Отчёта по лабораторной работе № 2

Управление версиями

Ортега Вероника

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

- Создать базовую конфигурацию для работы с git.
- Создать ключ SSH.
- Создать ключ PGP.
- Настроить подписи git.
- Зарегистрироваться на Github.
- Создать локальный каталог для выполнения заданий по предмету.

3 Теоретическое введение

3.1 Системы контроля версий. Общие понятия

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется. В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными

участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом. Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить. В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным. Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

3.2 Основные команды git

Наиболее часто используемые команды git: - создание основного дерева репозитория: git init - получение обновлений (изменений) текущего дерева из центрального репозитория: git pull - отправка всех произведённых изменений локального дерева в центральный репозиторий: git push - просмотр списка изменённых файлов в текущей директории git status - просмотр текущих изменения: git diff - добавить все изменённые и/или созданные файлы и/или каталоги: git add . - добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена_файлов - удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm име-

на_файлов - сохранить все добавленные изменения и все изменённые файлы: git commit -am ‘Описание коммита’ - сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit - создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки - переключение на некоторую ветку(при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой): git checkout имя_ветки - отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки - слияние ветки с текущим деревом: git merge –no-ff имя_ветки - удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки - принудительное удаление локальной ветки: git branch -D имя_ветки - удаление ветки с центрального репозитория: git push origin :имя_ветки Работа с локальным репозиторием Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория: git config –global user.name “Имя Фамилия” git config –global user.email “work@mail” и настроив utf-8 в выводе сообщений git: git config –global quotepath false Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке: cd mkdir tutorial cd tutorial git init После это в каталоге tutorial появится каталог .git, в котором будет храниться история изменений. Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий: echo ‘hello world’ > hello.txt git add hello.txt git commit -am ‘Новый файл’ Воспользуемся командой status для просмотра изменений в рабочем каталоге,сделанных с момента последней ревизии: git status

4 Выполнение лабораторной работы

Вначале я перешла по ссылке https://github.com и создала учетную запись nagithub. После подтверждения кодом, который пришел на электронную почту, я настроила основные данные аккаунта. После я перешла в терминал и установку программного обеспечения. Вначале я установила git-flow в Fedora Linux. Для этого я ввела три команды,так как ПО необходимо проводить вручную. Команды: * wget –no-check-certificate -q https://raw.github.com/petervanderdoes /gitflow/develop/contrib/gitflow-installer.sh * chmod +x gitflow-installer.sh * sudo ./gitflow-installer.sh install stable Далее я провела установку gh в Fedora Linux с помощью команды sudo dnf. install gh. (рис. **??**).(рис. **??**) Установка gh Установка gh

Следующим шагом я совершила базовую настройку git. С помощью двух команд я задала имя и email владельца репозитория: * git config –global user.name “” * git config –global user.email “eliz.parfenowa2003@yandex.ru” Далее я настроила utf-8 в выводе сообщений git, используя команду gitconfig –global core.quotepath false. После этого командой git config –global init.defaultBranch master я задала имя начальной ветки (master). Осталась настройка последних двух параметров. Первый параметр autocrlf я настроила командой git config - global core.autocrlf input, а второй параметр safecrlf – командой git config - global core.safecrlf warn. (рис. **??**) Базовая настройка git

Следующий шаг – создание ключа ssh. Вначале командой ssh-keygen -t rsa- b 4096 я создаю ключ по алгоритму rsa размером 4096 бит, а после командой ssh-keygen -t ed25519 – по алгоритму ed25519.(рис. **??**)(рис. **??**) Создание ключа

ssh

Создание ключа ssh

Следующим шагом я создала ключ pgp. Вначале необходимо было сгенерировать ключ с помощью команды gpg –full generate-key. Далее терминал предложил выбрать мне некоторые опции, и я выбрала их в соответствии с требованиями лабораторной работы. (тип RSA and RSA; размер 4096; не истекающий срок действия). Также я заполнила необходимую личную информацию (имя и адрес электронной почты)(рис. **??**) Создание ключа pgp

Далее созданный ключ я добавила в Github. Для того чтобы это сделать вначале было необходимо создать отпечаток приватного ключа. С помощью команды gpg –list-secret-keys –keyid-format LONG я вывела список ключей и нашла нужный(рис. **??**) Отпечаток приватного ключа

Скопировала его и вставила в данную комнаду gpg –armor –export | xclip -selclip вместо . Это нужно было для того, чтобы скопировать ключ в буфер обмена.(рис. **??**) Копирование ключа в буфер обмена

После я зашла на Github и добавила ключ туда, во вкладку GPG Keys.(рис. **??**) Копирование ключа в буфер обмена

Далее я настроила автоматические подписи коммитов git. Для этого используем введенный email.(рис. **??**) Настройка автоматических подписей коммитов git

После необходимо было настроить gh. Вначале с помощью команды gh auth login я авторизовалась. После ответила на вопросы, заданные в терминале. Он вывел код и запустила страницу браузера, где это необходимо было ввести. После ввода все завершилось успешно.(рис. **??**) Настройка gh

Следующим шагом нужно было создать репозиторий курса на основе представленного в лабораторной работе шаблона. Создание репозитория представляло собой серию вот таких команд: * mkdir -p ~/work/study/2021-2022/“Операционные системы” * cd ~/work/study/2021-2022/“Операционные системы” * gh repo create study_2021-2022_os-intro * git clone –recursive

git@github.com: parfenovaee/study_2021-2022_os-intro.git os-intro Однако, чтобы последняя команда прошла успешно, нужно выполнить дополнительные действия. Я загрузила ssh ключ на Github. Для этого я использовала команду cat ~/.ssh/id_rsa.pub | xclip -sel clip, которая скопировала существующий ключ в буфер обмена После переходим на Github в уже знакомую у , где добавляли GPG ключ, и добавляем SSH ключ в нужное окно.(рис. **??**) Добавление ключа в Github

Теперь все команды по созданию репозитория курса могут быть выполнены-успешно.(рис. **??**) Создание репозитория курса

Последним шагом является настройка каталога курса. Начала я ее с команды ****cd ~/work/study /2021-2022/“Операционные системы”/os-intro, которая позволила перейти в каталог. После я удалила лишние файлы командой rm package.json. Создала необходимые каталоги, используя команду make COURSE=os-intro**. *И отправила файлы на сервер следующими командами :* git add . * git commit -am ‘feat(main): make course structure’ (эта команда требует кодо- вой фразы, которую мы вводили ранее) * git push (рис. **??**)(рис. **??**) Настройка каталога курса Настройка каталога курса

5 Выводы

Мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

6 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

Система контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разра- батываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. 2. Объясните следующие понятия VCS иих отношения: хранилище, commit, история, рабочая копия. Хранилище – репозиторий - место хранения всех версий и служебной информации. Commit -это команда для записи индексированных изменений в репозиторий. История– место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах. Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища. 3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion. Децентрализованные систе- мы – система, в которой каждый пользователь имеет свой вариант репо-

зитория и есть возможность добавлять и забирать изменения из репозиториев. Пример –Git.

4. Опишите действия с VCS при единоличной работе с хранилищем.

В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому .

5. Опишите порядок работы с общим хранилищем VCS.

Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости от того, делает ли конкретный человек правки или нет.

6. Каковы основные задачи, решаемые инструментальным средством git?

У Git две основных задачи: первая — хранить информацию обо всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

7. Назовите и дайте краткую характеристику командам git.

– создание основного дерева репозитория: git init – получение обновлений (изменений) текущего дерева из центрального репозитория: git pull – отправка всех произведённых изменений локального дерева в центральный репозиторий: git push – просмотр списка изменённых файлов в текущей директории: git status – просмотр текущих изменения: git diff – сохранение текущих изменений: – добавить все изменённые и/или созданные файлы и/или каталоги: git add . –добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add – удалить файл и/или каталог из индекса репозитория (при этом файл и/или

каталог остаётся в локальной директории): git rm имена_файлов – сохранить все добавленные изменения и все изменённые файлы: git commit -am ‘Описание коммита’ – сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit – создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки – переключение на некоторую ветку: git checkout имя_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: 1 git push origin имя_ветки – слияние ветки с текущим деревом: 1 git merge –no-ff имя_ветки – удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки – принудительное удаление локальной ветки: git branch -D имя_ветки – удаление ветки с центрального репозитория: git push origin :имя_ветки

8. Приведите примеры использования при работе с локальным и удалённым репозитория

Работа с удаленным репозиторием: git remote – просмотр списка настроенных удаленных репозиториев. Работа с локальным репозиторием: git status - выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки

9. Что такое и зачем могут быть нужны ветви (branches)?

Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

10. Как и зачем можно игнорировать некоторые файлы при commit?

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл . Временно игнорировать изменения в файле можно командой git update-index-assumeunchanged

Список литературы