Отчет по Лабораторной работе №2

Markdown

Павличенко Родион Андреевич

Содержание

# 1 Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

# 2 Задание

-Сделайте отчёт по предыдущей лабораторной работе в формате Markdown. – В качестве отчёта просьба предоставить отчёты в 3 форматах: pdf, docx и md (в архиве, поскольку он должен содержать скриншоты, Makefile и т.д.)

# 3 Выполнение лабораторной работы

Переходим в режим суперпользователя, устанавливаем пакет git и пакет ghр

|  |
| --- |
| Установка пакетов git и gh |

Рис. 1: Установка пакетов git и gh

Задаем имя и email владельца репозитория (команды 1,2) ,настроим utf-8 (команда 3), зададим имя начальной ветке (команда 4), введем параметры autocrlf и safecrlf (команды 5,6)

|  |
| --- |
| Ввод имени и почты, настройка utf-8, autocrlf,safecrlf |

Рис. 2: Ввод имени и почты, настройка utf-8, autocrlf,safecrlf

Создаем ключ ssh размером 4096 бит Cоздание ключа SSH

Создаем ключ по алгоритму ed25519

|  |
| --- |
| Создание ключа ed25519 |

Рис. 3: Создание ключа ed25519

Генерируем pgp ключ с определенными параметрами

|  |
| --- |
| Генерация pgp ключа |

Рис. 4: Генерация pgp ключа

Копируем PGP-ключ в буфер обмена и переходим на github, вставляем полученный ключ

|  |
| --- |
| Копирование pgp ключа |

Рис. 5: Копирование pgp ключа

Вводим email и указываем git применять его при подписи коммитов, вводим три команды

|  |
| --- |
| Ввод email для git |

Рис. 6: Ввод email для git

При помощи gh авторизируемся

|  |
| --- |
| Авторизация при помощи gh |

Рис. 7: Авторизация при помощи gh

Создаем репозиторий курса на основе шаблона

|  |
| --- |
| Создание репозитория |

Рис. 8: Создание репозитория

|  |
| --- |
| Создание репозитория |

Рис. 9: Создание репозитория

Настраиваем каталог курса, сначала переходим в него (команда 1), далее удаляем лишние файлы (команда 2) и создаем необходимые каталоги (команда3,4)

|  |
| --- |
| Настройка каталога |

Рис. 10: Настройка каталога

Отправляем файлы на сервер

|  |
| --- |
| Отправка файлов на сервер |

Рис. 11: Отправка файлов на сервер

Ответы на контрольные вопросы: 1.Что такое системы контроля версий (VCS) и для решения каких задач они предназначены? Системы контроля версий (Version Control Systems, VCS) — это инструменты, позволяющие отслеживать изменения в коде, документации или любых других файлах, управлять разными версиями проекта и работать в команде без потери данных. Основные задачи: • Хранение истории изменений файлов • Возможность отката к предыдущ • Разрешение конфликтов при совместной работе • Управление параллельной разработкой (ветвление и слияние) • Обеспечение резервного копирования 2.Объяснение понятий VCS и их отношений: Хранилище (репозиторий) — это база данных, содержащая все файлы проекта и историю их изменений. Может быть локальным или удалённым. Commit — фиксация изменений в репозитории. Каждый коммит содержит снимокизменённых файлов и метаданные (автор, дата, комментарий). История — последовательность коммитов, отображающая все изменения в проекте с момента его создания. Рабочая копия — текущая версия проекта на локальном компьютере разработчика, которая может содержать несохранённые изменения. 3.Централизованные и децентрализованные VCS: Централизованные VCS (CVCS) используют единый центральный сервер для хранения всех версий файлов. Разработчики работают с рабочими копиями и получают обновления с сервера. Примеры: SVN (Subversion), Perforce, CVS Децентрализованные VCS (DVCS) — каждый разработчик имеет полную копию репозитория, включая всю историю изменений. Работа возможна без подключения к серверу. Примеры: Git, Mercurial, Fossil 4.Действия при единоличной работе с хранилищем (Git): Создание репозитория: git init Добавление файлов: git add Фиксация изменений: git commit -m “Описание изменений” Просмотр истории: git log Откат к предыдущей версии: git checkout Создание веток и переключение между ними: git branch / git checkout 5.Порядок работы с общим хранилищем (Git + удалённый репозиторий): Клонирование репозитория: git clone Создание новой ветки: git checkout -b feature\_branch Добавление изменений: git add . Фиксация изменений: git commit -m “Описание изменений” Отправка изменений в удалённый репозиторий: git push origin feature\_branch Обновление локальной версии: git pull Слияние изменений: git merge feature\_branch 6.Основные задачи, решаемые Git: Отслеживание изменений в файлах Ведение параллельных разработок с помощьюСлияние и разрешение конфликтов Работа с локальными и удалёнными репозиториями Откат к предыдущим версиям проекта 7.Основные команды Git и их краткая характеристика: git init – создание нового локального репозитория git clone – клонирование удалённого репозитория git add – добавление файлов в индекс для следующего commit git commit -m “сообщение” – создание commit’а git status – проверка состояния репозитория git log – просмотр истории commit’ов git branch – просмотр и создание веток git checkout – переключение на другую ветку git merge – слияние веток git push – отправка изменений в удалённый репозиторий git pull – получение изменений из удалённого репозитория git rebase – переписывание истории commit’ git stash – временное сохранение изменений без commit 8.Примеры использования Git с локальными и удалёнными репозиториями: Локальная работа: git init git add . git commit -m “Первый коммит” Работа с удалённым репозиторием: git clone https://github.com/user/repo.git git pull origin main git push origin main 9.Что такое ветви (branches) и зач Ветви позволяют разрабатывать новые функции и исправлять ошибки параллельно, не изменяя основную версию кода. После завершения работы изменения объединяются с основной веткой (обычно main или master). Пример создания и объединения ветки:git checkout -b new-feature работа с кодом… git commit -am “Добавлена новая фича” git checkout main git merge new-feature git branch -d new-feature 10. Как и зачем игнорировать файлы при commit? Некоторые файлы (например, логи, кэш, временные файлы, конфиденциальные данные) не должны попадать в репозиторий. Для их игнорирования используется файл .gitignore. Пример .gitignore: \*.log node\_modules/ .env Файл .gitignore предотвращает случайное добавление нежелательных файлов в репозиторий.

# 4 Выводы

Мы изучили идеологию и применение средств контроля версий. Освоили умения по работе с git.

# Список литературы