МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**О Т Ч Ё Т**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

**«Системы контроля версий. Git»**

**по курсу:**

***"Конструирование программного обеспечения"***

|  |  |
| --- | --- |
| Выполнили, студенты гр. 10701121: | Журович К.С.  Гвоздь В.И.  Асанов Н.К. |
|  |  |
| Проверил: | Станкевич С.Н. |

Минск – 2023

**Цель работы:** узнать, что представляют из себя системы контроля версий. Научиться работе с системой контроля версий Git.

**Что такое Git?**

Что же такое Git, если говорить коротко? Очень важно понять эту часть материала, потому что если вы поймёте, что такое Git и основы того, как он работает, тогда, возможно, вам будет гораздо проще его использовать. Пока вы изучаете Git, попробуйте забыть всё, что вы знаете о других СКВ, таких как Subversion и Perforce. Это позволит вам избежать определённых проблем при использовании инструмента. Git хранит и использует информацию совсем иначе по сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож, и понимание этих различий поможет вам избежать путаницы во время использования.

**Снимки, а не различия**

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т. д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, **основанным на различиях**).



Рисунок 4. Хранение данных как набора изменений относительно первоначальной версии каждого из файлов

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**.



Рисунок 5. Хранение данных как снимков проекта во времени

Это очень важное отличие между Git и почти любой другой СКВ. Git переосмысливает практически все аспекты контроля версий, которые были скопированы из предыдущего поколения большинством других систем. Это делает Git больше похожим на миниатюрную файловую систему с удивительно мощными утилитами, надстроенными над ней, нежели просто на СКВ.

**Почти все операции выполняются локально**

Для работы большинства операций в Git достаточно локальных файлов и ресурсов — в основном, системе не нужна никакая информация с других компьютеров в вашей сети. Если вы привыкли к ЦСКВ, где большинство операций страдают от задержек из-за работы с сетью, то этот аспект Git заставит вас думать, что боги скорости наделили Git несказанной мощью. Так как вся история проекта хранится прямо на вашем локальном диске, большинство операций кажутся чуть ли не мгновенными.

Для примера, чтобы посмотреть историю проекта, Git не нужно соединяться с сервером для её получения и отображения — система просто считывает данные напрямую из локальной базы данных. Это означает, что вы увидите историю проекта практически моментально. Если вам 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.

Это также означает, что есть лишь небольшое количество действий, которые вы не сможете выполнить, если вы находитесь оффлайн или не имеете доступа к VPN в данный момент. Если вы в самолёте или в поезде и хотите немного поработать, вы сможете создавать коммиты без каких-либо проблем (в вашу **локальную** копию, помните?): когда будет возможность подключиться к сети, все изменения можно будет синхронизировать. Если вы ушли домой и не можете подключиться через VPN, вы всё равно сможете работать. Добиться такого же поведения во многих других системах либо очень сложно, либо вовсе невозможно. В Perforce, для примера, если вы не подключены к серверу, вам не удастся сделать многого; в Subversion и CVS вы можете редактировать файлы, но вы не сможете сохранить изменения в базу данных (потому что вы не подключены к БД). Всё это может показаться не таким уж и значимым, но вы удивитесь, какое большое значение это может иметь.

**Целостность Git**

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. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его философии. 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. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

Вы будете постоянно встречать хеши в Git, потому что он использует их повсеместно. На самом деле, Git сохраняет все объекты в свою базу данных не по имени, а по хеш-сумме содержимого объекта.

**Git обычно только добавляет данные**

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 СКВ, 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.

**Три состояния**

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.

Мы подошли к трём основным секциям проекта Git: рабочая копия (working tree), область индексирования (staging area) и каталог Git (Git directory).



Рисунок 6. Рабочая копия, область индексирования и каталог Git

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.

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:

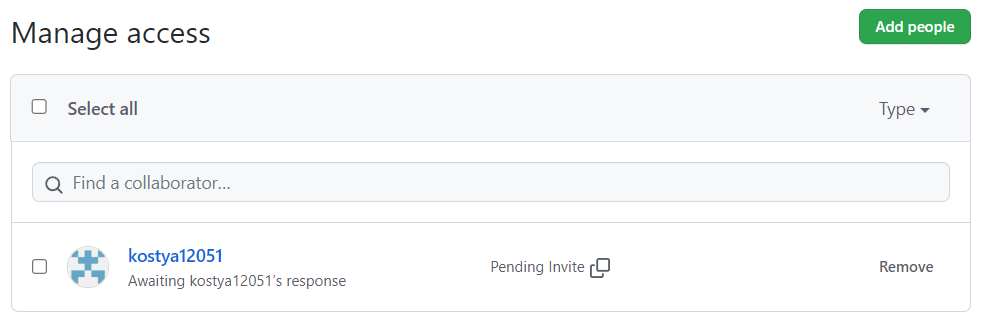
1. Изменяете файлы вашей рабочей копии.
2. Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки **только** этих изменений в индекс.
3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в ваш каталог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

**Результаты выполнения работы:**



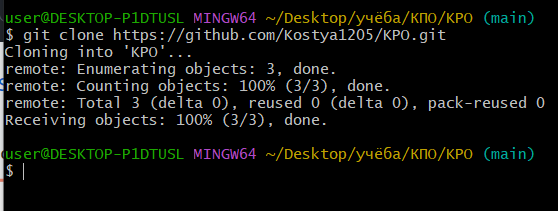
**Рисунок 1. Созданный репозиторий на GitHub**



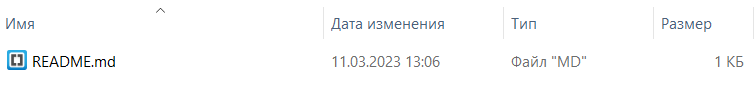
**Рисунок 2. Выдача доступа разработчику для работы с репозиторием.**

При помощи Git Bash клонируем репозиторий с GitHub на компьютер.

Для этого используется команда git clone.

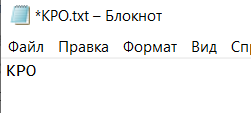


**Рисунок 3. Клонирование репозитория.**



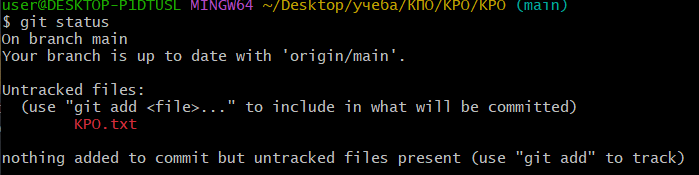
**Рисунок 4. Репозиторий успешно клонирован на компьютер.**

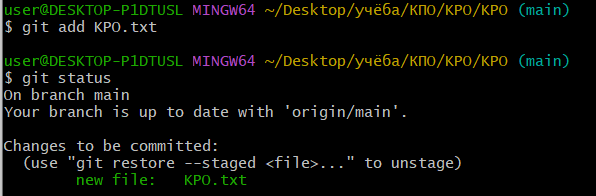




**Рисунок 5, 6. Создадим новый текстовый документ. Добавим в него нужную информацию.**

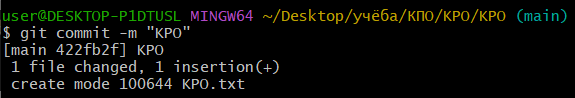
При помощи команды git status проверим какие файлы были добавлены в репозиторий на компьютере, а затем добавим файл в commit при помощи git add.





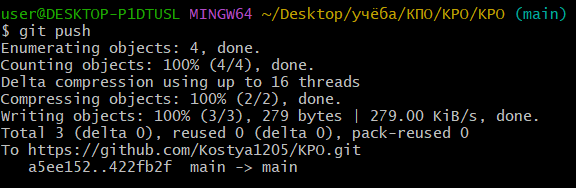
**Рисунок 7, 8. Проверка команды git status и git add.**

При помощи команды git commit -m “some text” сделаем коммит и добавим к нему комментарий “some text”



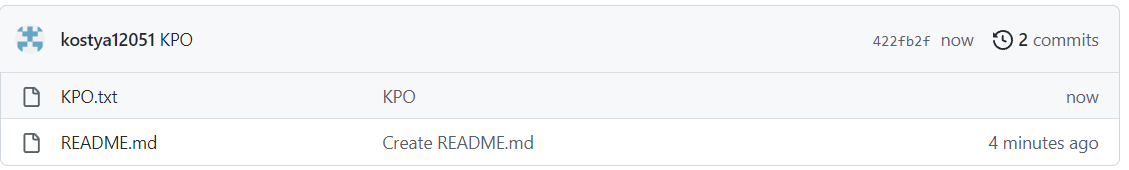
**Рисунок 9. Команда git commit.**

Затем при помощи команды git push загрузим изменённый файл на GitHub.



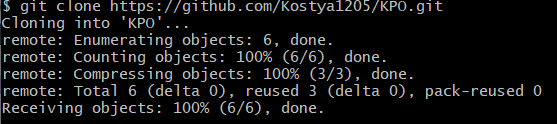
**Рисунок 10. Команда git push.**

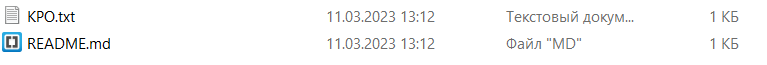
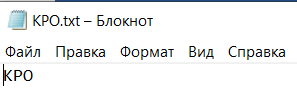
В результате файл KPO.txt появляется на GitHub.



**Рисунок 11. Репозиторий после добавления файла.**

С компьютера другого разработчика загрузим изменения на его компьютер.



**Рисунок 12, 13, 14. Результат обновления репозитория на компьютере другого разработчика.**

**Источники информации:**

Источники:

1. [Git - Что такое Git? (git-scm.com)](https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A7%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-Git%3F)