Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет Компьютерного Проектирования

Кафедра инженерной психологии и эргономики

ПОЛЬЗОВАТЕЛЬСКИЕ ИНТЕРФЕЙСЫ ИНФОРМАЦИОННЫХ СИСТЕМ

ОТЧЁТ

по практическому занятию №1

Студент гр.110901 Прохоренко А.В.

Проверил Давыдович К.И.

Минск 2023

Цель: изучить функции и команды системы контроля версий Git.

**1. Что такое система контроля версий.**

Система контроля версий (СКВ) - это инструмент, который записывает и отслеживает изменения в файлах или наборе файлов со временем. Она позволяет возвращаться к предыдущим версиям файлов, восстанавливать проекты к их исходному состоянию, а также просматривать и анализировать внесенные изменения. С помощью СКВ можно определить, кто и когда вносил изменения, идентифицировать проблемные моменты и отследить задачи, установленные и выполненные в рамках проекта.

**2. Локальные системы контроля версий.**

Локальные системы контроля версий обычно используют простую базу данных, которая сохраняет информацию о изменениях в файлах, обеспечивая контроль версий. Они помогают предотвратить ошибки, такие как случайное изменение неправильного файла или копирование неправильных файлов. Одной из популярных систем контроля версий является RCS, которая до сих пор широко распространена на многих компьютерах. RCS сохраняет наборы патчей (различий между файлами) на диске в специальном формате, и с их помощью можно восстановить состояние каждого файла на определенный момент времени.

**3. Централизованные системы контроля версий.**

Централизованные системы контроля версий решают проблему совместной работы разработчиков. В таких системах, например, CVS, Subversion и Perforce, существует один центральный сервер, который содержит все версии файлов, и несколько клиентов, которые получают файлы из этого централизованного хранилища. Этот подход имеет ряд преимуществ перед локальными системами контроля версий. Разработчики имеют общее представление о том, над чем работает каждый из них. Администраторы имеют полный контроль над правами доступа и могут легко администрировать централизованные системы контроля версий, в отличие от управления локальными базами данных на каждом клиенте.

Однако у централизованных систем контроля версий есть и недостатки. Если сервер выйдет из строя на некоторое время, то в это время невозможно будет использовать систему контроля версий для сохранения изменений или обмена ими с другими разработчиками. Если жесткий диск, на котором хранится центральная база данных, повреждается и отсутствуют регулярные резервные копии, вся история проекта может быть потеряна, за исключением отдельных копий репозитория, которые сохранены на локальных машинах разработчиков.

**4. Распределенные системы контроля версий.**

В распределенных системах контроля версий, таких как Git, Mercurial, Bazaar или Darcs, клиенты не только загружают снимок файлов на определенный момент времени, а полностью копируют репозиторий. Это означает, что если один из серверов, используемых для обмена данными, выходит из строя, любой клиентский репозиторий может быть скопирован на другой сервер, чтобы продолжить работу. Каждая копия репозитория является полным бэкапом всех данных.

Более того, многие распределенные системы контроля версий позволяют взаимодействовать с несколькими удаленными репозиториями одновременно. Это позволяет вам работать с различными группами людей и применять различные подходы в рамках одного проекта. Вы можете синхронизировать изменения между разными репозиториями, сливать ветви разработки и объединять работу разных команд в единое целое. Это дает большую гибкость и мощность в управлении версиями проекта.

**5. Что такое GIT и основные отличия от других СКВ.**

Git использует уникальный подход к хранению данных и управлению версиями файлов. Вместо того, чтобы хранить список изменений в файлах, Git сохраняет снимки состояний файлов и ссылки на эти снимки. Каждый коммит в Git представляет собой полный снимок всех файлов проекта на определенный момент времени.

Git также использует хеш-суммы SHA-1 для идентификации файлов и содержимого. Каждый файл и даже сама структура каталогов имеют свою уникальную хеш-сумму, которая вычисляется на основе их содержимого. Это позволяет Git быстро определять, изменился ли файл или каталог по сравнению с предыдущей версией.

Основные состояния файлов в Git включают измененные файлы (modified), проиндексированные файлы (staged) и зафиксированные файлы (committed). Рабочая копия представляет текущую версию проекта, область индексирования содержит информацию о файлах, которые будут включены в следующий коммит, а каталог Git хранит все метаданные и объекты, связанные с репозиторием.

Этот подход Git к управлению версиями позволяет выполнять множество операций локально, без необходимости подключения к удаленному серверу. Вы можете просматривать историю проекта, сравнивать изменения и создавать коммиты даже в отсутствие сетевого подключения. Когда у вас появляется возможность подключиться к серверу, вы можете синхронизировать свои изменения с другими репозиториями.

Этот распределенный подход к управлению версиями делает Git очень гибким и мощным инструментом для разработки программного обеспечения и сотрудничества над проектами.

Базовый подход в работе с Git выглядит так:

* Изменяете файлы вашей рабочей копии.
* Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в индекс.
* Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в ваш каталог Git.

Если определённая версия файла есть в каталоге Git, эта версия считается зафиксированной (committed). Если файл был изменён и добавлен в индекс, значит, он индексирован (staged). И если файл был изменён с момента последнего распаковывания из репозитория, но не был добавлен в индекс, он считается изменённым (modified).

**Обзор команд**

1. **git init** — это один из способов начать новый проект с помощью Git.

Чтобы инициализировать репозиторий, Git создает скрытый каталог с именем. В этом каталоге хранятся все объекты и ссылки, которые Git использует и создает как часть истории вашего проекта. Этот скрытый каталог отделяет обычный каталог от репозитория Git.

Распространенные варианты использования и варианты для git init

* git init: Преобразование текущего каталога в репозиторий Git
* git init <directory>: Преобразование каталога по текущему пути в репозиторий Git
* git init --bare: Создайте новый пустой репозиторий (репозиторий, который будет использоваться только в качестве удаленного репозитория, который не будет содержать активной разработки)

1. **Git add** добавляет новые или измененные файлы из рабочего каталога в

промежуточную область Git.

Это важная команда - без нее никто бы никогда ничего не сделал. Иногда может иметь репутацию ненужного шага в развитии. Но на самом деле это важный и мощный инструмент. Позволяет формировать историю, не меняя того, как вы работаете.

Распространенные варианты использования и варианты для git add

* git add <path>: Промежуточный каталог или файл
* git add .: разместить все файлы (не указанные в каталоге) во всем репозитории.gitignore
* git add -p: Интерактивная стадия изменений

1. **Git commit** создает фиксацию, которая похожа на моментальный снимок репозитория.

Эти фиксации являются моментальными снимками всего репозитория в определенное время. Вы должны часто делать новые коммиты, основанные на логических единицах изменения. Со временем коммиты должны рассказывать историю вашего репозитория и то, как он стал таким, какой он есть в настоящее время. Коммиты включают в себя множество метаданных в дополнение к содержимому и сообщению, таких как автор, метка времени и многое другое. Коммиты — это строительные блоки «точек сохранения» в системе управления версиями Git.

Распространенные варианты использования и параметры Git Commit

* git commit: запускает процесс фиксации, но, поскольку он не содержит флага для сообщения, будет открыт текстовый редактор по умолчанию, чтобы вы могли создать сообщение о фиксации. Если вы ничего не настроили, есть большая вероятность, что это будет VI или Vim. (Чтобы выйти, нажмите esc, затем , а затем Enter. :wink:)-m:w
* git commit -m "descriptive commit message": запускает процесс фиксации и позволяет одновременно включить сообщение о фиксации.
* git commit -am "descriptive commit message": помимо включения сообщения о фиксации, этот параметр позволяет пропустить этап промежуточного хранения. При добавлении автоматически будут размещены все файлы, которые уже отслеживаются Git (изменения в файлах, которые вы зафиксировали ранее).-a
* git commit --amend: заменяет последнюю фиксацию новой фиксацией.

1. **Git status** показывает текущее состояние рабочего каталога Git и промежуточной области.

Команда только выводит информацию, она не изменяет коммиты или изменения в вашем локальном репозитории.

Полезной особенностью является то, что он предоставит полезную информацию в зависимости от вашей текущей ситуации. В общем, вы можете рассчитывать на то, что он скажет вам:git status

- Где указывается, будь то ветвь или фиксация (это то, куда вы «выписаны») HEAD

- Если в текущем каталоге есть измененные файлы, которые еще не были зафиксированы

- Если измененные файлы находятся в стадии или нет

- Если ваша текущая локальная ветвь связана с удаленной ветвью, она сообщит вам, отстает или опережает ваша локальная ветвь какими-либо фиксациямиgit status

- Во время конфликтов слияния также сообщит вам, какие именно файлы являются источником конфликта.git status

Распространенные варианты использования и варианты для git status

* git status: Чаще всего используется в форме по умолчанию, это показывает хорошую базу информации
* git status -s: Дайте вывод в коротком формате
* git status -v: Показывает более «подробные» детали, включая текстовые изменения любых незафиксированных файлов.

1. **Git reset** сбрасывает весь индекс или удаляет из него изменения определенного файла, используется для возврата к определенному коммиту, отката изменений, «жесткого» или «мягкого»

«Мягкий» (с ключом --soft) резет оставит нетронутыми ваши индекс и все дерево файлов и директорий проекта, вернется к работе с указанным коммитом. Иными словами, если вы обнаруживаете ошибку в только что совершенном коммите или комментарии к нему, то легко можно исправить ситуацию.

«Жесткий» резет (ключ --hard) — команда, которую следует использовать с осторожностью. git reset --hard вернет дерево проекта и индекс в состояние, соответствующее указанному коммиту, удалив изменения последующих коммитов. Если команда достигнет точки ветвления, удаления коммита не произойдет.

1. **Git log** - разнообразная информация о коммитах в целом.

Иногда требуется получить информацию об истории коммитов; коммитах, изменивших отдельный файл; коммитах за определенный отрезок времени и так далее. Для этих целей используется команда git log.

Простейший пример использования, в котором приводится короткая справка по всем коммитам, коснувшимся активной в настоящий момент ветки

git log

Получает подробную информацию о каждом в виде патчей по файлам из коммитов можно, добавив ключ -p (или -u):

git log -p

Статистика изменения файлов, вроде числа измененных файлов, внесенных в них строк, удаленных файлов вызывается ключом --stat:

git log --stat

За информацию по созданиям, переименованиям и правам доступа файлов отвечает ключ --summary:

git log --summary

Чтобы просмотреть историю отдельного файла, достаточно указать в виде параметра его имя (кстати, в моей старой версии git этот способ не срабатывает, обязательно добавлять " — " перед «README»):

git log README

или, если версия git не совсем свежая:

git log — README

1. **Git branch** – это команда для управления ветками в репозитории Git (создание, перечисление и удаление веток)

Работа с ветками — очень легкая процедура в git, все необходимые механизмы сконцентрированы в одной команде.

Просто перечисляет существующие ветки, отметив активную:

git branch

Создаёт новую ветку new-branch:

git branch new-branch

Удаляет ветку, если та была залита (merged) с разрешением возможных конфликтов в текущую:

git branch -d new-branch

Удаляет ветку в любом случае:

git branch -D new-branch

Переименовывает ветку:

git branch -m new-name-branch

Вывод: изучены функции и команды системы контроля версий Git.