – просмотр списка интерфейсов

sudo ifconfig eth0 – просмотр интерфейса из списка

sudo ifconfig –s – просмотр список интерфейсов с минимальной информацией о них

sudo ifconfig –a - вывести все интерфейсы, даже те, которые сейчас отключены

sudo ifconfig eth0 broadcast 192.168.1.255 - установить широковещательный адрес

sudo ifconfig eth0 hw ether BC:AE:C5:BE:8B:B7 - установить аппаратный адрес или так называемый, MAC адрес.

sudo ifconfig eth0 mtu 1000 - изменить максимальный размер пакета

Пример:

$ sudo ifconfig eth0 up

$ sudo ifconfig eth0 add 192.168.1.10

$ sudo ifconfig eth0 netmask 255.255.255.0

$ sudo ifconfig eth0 broadcast 192.168.1.255

$ sudo route add default gw 192.168.1.1

ip

посмотреть сетевые интерфейсы и IP адреса им присвоенные

$ ip [опции] объект команда [параметры

 объект - это тип данных, с которым надо будет работать, например: адреса, устройства, таблица arp, таблица маршрутизации и так далее;

 команды - какое-либо действие с объектом;

 параметры - само собой, командам иногда нужно передавать параметры, они передаются в этом пункте

Теперь давайте рассмотрим самые важные объекты.

 address или a - сетевые адреса.

 link или l - физическое сетевое устройство.

 neighbour или neigh - просмотр и управление ARP.

 route или r - управление маршрутизацией.

 rule или ru - правила маршрутизации.

 tunnel или t - настройка туннелирования

ip a / ip addr show - посмотреть все IP адреса

ip -br a show - Для просмотра информации в кратком виде

ip a show enp0s3 - посмотреть IP адреса только по определённому сетевому интерфейсу

ip a show dev enp0s3 permanen - отобразить только статические IP адреса

ip a show dev enp0s3 dynamic – динамические

ip addr add 10.0.2.100/255.255.255.0 dev enp0s - присвоим тому же интерфейсу enp0s3 IP адрес 10.0.2.100 с маской подсети 255.255.255.0

ip addr del 10.0.2.100/255.255.255.0 dev enp0s3 - удалить IP адрес из интерфейса

ss

Позволяет исследовать сокеты

Ss – отображает только слушающие сокеты

Ss –a – отображает все сокеты

Ss –p - информацию о процессе вместе с другой информацией

Ss –s – небольшая сводка

-t, --tcp Показывает сокеты TCP.

-u, --udp Показывает сокеты UDP.

-d, --dccp Показывает сокеты DCCP.

-w, --raw Показывает сокеты RAW.

-x, --unix Показывает сокеты Unix domain (алиас для -f unix).

-S, --sctp Показывает сокеты SCTP.

--vsock Показывает сокеты vsock (алиас для -f vsock).

? Работа с редактором VIM:

i – (insert) начать редактирование

esc – выйти из режима редактирования

q – (quit) выйти из редактора не сохраняясь

wq – (wright & quit) выйти из редактора сохранив изменения

**Mobile**

? как сервер понимает что зашли с мобилы? – в хедере user-agent; по разным областям в js

? троттлинг – приостанавливаются системы в мобилке при перегреве, пока не остынет

? как снять логи с мобильного

**Крэш-лог (Crash Log)**– файл, в котором хранится вся информация по ошибке неработоспособности/экстренного завершения работы программы.

**Лог-файл (журнал событий, Log)**– это файлы, содержащие системную информацию работы сервера или компьютера, в которые вносятся определенные действия пользователя или программы.

**Снятие логов в Android:**

* Использовать ddms.bat (находится в папке tools — Android sdk).
* Catlog.
* Screens — Power + Громкость.

**Снятие логов в iOS:**

* iTunes.
* Xcode.
* QuickTime Player.
* Organizer — Devices ~ /Library/Logs/CrashReporter/MobileDevice.
* Screens — Home+Power.

? Особенности тестирования мобильных приложений

* Запускается ли приложение на другом устройстве
* Проверить приложение на прерывание звонков
* Ориентация
* Прерывание
* Не стабильное соединение
* Низкий заряд аккумулятора
* Подсоединение провода питания, наушников
* Поведение с подключением и без него
* Отклики на сенсор, мех клавиатуру
* Как ведёт себя приложение при всплыве клавиатуры
* Память и ограничение хранения
* Блокировка экрана
* Смена режима сети 2/3/4/5G/Wi-Fi
* Авиарежим
* Работа приложение с картой памяти
* Работа приложение с сим-картой
* …

## Что основное проверить при тестировании мобильного приложения?

Проверяем в начале тестирования:

* достаточно ли устройств для тестирования (по целевым рынкам) и готово ли покрытие
* согласованы все процедуры на проекте (понять что все готово, выстроили регламент цикл разработки, договорились кто когда вступает в работу, набрана команда)
* аппа соответствует гайдлайнам
* аппа работает в различных окружениях
* аппа должна корректно работать с внешними и внутренними запросами
* Проверка энергопотребления (как типичный юзер)
* Выбор тестов для автоматизации (когда уже написано достаточно ручных)

Список кейсов под iPhone из утерянного источника, довольно неплохой:

* Перепроверка функциональности, где ранее были обнаружены наиболее критические дефекты (регрессионное тестирование)
* Проверка функциональности на корректных данных (текущая дата, короткие имена и т.д.)
* Проверка на некорректных значениях (например: пустые поля, длинные имена, установка на телефоне даты в прошлом и т.д.)
* Проверка интерфейса приложения на соответствие требованиям Apple (Human interface guidelines for iPhone/iPad)
* Производительность приложения и скорость ответа интерфейса (используется iPhone 2g)
* Тесты на удобство пользования приложением (Usability tests)
* Тест на совместимость с другими приложениями/функциональностью iPhone (будильник/таймер/напоминания/входящий звонок/смс)
* Проверка настроек приложения и корректность их применения
* Поиск возможных мест «падения» приложения (crash) и причин их возникновения
* Корректность работы приложения при использовании wi-fi/gprs (включая обрывы связи/ее отсутствие)
* Проверка на корректность работы приложения с памятью iPhone (memory leaks)
* проверка того, что звук не пропадает при подключении наушников
* Поведение приложения при переходе iPhone в спящий режим
* Работа приложения с акселерометром (поворот экрана в соответствии с положением iPhone, использование функции акселерометра для получения данных приложением (шагомер))
* Тестирование локализации (при поддержке приложением)
* Проверка корректности работы приложения с камерой iPhone (если такая функциональность поддерживается), а также корректность работы приложения с iPod.
* быстрые «клики» по элементам интерфейса (переход по категориям, переход по записям внутри категории)
* если есть длительный workflow – проводить его весь (вроде длинных программ в Yoga) в реальном времени
* если есть готовый список и поле для вбивания параметров, то проверить поведение, когда в поле появляется подсказка из словаря и одновременно кликаешь по записи в списке <> подсказке. возможны конфликты между подсказкой айфона и реальным выбором.
* проверка контента: адекватный размер изображений (до 1МБ) и достаточное качество. Дополнительно смотреть на iPhone4 (большее разрешение) + см. MobileHIG.pdf chapter 11 для требований к разрешению изображений.
* GUI: иконки соответствуют тому, к чему относятся (хелп – знак вопроса, настройки – шестеренка и т.д.), новые окна плавно открываются справа, присутствует значок загрузки если происходит длительный процесс)
* Наличие экрана Game Over и корректные ссылки на нем – для игровых проектов (+ корректная отработка попадания на этот экран)

Мультиплеерные игры.

* корректность подключения игроков (напр., списывание баланса только после подключения)
* временные лаги
* подключение через различные сети
* корректное поведение при отключении игроков
* подключение ботов (если используются)

Conformance testing:

* Protocol testing
* Safety/[Security testing](https://www.google.com/url?q=https://www.guru99.com/what-is-security-testing.html&sa=D&source=editors&ust=1613993129610000&usg=AOvVaw2RVuJ4LmXUFdFQ9f2G-yob)
* SIM card testing
* Radio Frequency(RF) testing
* Audio Tests
* Specific Absorption Tests

И еще: Физическая/виртуальная клавиатура, проверка обновления и чистой установки. Платный контент: соответствие цен и содержимого, покупки 2 типов (восстанавливаемые и невосстанавливаемые (кредиты)) - проверка восстановления покупок привязанных к учетке (переустановка/обновление/другое устройство)+должен быть выбор из текущего прогресса и сохраненного в учетке. Реклама: не должна перекрывать элементы, должна иметь доступную кнопку закрытия (А если кнопка еще не появилась, то каунтдаун до этого). Глобализация - меняется все что нужно и это происходит корректно. Защита от получения преимуществ при манипуляциях с датой и временем. Копирование и вставка из/в приложение.

Больше чек-листов и идей можно найти в разделе полезных ресурсов.

? разница между iOS и Android

iOS – упор на контент, нету теней, шрифт San Francisco, используется только на у-вах Apple, таких, как iPhone.

Android – упор на дизайн, есть тени, шрифт Roboto, был основан на Linux и больше похож на ПК.

| **Элемент дизайна** | **iOS** | **Android** |
| --- | --- | --- |
| [Минимальный размер цели нажатия](https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#tap-target) | 44×44 pt | 48×48 dp |
| [Основная навигация приложений](https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#primary-nav) | Панель нижней навигации | Вкладки вверху экрана |
| [Дополнительная навигация приложений](https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#secondary-nav) | Кнопка «More» на нижней панели или интерфейс текущей страницы | Панель нижней навигации или боковое гамбургер-меню |
| [Главная кнопка/действие](https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#primary-actions) | Верхняя навигация, с правой стороны | Floating action button |
| Вторичные действия | Интерфейс текущей страницы | Верхняя навигация, с правой стороны |
| [Список с единственным вариантом выбора](https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#selection) | Список с галочкой для выбранного элемента | Список Radio button |
| Списки множественного выбора | Список с переключателями или список с галочками для выбранных эл-тов | Список с галочками или список с переключателями |
| [Подтверждение или отмена деструктивного действия](https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#undo) | Модальное диалоговое окно подтверждения выбора | Отмена действия посредством временного уведомления на экране |

? Различия Фриз и Краш

Фриз – приложение висит в памяти, но ничего не работает

Краш – приложение вылетело из памяти и нас выкинуло на рабочий стол

? Одна из осн разниц натива от вэба – натив может сделать push-уведомление, а web – нет

? adb – android debug bridge

? pt в iOS и dp в Андроид – разрешение экранов – пикселей и точек на дюйм – кроме названия ничем не отличаются

? Файлы в windows собираются в **exe**, а в iOS, Android-?

iOS – **ipa**

Android – **apk**

? моб приложения надо тестировать на наиболее старом (в рамках разумного) девайсе

? ADB – Android Debug Bridge (ADB) – утилита командной строки для универсального использования, с помощью которой можно открыть доступ к устройствам на Android.

ADB позволяет:

* Осуществлять установку файлов, а также их копирование и удаление;
* Выполнять установку программы на устройство;
* Записывать видео или делать скриншот экрана телефона;
* В случае сбоя работы устройства, выполнить его отладку;
* Изучать логи с телефона;
* Осуществлять прошивку программ и составляющих элементов системы;
* Получать полый доступ к данным об ОС и о самом устройстве

? Взаимодействие Андроида с ядром телефона – Kernel-Linux (ядро Линукс) – Андроид, по сути, просто нахлёбка на Линукс.

?

?

?

?

**Security**

? безопасность –

* конфиденциальность
* целостность
* доступность

**? Алгоритм Диффи-Хеллмана** – алгоритм шифрования SSL, TLS

Протокол Ди́ффи — Хе́ллмана (англ. Diffie–Hellman, DH) — криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования.

Схема открытого распределения ключей снимала основную проблему классической криптографии — проблему распределения ключей.

В чистом виде алгоритм Диффи — Хеллмана уязвим для модификации данных в канале связи, в том числе для атаки «Man-in-the-middle (человек посередине)», поэтому схемы с его использованием применяют дополнительные методы односторонней или двусторонней аутентификации.

Предположим, существует два абонента: Алиса и Боб. Обоим абонентам известны некоторые два числа **g** и **p**, которые не являются секретными и могут быть известны также другим заинтересованным лицам.

Этап I:

Для того, чтобы создать неизвестный более никому секретный ключ, оба абонента генерируют большие случайные числа: Алиса — число **a**, Боб — число **b**.

Затем Алиса вычисляет остаток от деления (1):

(1): **A = ga mod p**

и пересылает его Бобу, а Боб вычисляет остаток от деления (2):

(2): **B = gb mod p**

и передаёт Алисе. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (то есть, у него нет возможности вмешаться в процесс передачи).

Этап II:

На втором этапе Алиса на основе имеющегося у неё **a** и полученного по сети **B** вычисляет значение (3):

(3): **Ba mod p = gab mod p = K**

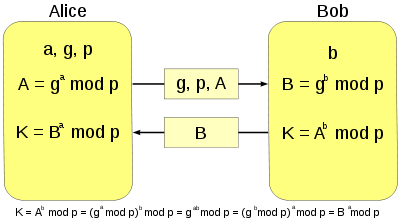
Боб, на основе имеющегося у него **b** и полученного по сети **A** вычисляет значение (4):

(4): **Ab mod p = gab mod p = K**

Как нетрудно видеть, у Алисы и Боба получилось одно и то же число (5):

(5): **K = gab mod p**

Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник встретится с практически неразрешимой (за разумное время) проблемой вычисления (3) или (4) по перехваченным   
**ga mod p** и **gb mod p**, если числа **p**, **a**, **b** выбраны достаточно большими. В практических реализациях для a и b используются числа порядка 10100 и p порядка 10300. Число **g** не обязано быть большим и обычно имеет значение в пределах первого десятка.



**? Сертификаты TLS, SSL**

SSL ( Secure Socket Layer), что означает «уровень защищенных сокетов».

TLS (Transport Layer Security), «безопасность транспортного уровня». ...

Их отличие состоит лишь только в том, что TLS основан на уже действующей спецификации SSL 3.0

**? в PowerShell разрешить доступ к веб ресурсу**, если выходит ошибка: «Invoke-WebRequest: запрос был прерван: не удалось создать безопасный канал SSL / TLS» –

[Net.ServicePointManager]::SecurityProtocol = "tls12, tls11, tls"

**Тест Менеджмент**

?

?

?

?

**Риски Тестирования**

?

?

?

?

**Programming**

? Даны 2 переменные: a и b. Н-р, a=2, b=3. Необходимо поменять местами эти числа (a=3, b=2). Но не используя третью переменную.

|  |  |  |  |
| --- | --- | --- | --- |
| Python | Java | PHP |  |
| a,b = b,a | a=a+b-(b=a) | list($a, $b) = [$b, $a];  a=a+b; b=a-b; a=a-b; |  |

? Компиляция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера), выполняемая [компилятором](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80)

?

?

?

?

**Unsorted**

? На эмуляторе Андроида – скачивается именно Андроид на компьютер – полная имитация

На симуляторе – просто графическая оболочка копирует систему

Для iOS – подходит только симулятор, iOS скачать нельзя на комп

? В DevTool можно поменять свою геолокацию

? Как изменить настройки файлов cookie

Откройте Chrome на компьютере.

В правом верхнем углу экрана нажмите на значок "Настройка и управление Google Chrome" Настройки.

В разделе "Конфиденциальность и безопасность" выберите Файлы cookie и другие данные сайтов.

Выберите один из вариантов: "Показать все файлы cookie";

? XSS-инъекция

XSS (Cross-Site Scripting — «межсайтовый скриптинг») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

? Билд –это версия кот передаётся команде тестирования, а релиз – то что передаётся заказчику

**? Выкинет ли нас из логина если поменяется IP (например, едем в автобусе, меняются моб станции)**

Если поменялся IP, то нас не выкидывает из приложение, т.к. сверка идёт по токену + юзер-агенту

? Social login – логин через соцсети

? При ошибочном вводе логина ИЛИ пароля, не пишут что именно не правильно было введено: неверный логин, не верный пароль; пишется: ошибка при вводе логина ИЛИ пароля – в целях безопасности, чтобы не подсказать злоумышленнику какое именно из двух полей ему удалось преодолеть.