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 практической работе №1

Студентка гр.110902 Глушакова З.А.

Проверил Давыдович К.И.

Минск 2023

1. Что такое система контроля версий?

Контроль версий, также известный как управление исходным кодом, – это практика отслеживания изменений программного кода и управления ими. Система управления контролем (далее VCS) – это система, записывающая изменения в файл или набор файлов в течение определенного времени и позволяющая позже вернуться к определенной версии. Она возвращает исходные данные к состоянию, в котором они были внесены изменения, позволяет вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последним что-то менял и вызвал проблему, кто поставил задачу и когда и многое другое. Использование VCS также означает, что, если вы что-то сломали или потеряли файлы, вы спокойно можете всё исправить. В свете усложнения сред разработки они помогают командам разработчиков работать быстрее и эффективнее.

1. Локальные системы контроля версий.

Многие люди в качестве метода используют метод контроля копирования файлов в отдельный каталог (возможно даже, каталог с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно прогрессирует в появлении ошибок. Можно легко забыть в каком-то каталоге, который вы находитесь, и случайно изменить не тот файл или не скопировать те файлы, которые вам нужны.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные VCS с простой базой данных, которая хранит записи обо всех изменениях в файлах, осуществляя тем самым контрольные ревизии.



Рисунок 1. Схема локального контроля

Одной из популярных систем VCS была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различные между файлами) в специальном формате, который может изменять состояние каждого файла в заданный момент времени.

1. Централизованные системы контроля версий.

Следующая серьёзная проблема, с которой сталкиваются люди, – это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (Centralized Version Control System, далее CVCS). 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. Применение CVCS являлось стандартом на протяжении многих лет.



Рисунок 2. Централизованный контроль версий

Такой подход имеет множество преимуществ, особенно перед локальными VCS. Например, 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 CVCS, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус – это единая точка отказа, представленная централизованным сервером. 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, вы потеряете всё – всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные VCS страдают от той же самой проблемы: когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

1. Распределенные системы контроля версий.

Здесь в игру вступают распределённые системы контроля версий (Distributed Version Control System, далее DVCS). В DVCS (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.



Рисунок 3. Распределённый контроль версий

Более того, многие DVCS могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

1. Что такое GIT и основные отличия от других СКВ.

Git – распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками.

Git стоит отдельно от других СКВ из-за подхода к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

**Преимущества Git:**

* Бесплатный и open-source. Это значит, что его можно бесплатно скачать и вносить любые изменения в исходный код;
* Небольшой и быстрый. Он выполняет все операции локально, что увеличивает его скорость. Кроме того, Git локально сохраняет весь репозиторий в небольшой файл без потери качества данных;
* Резервное копирование. Git эффективен в хранении бэкапов, поэтому известно мало случаев, когда кто-то терял данные при использовании Git;
* Простое ветвление. В других СКВ создание веток – утомительная и трудоёмкая задача, так как весь код копируется в новую ветку. В Git управление ветками реализовано гораздо проще и эффективнее.

1. Команды:
   1. Git init

git-init – создать пустой репозиторий Git или повторно инициализировать существующий.

* 1. Git add

Команда git add добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

* 1. Git commit

Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

* 1. Git status

Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

* 1. Git reset

Команда git reset, как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочей директории при использовании параметра – hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений, прежде чем использовать его.

* 1. Git log

Команда git log используется для просмотра истории коммитов, начиная с самого свежего и уходя к истокам проекта. По умолчанию, она показывает лишь историю текущей ветки, но может быть настроена на вывод истории других, даже нескольких сразу, веток. Также её можно использовать для просмотра различий между ветками на уровне коммито

* 1. Git branch

Команда git branch – это своего рода “менеджер веток”. Она умеет перечислять ваши ветки, создавать новые, удалять и переименовывать их.