Практическая работа №1

Тема: Git - Система управления версиями

Цель: Получение знаний про системы контроля выходным кодом проекта.

1. Теоретические сведения.

1.1. Система контроля версий (Version Control System VCS) - это программное обеспечение, которое помогает разработчикам программного обеспечения работатьвместе и поддерживать полную историю своей работы.

Ниже перечислены функции VCS:

- Позволяет разработчикам работать одновременно.
- Не позволяет перезаписывать изменения друг друга.
- Поддерживает историю каждой версии.

Ниже приведены типы VCS:

- Централизованная система контроля версий (Centralized VCS - CVCS).
- Распределенная / децентрализованная система контроля версий (Distributed/Decentralized VCS - DVCS).

Сосредоточимся только на распределенной системе управления версиями и особенно наGit.

1.2. Распределенная система контроля версий

Централизованная система контроля версий (CVCS) использует центральный сервер для хранения всех файлов и позволяет совместную работу. Но основным недостатком CVCS

является его единственная точка отказа, то есть отказ центрального сервера. К сожалению, если центральный сервер падает на час, то в течение этого часа никто не может взаимодействовать. В худшем случае, если диск центрального сервера поврежден и

надлежащее резервное копирование не было выполнено, вы потеряете всю историю проекта. Здесь в систему входит система управления распределенной версией (DVCS).

Репозиторий, или проект Git, охватывает весь набор файлов и папок, связанных с проектом, а также историю изменений каждого файла. История файлов отображается как моментальные снимки во времени, называемые *коммитами*, а коммиты существуют как отношения связанных списков и могут быть организованы в *ветви* . Поскольку Git является DVCS, репозитории являются автономными единицами, и каждый, кто владеет копией репозитория, может получить доступ ко всей базе кода и к своей истории.

Используя командную строку или другие легкодоступные интерфейсы, репозиторий git также позволяет: взаимодействовать с историей, клонировать, создавать ветви, коммитить, объединять, сравнивать изменения между версиями кода и многое другое.

С помощью таких платформ, как GitHub, **BitBucket, SourceForge**, Git также предоставляет больше возможностей для прозрачности проекта и совместной работы. Публичные хранилища помогают командам работать вместе, чтобы создать наилучший конечный продукт.

1.3. Основные команды Git

Для использования Git разработчики используют определенные команды для копирования, создания, изменения и комбинирования кода. Эти команды могут быть выполнены непосредственно из командной строки или с помощью приложения, такого как GitHub Desktop или Git Kraken. Вот несколько общих команд для использования Git:

- git init инициализирует новый репозиторий Git и начинает отслеживать существующий каталог. Он добавляет скрытую подпапку в существующую директорию, в которой размещается внутренняя структура данных, необходимаядля контроля версий.
- git clone создает локальную копию проекта, который уже существует удаленно.Клон включает в себя все файлы проекта, историю и ветви.
- git add этапы изменения. Git отслеживает изменения в коде проекта разработчика, но необходимо создать и сделать снимок изменений(snapshot), чтобы включить их в историю проекта. Эта команда выполняет первый этап двухэтапного процесса. Любые изменения, которые будут поставлены, станут частью следующего моментального снимка и части истории проекта. Внесение изменений в два этапа дает разработчикам полный контроль над историей своего проекта.
- git commit сохраняет моментальный снимок в истории проекта и завершает процесс отслеживания изменений. Короче говоря, коммит функционирует подобно съемке. Все, что было поставлено с git add , станет частью моментального снимкас git commit.
- git statusпоказывает статус изменений
- git branchпоказывает ветви, обрабатываемые локально. Создает ветку.
- git merge объединяет линии развития вместе. Эта команда обычно используется для объединения изменений, выполненных на двух разных ветвях.
- git pull обновляет локальную линию разработки обновлениями от своего удаленного репозитория.
- git push обновляет удаленный репозиторий любыми коммитами, сделаннымилокально.

Пример создания нового репозитория в командной строке Git Bash и добавлениеизменений на GitHub

1. создание и инициализация нового локального репозитория git init myRepo
2. переход в дирректорию `myRepo`

cd myRepo

3. создание нового файла в проекте

touch README.md

4. добавление изменений в репозиторий

git add README.md

5. фиксация изменений с добавлением комментария
git commit -m "add README to initial commit"
6.установление пути к вашему репозиторию, созданному на github
git remote add origin https://github.com/'yousername'/'repositoryname'.git
7. Перенести изменения на на github
git push origin master

Задание

В качестве системы контроля версий предлагается GIT на базе Github.

Задание 1. Создание репозитория.

1.1. Создать аккаунт на GitHub.
1.2. Создать публичный репозиторий на сервере.
1.3. Создать локальный репозиторий .
1.4. Добавить файл1.
1.5. Перенести изменения на GitHub
1.6. Прислать ссылку на репозиторий на почту преподавателя (rabota.apt@mail.ru)

Задание 2. Клонирование репозитория.

2.1. На сервере создать еще один публичный репозиторий
2.2. Добавить в него файл2.
2.3. Сделать клонирование в локальный репозиторий

Задание 3. Работа с ветками.

3.1. В локальном репозитории добавить еще одну ветку.
3.2. В новой ветке произвести изменения.
3.3. Слить ветки
3.4. Вывести статус репозитория
3.5. Отправить изменения на удаленный репозиторий

Отчет должен содержать:

- ТЛ
- Цель
- Протокол основных операций над репозиторием
- Скриншоты результатов операций
- Ссылки на репозитории
- Вывод