# IT-проект: основные понятия

* + **IT-проект** - это организованная работа, направленная на создание или модификацию информационной системы или программного продукта. Включает в себя этапы планирования, разработки, тестирования и внедрения.

# Компьютерный червь: свойства, особенности

* + **Компьютерный червь** - это вредоносная программа, способная самостоятельно распространяться через компьютерные сети. Особенностью является способность к саморепликации без вмешательства пользователя.

# Компьютерный троян: свойства, особенности

* + **Компьютерный троян** - вредоносная программа, скрывающая свою настоящую природу, маскируясь под полезное ПО. Особенностью является скрытое внедрение и выполнение вредоносных действий без ведома пользователя.

# Компьютерный блокировщик экрана (WinLocker): свойства, особенности

* + **Компьютерный блокировщик экрана** блокирует доступ к компьютеру, часто требуя оплаты для разблокировки. WinLocker - один из примеров. Особенностью является блокировка интерфейса пользователя.

# Необходимые условия применения измерительных мониторов

Необходимые условия применения измерительных методов

1. Наличие готовой программы, подлежащей измерительному исследованию;
2. Наличие реальной вычислительной системы (а не её модели) для прогона программы
3. Наличие аппаратных или программных средств проведения измерений
4. Создание условий снижения искажений, вносимых в функционирование системы в процессе проведения измерений, до приемлемого уровня.

# Измерительные методы: назначение

В целом измерительные методы имеют следующее назначение

1. Измерение параметров потребления программами ресурсов вычислительной системы с целью устранения дефектов производительности.
2. Предварительное измерение параметров системы для имитационных или аналитических моделей программ перед их последующим использованием. Это связано со сложностью оценки параметров моделей, особенно при использовании новых системных средств с неизвестными динамическими параметрами (например устройств ввода-вывода, нового процессора с не вполне известными характеристиками и т.п.).
3. Проверка адекватности имитационных или аналитических моделей и методов расчета характеристик выполнения программ по результатам моделирования.

# Описание компонентов схемы измерений

* + **Схема измерений** включает измерительные приборы, сенсоры, калибровочное оборудование и средства сбора данных. Компоненты схемы обеспечивают точность и надежность измерений.

# Аппаратные измерительные мониторы: виды, основные назначения

* + **Аппаратные измерительные мониторы** - это физические устройства для наблюдения и контроля. Виды включают сенсоры температуры, влажности, давления и т.д. Основное назначение - получение данных о состоянии окружающей среды или устройства.

# Измерительные мониторы: виды, основное назначение

* + **Измерительные мониторы** могут включать в себя и программные решения, такие как мониторы производительности, средства анализа сетевого трафика и др. Основное назначение - наблюдение и контроль за различными аспектами работы системы.

# Статическое тестирование: общие сведения, значение

* + **Статическое тестирование** - это метод проверки кода без его фактического выполнения. Значение включает выявление потенциальных ошибок, улучшение читаемости и структуры кода на ранних этапах разработки.

# Динамическое тестирование: общие сведения, значение

* **Динамическое тестирование** - это метод проверки программного кода при его выполнении. Он включает в себя запуск программы с тестовыми данными для оценки её поведения в реальном времени. Значение включает выявление ошибок, недочетов, оптимизацию производительности и обеспечение корректного функционирования программы в различных условиях.

# Этапы формирования эталонов для тестирования и их описание

* + Этапы:
    1. **Выбор эталонных данных** - определение набора данных, который будет использоваться для проверки правильности работы программы.
    2. **Создание тестовых случаев** - разработка конкретных сценариев использования, которые будут применяться к программе.
    3. **Выполнение тестов** - запуск программы с эталонными данными и анализ результатов.
    4. **Формирование отчетов** - документирование результатов тестирования и выявленных проблем.

# Классификация анализа вредоносных программ

* + Классификация включает:
    - **Статический анализ** - изучение кода без его выполнения.
    - **Динамический анализ** - изучение поведения программы при её выполнении.
    - **Анализ сигнатур** - поиск характерных признаков вредоносных программ.
    - **Поведенческий анализ** - изучение характерного поведения программы.

# Этапы изучения вредоносного программного обеспечения и их описание

* + Этапы:
    1. **Сбор образцов вредоносных программ**.
    2. **Статический анализ кода**.
    3. **Динамический анализ поведения**.
    4. **Идентификация сигнатур**.
    5. **Разработка средств защиты**.

# Основные инструменты для исследования кода программ: виды, свойства, значение

* + Инструменты:
    - **Отладчики** - для анализа и исправления кода.
    - **Декомпиляторы** - для восстановления исходного кода из исполняемого.
    - **Статические анализаторы кода** - для выявления потенциальных ошибок на этапе разработки.
  + Значение: обеспечение качественного анализа и модификации программного кода.

# Файловый монитор (FileMonitor): возможности, предназначение

* + **Файловый монитор** следит за изменениями в файлах и каталогах. Предназначение включает в себя обнаружение и реагирование на несанкционированные изменения, что полезно для обнаружения вредоносных действий.

# Методы, применяемые для обнаружения модифицированного кода. Преимущества и недостатки их применения

* + Методы:
    - **Хеширование** - сравнение хеш-сумм файлов.
    - **Сравнение с эталоном** - сравнение с известным безопасным состоянием.
    - **Анализ сигнатур** - поиск характерных признаков вредоносных программ.
  + Преимущества: эффективность в выявлении модификаций.
  + Недостатки: не всегда эффективны при использовании полиморфных вредоносных программ.

# Понятие отладчика. Назначение, виды

* + **Отладчик** - программа для анализа и исправления кода. Назначение включает отслеживание выполнения программы, выявление ошибок и тестирование. Виды включают **пользовательские отладчики** и **системные отладчики**, вст0роенные в интегрированные среды разработки.

# Отладочный механизм: трассировка кода программы. Назначение, принцип работы

* + **Трассировка кода** - метод отслеживания выполнения программы. Назначение включает в себя выявление ошибок и понимание последовательности выполнения кода. Принцип работы включает запись шагов выполнения программы для последующего анализа.

# Отладочный механизм: контрольные точки останова. Назначение, принцип работы

* + **Контрольные точки останова** - места в коде, где выполнение программы приостанавливается. Назначение включает анализ состояния программы в конкретные моменты времени. Принцип работы включает установку точек останова программистом и ожидание их срабатывания при выполнении программы.

# Понятие тестирования программ. Назначение, методы

* **Тестирование программ** - процесс проверки программного продукта с целью выявления ошибок и уверенности в его корректной работе. Назначение включает обеспечение качества продукта. Методы тестирования могут включать в себя юнит-тестирование, интеграционное тестирование, системное тестирование, и др.

# Понятие анализа программного кода

* **Анализ программного кода** - процесс изучения и понимания кода для выявления ошибок, оптимизации и обеспечения соответствия стандартам. Включает в себя статический и динамический анализ кода.

# Понятие и основные отличия тестирования моделей

* **Тестирование моделей** - проверка моделей на соответствие требованиям и предсказание их поведения в реальных условиях. Отличия включают в себя использование специализированных методов для оценки качества моделей, таких как метрики точности, полноты, и др.

# Характеристика и описание процесса тестирования методом «черный ящик»

* **Метод "черный ящик"** - тестирование, при котором тестировщик не знает внутренней реализации программы. Оценивается только внешнее поведение программы на основе входных и выходных данных.

# Характеристика и описание процесса тестирования методом «стеклянный ящик»

* **Метод "стеклянный ящик"** - тестирование, при котором тестировщик имеет частичное знание о внутренней структуре программы. Оценивается как внешнее, так и внутреннее поведение программы.

# Определение и основные особенности для термина: «эмулирующий отладчик»

* **Эмулирующий отладчик** - это отладочный инструмент, который эмулирует работу целевой аппаратуры или программной среды, позволяя разработчику отлаживать код без фактического наличия целевой платформы. Особенности включают в себя возможность тестирования на различных аппаратных платформах.

# Методы проникновения вредоносного программного обеспечения на персональный компьютер

* Методы включают в себя использование уязвимостей в операционной системе, социальной инженерии, фишинга, вредоносных программ, эксплуатации слабых паролей и др.

# Понятие вредоносного программного обеспечения

* **Вредоносное программное обеспечение (вирус)** - это вредоносная программа, которая спроектирована для нанесения ущерба компьютерной системе, включая разрушение данных, кражу личной информации или привнесение других видов негативных последствий.

# Понятие метрики измерений

1. **Метрика измерений** – это измеримая в конкретные моменты времени величина показателя, который используется в организации для измерения прогресса. Метрика может быть конкретным значением, порогом или диапазоном в рамках различных периодов времени (год, месяц, неделя, день).

# Хранилище (Repository): основные понятия

* **Хранилище (Repository)** – это система управления версиями, предназначенная для хранения, отслеживания и управления изменениями в исходном коде и других ресурсах проекта. Основные понятия включают в себя коммиты, ветки, слияния, откаты и др.

# Рабочая копия (Working Copy): основные понятия.

* **Рабочая копия (Working Copy)** - это локальная копия проекта из системы управления версиями (СКВ), предназначенная для работы над изменениями. В рабочей копии программист вносит изменения, которые затем могут быть зафиксированы и отправлены в основное хранилище.

# Тестирование ПО: общая характеристика, понятия, виды.

* **Тестирование ПО** - процесс проверки программного продукта с целью выявления ошибок, обеспечения его корректной работы и соответствия требованиям. Включает в себя различные виды тестирования, такие как функциональное, нефункциональное, регрессионное, юнит-тестирование и другие.

# Понятия управления контроля версиями, применение и функции системы контроля версий.

* **Система управления версиями (СКВ)** - это инструмент для отслеживания изменений в исходном коде и других ресурсах проекта. Применение включает в себя управление изменениями, работу в команде и восстановление предыдущих версий. Функции включают коммиты, ветки, слияния и откаты.

# Классификация СКВ.

* Классификация включает:
  + **Централизованные СКВ (CVS, SVN)** - с одним центральным репозиторием.
  + **Децентрализованные СКВ (Git, Mercurial)** - каждый пользователь имеет свою локальную копию репозитория.
  + **Распределенные СКВ** - комбинация централизованных и децентрализованных принципов.

# Смысл контроля версий типа SVN. Современные подходы к анализу ПО.

* **SVN (Apache Subversion)** - централизованная система управления версиями. Смысл в обеспечении отслеживания изменений, совместной работы и контроля версий. Современные подходы к анализу ПО включают использование статического и динамического анализа, тестирования, метрик и т.д.

# Уровни тестирования: описание. Цикл тестирования: иерархическое древо: графическое отображение.

* **Уровни тестирования** включают:
  + **Юнит-тестирование** - проверка отдельных компонентов.
  + **Интеграционное тестирование** - проверка взаимодействия компонентов.
  + **Системное тестирование** - проверка всей системы.
  + **Приемочное тестирование** - проверка соответствия требованиям заказчика.
* **Цикл тестирования** представляется в виде иерархического древа, где каждый уровень включает в себя определенные виды тестирования.

# Импорт и экспорт настроек: общие сведения, применение.

* **Импорт и экспорт настроек** - это процесс сохранения и восстановления настроек разработчика в рабочей среде. Применение включает перенос настроек между проектами, машинами, а также обеспечение единых рабочих условий для команды разработчиков.

# Обратное проектирование: основные понятия, виды применяемых инструментов и описание процесса.

* **Обратное проектирование** - это процесс анализа готового программного продукта для выявления его структуры и логики. Инструменты включают декомпиляторы, статические анализаторы кода. Процесс включает в себя анализ исполняемого кода для извлечения высокоуровневой структуры программы.

# Применение и функции обратного проектирования.

* **Обратное проектирование** применяется для понимания кода, восстановления документации, анализа уязвимостей и оптимизации производительности. Функции включают анализ структуры программы, выделение ключевых компонентов и создание документации.

# Описание анализа обратного проектирования.

* **Анализ обратного проектирования** включает в себя изучение скомпилированного или исполняемого кода с целью понимания его структуры, логики и функциональности. Этот процесс помогает разработчикам понять код, созданный другими разработчиками, особенно в случаях, когда нет доступа к исходному коду.

# Модульное тестирование: основные понятия, элементы, результат.

* **Модульное тестирование** - это проверка отдельных модулей или компонентов программы на корректность их работы. Основные понятия включают тестовые случаи, фикстуры, ассерты. Результатом модульного тестирования является убеждение в правильности работы каждого модуля отдельно.

# Интеграционное тестирование: основные понятия, элементы, результат.

* **Интеграционное тестирование** - это проверка взаимодействия между различными модулями или компонентами программы. Основные понятия включают в себя тестовые сценарии, точки интеграции и стабы. Результатом интеграционного тестирования является уверенность в корректности взаимодействия между компонентами.

# Системное тестирование: основные понятия, элементы, результат.

* **Системное тестирование** - это проверка всей системы как единого целого. Основные понятия включают системные тесты, использование приложения в условиях, максимально приближенных к реальным. Результатом системного тестирования является убеждение в том, что вся система работает согласно требованиям.

1. **Локальный репозиторий: определение и функции.**

* **Локальный репозиторий** - это копия удаленного репозитория, расположенная на локальной машине разработчика. Функции локального репозитория включают в себя хранение изменений, возможность работы без подключения к интернету, и облегчение процесса разработки.

# Понятие тестирования методом «серый ящик» и описание процесса.

* **Тестирование методом "серый ящик"** - это подход, при котором тестировщик имеет частичное знание о внутренней структуре программы. Процесс включает в себя проверку функциональности, входных и выходных данных, а также анализ внутренней логики программы.

# Получение доступа к GitHub. Основные функции и принцип GitHub.

* Для получения доступа к GitHub нужно создать аккаунт. Основные функции GitHub включают в себя управление репозиториями, контроль версий, возможность сотрудничества, отслеживание ошибок (issues), запросы на слияние (pull requests) и другие инструменты для разработки.

# Организация разработки в сервисе Github.

* Организация разработки в GitHub включает создание репозиториев, ветвей разработки, управление задачами через issues, обсуждения кода в pull requests, а также использование инструментов для непрерывной интеграции (CI) и других средств совместной разработки.

# Описание уровней доступа GitHub.

* Уровни доступа в GitHub включают:
  + **Владелец (Owner)** - полный доступ ко всему.
  + **Соавтор (Collaborator)** - доступ к определенным репозиториям.
  + **Участник (Contributor)** - право создавать запросы на слияние.
  + **Гость (Guest)** - ограниченный доступ.

# Характеристика способов слияния копий репозитория.

* Способы слияния включают:
  + **Слияние (Merge)** - объединение изменений из одной ветки в другую.
  + **Перебазирование (Rebase)** - перемещение или изменение порядка коммитов в истории.
  + **Слияние с родителем (Squash and Merge)** - объединение коммитов в один перед слиянием.

# Модели создания кода в репозитории: описание и графическое представление.

* Модели создания кода в репозитории включают:
  + **Fork and Pull Request Model** - разветвление репозитория, внесение изменений и создание запроса на слияние.
  + **Branching Model** - использование различных веток для разработки новых функций и исправления ошибок.
  + **Gitflow Model** - использование особых веток для различных этапов разработки (feature, release, hotfix).

# Архитектура MVC: основные понятия, структура.

* **MVC (Model-View-Controller)** - это архитектурный паттерн, используемый в разработке программного обеспечения.
* **Основные понятия:**
  + **Модель (Model)** - представляет данные и бизнес-логику.
  + **Представление (View)** - отвечает за отображение данных пользователю.
  + **Контроллер (Controller)** - обрабатывает пользовательский ввод, обновляет модель и обновляет представление.

# Описание процесса реализации проверки на стороне клиента.

* Проверка на стороне клиента включает в себя валидацию вводимых данных в браузере до их отправки на сервер. Процесс включает создание и привязку сценариев проверки к элементам формы, обработку событий и вывод сообщений об ошибках.

# JavaScript: инструменты языка. Состав среды разработки (IDE).

* **Инструменты JavaScript:**
  + **Браузерные инструменты разработчика** (Chrome DevTools, Firefox Developer Tools).
  + **Среды разработки (IDE)**: Visual Studio Code, WebStorm, Sublime Text.
* **Состав среды разработки:**
  + Редактор кода, отладчик, консоль разработчика, плагины для улучшения работы с JavaScript.

# Описание процесса реализации проверки на стороне клиента. Преимущества проверки на сервере.

* Процесс реализации включает в себя использование JavaScript для валидации данных в браузере. Преимущества включают улучшенный пользовательский опыт, быструю обратную связь, снижение нагрузки на сервер.

# Библиотека классов: понятие, доступ к членам класса.

* **Библиотека классов** - это совокупность классов и методов, предоставляемых языком программирования для облегчения разработки. Доступ к членам класса осуществляется через модификаторы доступа (public, private, protected).

# Язык Ассемблера: возможности языка, преимущества и недостатки.

* Язык Ассемблера предоставляет низкоуровневое представление команд процессора. Преимущества включают точное управление аппаратурой. Недостатки включают сложность программирования и зависимость от аппаратной платформы.

# Сегментная адресация: определение, примеры применения.

* **Сегментная адресация памяти** — схема логической адресации памяти компьютера в архитектуре [x86](https://ru.wikipedia.org/wiki/X86). Линейный адрес конкретной ячейки памяти, который в некоторых режимах работы процессора будет совпадать с физическим адресом, делится на две части: *сегмент* и *смещение*. *Сегментом* называется условно выделенная область адресного пространства определённого размера, а *смещением* — адрес ячейки памяти относительно начала сегмента. *Базой сегмента* называется линейный адрес (адрес относительно всего объёма памяти), который указывает на начало сегмента в адресном пространстве. В результате получается *сегментный (логический) адрес*, который соответствует линейному адресу *база сегмента*+*смещение* и который выставляется процессором на шину адреса.
* Кодовый сегмент отвечает за хранение инструкций программы, сегмент данных — за хранение переменных, стековый сегмент — за управление стеком вызовов. Благодаря разделению памяти на сегменты, процессор может эффективно обрабатывать выполнение программ, обеспечивая доступ к нужным данным и инструкциям.

# Дизассемблер: возможности программы, преимущества и недостатки.

* **Дизассемблер** - это инструмент, преобразующий машинный код обратно в исходный код. Возможности включают в себя анализ исполняемых файлов. Преимущества - возможность изучения или восстановления исходного кода. Недостатки - сложность чтения и понимания кода.

# Мониторинг работоспособности системы контроля версий: назначение и инструменты мониторинга.

* **Мониторинг работоспособности системы контроля версий (СКВ)** включает в себя отслеживание активности, производительности и доступности репозиториев. Инструменты мониторинга могут включать в себя Grafana, Prometheus, а также встроенные средства управления СКВ.

1. **Коллективная разработка: характеристика, структура.**

* **Коллективная разработка** - это методология, при которой группа разработчиков совместно работает над проектом. Структура включает в себя распределение задач, использование систем управления версиями, обмен знаниями и опытом, а также построение эффективного командного взаимодействия.

# Проверка сервера: характеристика, принцип работы, особенности.

* **Характеристика:** Проверка сервера включает в себя мониторинг работоспособности, производительности, наличия неисправностей и обеспечение доступности сервисов.
* **Принцип работы:** Использует различные инструменты и технологии для отслеживания ресурсов, анализа журналов, тестирования соединений и других параметров.
* **Особенности:** Обнаружение и устранение проблем с производительностью, отказоустойчивость, масштабируемость.

# Проверка клиента: характеристика, принцип работы.

* **Характеристика:** Проверка клиента включает в себя оценку корректности отображения веб-страниц, адаптивности к разным устройствам и браузерам.
* **Принцип работы:** Использует инструменты разработчика браузера, тестирование с использованием различных устройств и браузеров.

# Отличительные особенности IDE C++/C#.

* **IDE C++:** Обычно поддерживает функции, специфичные для C++, такие как отладка низкоуровневого кода, анализ памяти, интеграция с компиляторами.
* **IDE C#:** Ориентирована на разработку на платформе .NET, поддерживает функции, специфичные для C#, такие как управление ресурсами .NET и инструменты для разработки веб-приложений.

# Отличительные особенности IDE Java.

* **IDE Java:** Поддерживает функции, специфичные для Java, такие как управление памятью, работа с виртуальной машиной Java, инструменты для разработки многопоточных приложений.

# Отключение JavaScript на стороне клиента: особенности.

* Отключение JavaScript на стороне клиента приводит к потере функциональности, требующей выполнения JavaScript. Это может повлиять на интерактивность веб-страниц, валидацию форм, загрузку динамических данных и другие аспекты веб-приложений.

# Команды git: git init, git branch, git push/pull - характеристика, основные действия.

* **git init:** Инициализирует новый репозиторий.
* **git branch:** Показывает и создает ветки.
* **git push:** Загружает изменения в удаленный репозиторий.
* **git pull:** Загружает изменения из удаленного репозитория.
* **Характеристика:** Используются для управления репозиторием, создания веток и обмена изменениями между локальным и удаленным репозиторием.

# Приемочное тестирование: основные понятия, элементы, результат.

* **Приемочное тестирование:** Проверка соответствия продукта конечным пользователям и их требованиям.
* **Основные понятия:** Тестовые случаи, сценарии, документация требований, отчеты.
* **Элементы:** Тестовые сценарии, проверка требований, валидация функциональности.
* **Результат:** Определение готовности продукта к выпуску на основе выполнения тестовых сценариев.

# Выходное тестирование: основные понятия, элементы, результат.

* **Выходное тестирование:** Тестирование продукта перед его выпуском для обеспечения высокого качества.
* **Основные понятия:** Проверка всех функциональных и нефункциональных требований.
* **Элементы:** Проверка системы в целом, тестирование производительности, тестирование безопасности.
* **Результат:** Убеждение в готовности продукта к публичному использованию.

# Основные элементы для тестирования ПО, результат.

* **Элементы тестирования ПО:** Тестовые случаи, тестовые сценарии, наборы данных, отчеты о тестировании, баг-репорты.
* **Результат тестирования:** Отчеты о выполненных тестах, выявленные ошибки, степень покрытия кода тестами.

# Команды git rebase, git clone, git log: атрибуты, назначение, применение.

* **git rebase:** Переписывает историю коммитов, объединяет или изменяет порядок коммитов.
* **git clone:** Клонирует репозиторий.
* **git log:** Показывает историю коммитов.
* **Атрибуты:** git rebase -i (интерактивный режим), git clone - рекурсивное клонирование подмодулей, git log - различные параметры для настройки вывода.
* **Назначение:** Улучшение истории коммитов, клонирование

Практические задания:

10. Ввести в действие пять ветвей разработок в двух репозиториях, переместить в подкаталог.

11. Переместить Branch1 из исходного репозитория в соседний, добавить изменения и применить команду «Push», и просмотреть по системе контроля версий, дату обновления файла Branch1.

12. Извлечь изменения из удалённого репозитория и применить их к текущей ветви с помощью команды «Git Pull».

13. Синхронизировать финальную версию предлагаемого проекта с репозиторием на GitHub.

14. Объединить ветви «final» и «master» из двух разных репозиториев, представленных удалённо.

15. Разработать в репозитории с помощью функции «menu» две ветви проекта.

16. В программе IDA pro 8.0 дизассемблировать файл «6.exe», находящийся в папке «экзамен» на рабочем столе. Отразить в виде таблицы в MS Word библиотеки, к которым он обращается, и указать, где начинается работа программы.

17. Создать хранилища (репозитория) SVN и разместить в нем каталог. Выполнить импорт данных (Tortoise SVN).

18. Установить параметры работы среды Visual Studio, соответствующие Web-разработке.

19. В Visual Studio импортировать настройки из файла в папке «Экзамен» на рабочем столе, в который выгружены настройки.

20. В программе IDA pro 8.0 дизассемблировать файл \*\*\*.exe. Отобразить главную функцию работы программы. Для функции Start выделить функции sub,call и push.

21. Создать более двух ветвей в локальном репозитории, и соединить с глобальным репозиторием GitHub и настройкой доступа к каждой ветке.

22. Создать новую ветку СКВ с помощью команды Git Branch, перейти в репозиторий и создать новый текстовый файл.

23. Создать локальный репозиторий, связанный с заданным глобальным репозиторием. Настроить параметры доступа к локальному репозиторию GitHub.

24. Выполнить загрузку изменений выполненных в локальном репозитории в глобальный репозитарий GitHub.

25. Создать репозиторий на сервисе GitHub. Создать несколько типов организаций с различными параметрами доступа.

26. Создать несколько учетных записей для разработчиков. Задать параметры, необходимые для отправки и слияния копии репозитория.

27. Создать приложение MVC в среде Visual Studio, обеспечивающее ввод данных о некотором объекте.

28. Проверьте правильность ввода данных с помощью отрицательного и положительного теста на стороне клиента.

29. Создать сервер (TCP/IP) в среде Visual Studio на языке C#.

30. Реализовать на сервере проверку данных (TCP/IP) в среде Visual Studio на языке C#.

31. Создать файл hello\_world.sln в локальной копии репозитория (который должен был появиться). Реализовать программу Hello world на языке C# используя плохой стиль кода. Например, после заголовочных файлов вставьте строку using namespace. «Запуште» изменения в удалёный репозиторий.

32. Изменить исходный код так, чтобы программа через стандартный поток ввода переменных запрашивала имя пользователя. А в стандартный поток вывода выводилось сообщение Hello world from @name, где @name имя пользователя. Провести «коммит» новой версии программы.

33. Проверить, что история коммитов доступна в удалённом репозитории.

34. В локальной копии репозитория создать локальную ветку patch1. Commit, push локальную ветку в удалённый репозиторий. Проверить, что ветка patch1 доступна в удалённом репозитории.

35. В удалённом репозитории выполнить слияние PR patch1 -> master и удалить ветку patch1 в удаленном репозитории. Локально выполнить pull. С помощью команды git log просмотреть историю в локальной версии ветки master.