**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

*Дисциплина: Информационная безопасность*

Студент: Колосова К. А.

Группа: НПИбд-01-20

**МОСКВА**

**2023 г.**

2.1. Цель работы – Изучить идеологию и применение средств контроля версий. – Освоить умения по работе с git. 2.2. 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. 2.3. Указания к лабораторной работе Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями. 22 Лабораторная работа № 2. Управление версиями Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией. 2.3.1. Основные команды git Наиболее часто используемые команды git: – создание основного дерева репозитория: 1 git init – получение обновлений (изменений) текущего дерева из центрального репозитория: 1 git pull – отправка всех произведённых изменений локального дерева в центральный репозиторий: 1 git push – просмотр списка изменённых файлов в текущей директории: 1 git status – просмотр текущих изменения: 1 git diff – сохранение текущих изменений: – добавить все изменённые и/или созданные файлы и/или каталоги: 1 git add . – добавить конкретные изменённые и/или созданные файлы и/или каталоги: 1 git add имена\_файлов – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): 1 git rm имена\_файлов – сохранение добавленных изменений: – сохранить все добавленные изменения и все изменённые файлы: 1 git commit -am 'Описание коммита' – сохранить добавленные изменения с внесением комментария через встроенный редактор: Кулябов Д. С. и др. Операционные системы 23 1 git commit – создание новой ветки, базирующейся на текущей: 1 git checkout -b имя\_ветки – переключение на некоторую ветку: 1 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 ветки: – удаление локальной уже слитой с основным деревом ветки: 1 git branch -d имя\_ветки – принудительное удаление локальной ветки: 1 git branch -D имя\_ветки – удаление ветки с центрального репозитория: 1 git push origin :имя\_ветки 2.3.2. Стандартные процедуры работы при наличии центрального репозитория Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений): 1 git checkout master 2 git pull 3 git checkout -b имя\_ветки Затем можно вносить изменения в локальном дереве и/или ветке. После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории.Для этого необходимо проверить, какие файлы изменились к текущему моменту: 24 Лабораторная работа № 2. Управление версиями 1 git status и при необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий. Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов: 1 git diff Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями: 1 git add … 2 git rm … Если нужно сохранить все изменения в текущем каталоге, то используем: 1 git add . Затем сохраняем изменения, поясняя, что было сделано: 1 git commit -am "Some commit message" и отправляем в центральный репозиторий: 1 git push origin имя\_ветки или 1 git push 2.3.3. Работа с локальным репозиторием Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория: 1 git config --global user.name "Имя Фамилия" 2 git config --global user.email "work@mail" и настроив utf-8 в выводе сообщений git: 1 git config --global quotepath false Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке: Кулябов Д. С. и др. Операционные системы 25 1 cd 2 mkdir tutorial 3 cd tutorial 4 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: 1 echo 'hello world' > hello.txt 2 git add hello.txt 3 git commit -am 'Новый файл' Воспользуемся командой status для просмотра изменений в рабочем каталоге, сделанных с момента последней ревизии: 1 git status Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов: 1 curl -L -s https://www.gitignore.io/api/list Затем скачать шаблон, например, для C и C++ 1 curl -L -s https://www.gitignore.io/api/c >> .gitignore 2 curl -L -s https://www.gitignore.io/api/c++ >> .gitignore 2.3.4. Работа с сервером репозиториев Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый): 1 ssh-keygen -C "Имя Фамилия " Ключи сохраняться в каталоге ~/.ssh/. Существует несколько доступных серверов репозиториев с возможностью бесплатного размещения данных. Например, https://github.com/. Для работы с ним необходимо сначала завести на сайте https://github.com/ учётную запись. Затем необходимо загрузить сгенерённый нами ранее открытый ключ. Для этого зайти на сайт https://github.com/ под своей учётной записью и перейти в меню GitHub setting . После этого выбрать в боковом меню GitHub setting SSH-ключи и нажать кнопку Добавить ключ . Скопировав из локальной консоли ключ в буфер обмена 26 Лабораторная работа № 2. Управление версиями 1 cat ~/.ssh/id\_rsa.pub | xclip -sel clip вставляем ключ в появившееся на сайте поле. После этого можно создать на сайте репозиторий, выбрав в меню Репозитории Создать репозиторий , дать ему название и сделать общедоступным (публичным). Для загрузки репозитория из локального каталога на сервер выполняем следующие команды: 1 git remote add origin 2 ssh://git@github.com//.git 3 git push -u origin master Далее на локальном компьютере можно выполнять стандартные процедуры для работы с git при наличии центрального репозитория. 2.4. Рабочий процесс Gitflow Рабочий процесс Gitflow Workflow. Будем описывать его с использованием пакета gitflow. 2.4.1. Общая информация – Gitflow Workflow опубликована и популяризована Винсентом Дриссеном из компании vie. – Gitflow Workflow предполагает выстраивание строгой модели ветвления с учётом выпуска проекта. – Данная модель отлично подходит для организации рабочего процесса на основе релизов. – Работа по модели Gitflow включает создание отдельной ветки для исправлений ошибок в рабочей среде. – Последовательность действий при работе по модели Gitflow: – Из ветки master создаётся ветка develop. – Из ветки develop создаётся ветка release. – Из ветки develop создаются ветки feature. – Когда работа над веткой feature завершена, она сливается с веткой develop. – Когда работа над веткой релиза release завершена, она сливается в ветки develop и master. – Если в master обнаружена проблема, из master создаётся ветка hotfix. – Когда работа над веткой исправления hotfix завершена, она сливается в ветки develop и master. 2.4.2. Установка программного обеспечения – Для Windows используется пакетный менеджер Chocolatey. Git-flow входит в состав пакета git. 1 choco install git Кулябов Д. С. и др. Операционные системы 27 – Для MacOS используется пакетный менеджер Homebrew. 1 brew install git-flow – Linux – Gentoo 1 emerge dev-vcs/git-flow – Ubuntu 1 apt-get install git-flow 2.4.3. Процесс работы с Gitflow 2.4.3.1. Основные ветки (master) и ветки разработки (develop) Для фиксации истории проекта в рамках этого процесса вместо одной ветки master используются две ветки. В ветке master хранится официальная история релиза, а ветка develop предназначена для объединения всех функций. Кроме того, для удобства рекомендуется присваивать всем коммитам в ветке master номер версии. При использовании библиотеки расширений git-flow нужно инициализировать структуру в существующем репозитории: 1 git flow init Для github параметр Version tag prefix следует установить в v. После этого проверьте, на какой ветке Вы находитесь: 1 git branch 2.4.3.2. Функциональные ветки (feature) Под каждую новую функцию должна быть отведена собственная ветка, которую можно отправлять в центральный репозиторий для создания резервной копии или совместной работы команды. Ветки feature создаются не на основе master, а на основе develop. Когда работа над функцией завершается, соответствующая ветка сливается обратно с веткой develop. Функции не следует отправлять напрямую в ветку master. Как правило, ветки feature создаются на основе последней ветки develop. Создание функциональной ветки Создадим новую функциональную ветку: 1 git flow feature start feature\_branch Далее работаем как обычно. 28 Лабораторная работа № 2. Управление версиями Окончание работы с функциональной веткой По завершении работы над функцией следует объединить ветку feature\_branch с develop: 1 git flow feature finish feature\_branch 2.4.3.3. Ветки выпуска (release) Когда в ветке develop оказывается достаточно функций для выпуска, из ветки develop создаётся ветка release. Создание этой ветки запускает следующий цикл выпуска, и с этого момента новые функции добавить больше нельзя — допускается лишь отладка, создание документации и решение других задач. Когда подготовка релиза завершается, ветка release сливается с master и ей присваивается номер версии. После нужно выполнить слияние с веткой develop, в которой с момента создания ветки релиза могли возникнуть изменения. Благодаря тому, что для подготовки выпусков используется специальная ветка, одна команда может дорабатывать текущий выпуск, в то время как другая команда продолжает работу над функциями для следующего. Создать новую ветку release можно с помощью следующей команды: 1 git flow release start 1.0.0 Для завершения работы на ветке release используются следующие команды: 1 git flow release finish 1.0.0 2.4.3.4. Ветки исправления (hotfix) Ветки поддержки или ветки hotfix используются для быстрого внесения исправлений в рабочие релизы. Они создаются от ветки master. Это единственная ветка, которая должна быть создана непосредственно от master. Как только исправление завершено, ветку следует объединить с master и develop. Ветка master должна быть помечена обновленным номером версии. Наличие специальной ветки для исправления ошибок позволяет команде решать проблемы, не прерывая остальную часть рабочего процесса и не ожидая следующего цикла релиза. Ветку hotfix можно создать с помощью следующих команд: 1 git flow hotfix start hotfix\_branch По завершении работы ветка hotfix объединяется с master и develop: 1 git flow hotfix finish hotfix\_branch Кулябов Д. С. и др. Операционные системы 29 2.5. 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. 2.6. Последовательность выполнения работы 2.6.1. Настройка github 1. Создайте учётную запись на https://github.com. 2. Заполните основные данные на https://github.com. 2.6.2. Установка программного обеспечения 2.6.2.1. Установка git-flow в Fedora Linux – Это программное обеспечение удалено из репозитория. – Необходимо устанавливать его вручную: 1 cd /tmp 2 wget --no-check-certificate -q https://raw.github.com/petervanderdoes ⌋ ↪ /gitflow/develop/contrib/gitflow-installer.sh 3 chmod +x gitflow-installer.sh 4 sudo ./gitflow-installer.sh install stable 2.6.2.2. Установка gh в Fedora Linux 1 sudo dnf install gh 2.6.3. Базовая настройка git – Зададим имя и email владельца репозитория: 1 git config --global user.name "Name Surname" 2 git config --global user.email "work@mail" – Настроим utf-8 в выводе сообщений git: 1 git config --global core.quotepath false – Настройте верификацию и подписание коммитов git. – Зададим имя начальной ветки (будем называть её master): 30 Лабораторная работа № 2. Управление версиями 1 git config --global init.defaultBranch master – Параметр autocrlf: 1 git config --global core.autocrlf input – Параметр safecrlf: 1 git config --global core.safecrlf warn 2.6.4. Создайте ключи ssh – по алгоритму rsa с ключём размером 4096 бит: 1 ssh-keygen -t rsa -b 4096 – по алгоритму ed25519: 1 ssh-keygen -t ed25519 2.6.5. Создайте ключи pgp – Генерируем ключ 1 gpg --full-generate-key – Из предложенных опций выбираем: – тип RSA and RSA; – размер 4096; – выберите срок действия; значение по умолчанию— 0 (срок действия не истекает никогда). – GPG запросит личную информацию, которая сохранится в ключе: – Имя (не менее 5 символов). – Адрес электронной почты. – При вводе email убедитесь, что он соответствует адресу, используемому на GitHub. – Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым. 2.6.6. Добавление PGP ключа в GitHub – Выводим список ключей и копируем отпечаток приватного ключа: 1 gpg --list-secret-keys --keyid-format LONG – Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа. Кулябов Д. С. и др. Операционные системы 31 – Формат строки: sec Алгоритм/Отпечаток\_ключа Дата\_создания [Флаги] [Годен\_до] ID\_ключа – Cкопируйте ваш сгенерированный PGP ключ в буфер обмена: 1 gpg --armor --export | xclip -sel clip – Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку New GPG key и вставьте полученный ключ в поле ввода. 2.6.7. Настройка автоматических подписей коммитов git – Используя введёный email, укажите Git применять его при подписи коммитов: 1 git config --global user.signingkey 2 git config --global commit.gpgsign true 3 git config --global gpg.program $(which gpg2) 2.6.8. Настройка gh – Для начала необходимо авторизоваться 1 gh auth login – Утилита задаст несколько наводящих вопросов. – Авторизоваться можно через броузер. 2.6.9. Шаблон для рабочего пространства – Репозиторий: https://github.com/yamadharma/course-directory-student-template. 2.6.9.1. Сознание репозитория курса на основе шаблона – Необходимо создать шаблон рабочего пространства. – Например, для 2021–2022 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид: 1 mkdir -p ~/work/study/2021-2022/"Операционные системы" 2 cd ~/work/study/2021-2022/"Операционные системы" 3 gh repo create study\_2021-2022\_os-intro ↪ --template=yamadharma/course-directory-student-template --public 4 git clone --recursive ↪ git@github.com:/study\_2021-2022\_os-intro.git os-intro 2.6.9.2. Настройка каталога курса – Перейдите в каталог курса: 32 Лабораторная работа № 2. Управление версиями 1 cd ~/work/study/2021-2022/"Операционные системы"/os-intro – Удалите лишние файлы: 1 rm package.json – Создайте необходимые каталоги: 1 make COURSE=os-intro – Отправьте файлы на сервер: 1 git add . 2 git commit -am 'feat(main): make course structure' 3 git push 2.7. Содержание отчёта 1. Титульный лист с указанием номера лабораторной работы и ФИО студента. 2. Формулировка цели работы. 3. Описание результатов выполнения задания: – скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы; – листинги (исходный код) программ (если они есть); – результаты выполнения программ (текст или снимок экрана в зависимости от задания). 4. Выводы, согласованные с целью работы. 5. Ответы на контрольные вопросы. 2.8. Контрольные вопросы 1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? 2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. 3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. 4. Опишите действия с VCS при единоличной работе с хранилищем. 5. Опишите порядок работы с общим хранилищем VCS. 6. Каковы основные задачи, решаемые инструментальным средством git? 7. Назовите и дайте краткую характеристику командам git. 8. Приведите примеры использования при работе с локальным и удалённым репозиториями. 9. Что такое и зачем могут быть нужны ветви (branches)? 10. Как и зачем можно игнорировать некоторые файлы при commit