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 Попова Е.С.

Проверил Давыдович К.И.

Минск 2023

Цель: изучить функции и команды системы контроля версий Git.

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

Система контроля версий ­– это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда и многое другое.

**2. Локальные системы контроля версий.**

Локальные системы контроля версий имеют простую базу данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий. Они предотвращают такие ошибки, как случайное изменение не того файла или копирование не тех файлов, которые вы хотели. Одной из популярных систем контроля версий была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

**3. Централизованные системы контроля версий.**

Централизованные системы контроля версий решают такую проблему, как необходимость взаимодействовать с другими разработчиками. Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Такой подход имеет множество преимуществ, особенно перед локальными системами контроля версий: все разработчики проекта в определённой степени знают, чем занимается каждый из них; администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать централизованные системы контроля версий, чем оперировать локальными базами данных на каждом клиенте. Централизованные системы контроля также имеют серьезные минусы. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вся историю проекта будет утеряна, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков.

**4. Распределенные системы контроля версий.**

В распределённых системах контроля версий (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

Более того, многие распределённые системы контроля версий могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта.

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

Git хранит и использует информацию совсем иначе по сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож. Основное отличие Git от любой другой системы контроля версий — это подход к работе со своими данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени. Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, поток снимков. Это очень важное различие между Git и почти любой другой системой контроля версий. Git переосмысливает практически все аспекты контроля версий, которые были скопированы из предыдущего поколения большинством других систем.

Для работы большинства операций в Git достаточно локальных файлов и ресурсов — в основном, системе не нужна никакая информация с других компьютеров в сети. Чтобы посмотреть историю проекта, Git не нужно соединяться с сервером для её получения и отображения — система просто считывает данные напрямую из локальной базы данных. Если необходимо посмотреть изменения, сделанные между текущей версией файла и версией, созданной месяц назад, Git может найти файл месячной давности и локально вычислить изменения, вместо того, чтобы запрашивать удалённый сервер выполнить эту операцию, либо вместо получения старой версии файла с сервера и выполнения операции локально. Если вы находитесь оффлайн, то сможете создавать коммиты без каких-либо проблем. Когда будет возможность подключиться к сети, все изменения можно будет синхронизировать.

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git. Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длиной в 40 шестнадцатеричных символов (0-9 и a-f), она вычисляется на основе содержимого файла или структуры каталога. Git сохраняет все объекты в свою базу данных не по имени, а по хеш-сумме содержимого объекта.

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой системе контроля версий, вы можете потерять или испортить свои изменения, пока они не зафиксированы, но после того, как вы зафиксируете снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

Рабочая копия является снимком одной версии проекта. Эти файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск, для того чтобы их можно было использовать или редактировать.

У Git есть три основных состояния, в которых могут находиться ваши файлы: изменён (modified), индексирован (staged) и зафиксирован (committed). К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Индексированный — это изменённый файл в его текущей версии, отмеченный для включения в следующий коммит. Зафиксированный значит, что файл уже сохранён в вашей локальной базе.

Три основные секции проекта Git: рабочая копия (working tree), область индексирования (staging area) и каталог Git (Git directory). Область индексирования — это файл, обычно находящийся в каталоге Git, в нём содержится информация о том, что попадёт в следующий коммит. Её техническое название на языке Git — «индекс», но фраза «область индексирования» также работает. Каталог Git — это то место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git и это та часть, которая копируется при клонировании репозитория с другого компьютера.

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.