|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **НПОУ «ЯКУТСКИЙ КОЛЛЕДЖ ИННОВАЦИОННЫХ ТЕХНОЛОГИЙ»** | | | | | | | |  | |
| (наименование образовательной организации) | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | **ОТДЕЛЕНИЕ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ТУРИЗМА** | | | | | | | |  |
| (наименование структурного подразделения (кафедра / отделение)) | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
| **ОТЧЕТ** | | | | | | | | | | |
| **по учебной практике** | | | | | | | | | | |
| ПМ.01.03 Разработка модулей программного обеспечения для компьютерных систем | | | | | | | | | | |
| (наименование типа практики) | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | | |  | Выполнил: | | | | | |
|  | | | |  | обучающийся | | КИСП-22(1) | группы | | |
|  | | | |  | Зенков Данил Константинович | | | | | |
|  | | | |  | (фамилия, имя, отчество (при наличии)) | | | | | |
|  | | | |  | \_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_ 20\_\_\_ г. | | | | | |
|  | | | |  | (личная подпись, дата) | | | | | |
|  | | | |  |  | | | | | |
|  | | | |  |  | | | | | |
| Проверил руководитель практики | | | |  | Проверил руководитель практики | | | | | |
| от профильной организации: | | | |  | от образовательной организации: | | | | | |
|  | | | |  | преподаватель | | | | | |
| (уч. степень, уч. звание, должность) | | | |  | (уч. степень, уч. звание, должность) | | | | | |
|  | | | |  | Тронь Татьяна Александровна | | | | | |
| (фамилия, имя, отчество (при наличии)) | | | |  | (фамилия, имя, отчество (при наличии)) | | | | | |
| Оценка | | |  |  | Оценка |  | | | | |
| \_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_ 20\_\_\_ г. | | | |  | \_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_ 20\_\_\_ г. | | | | | |
| (личная подпись, дата) | | | |  | (личная подпись, дата) | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
| Рег. № \_\_\_\_\_\_\_ от «\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_\_ г. | | | | | | | | | | |
|  | | | | | | | | | | |
| Якутск, 2024 | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **НПОУ «Якутский колледж инновационных технологий»** | | | | | | | |  |
| (наименование образовательной организации) | | | | | | | |  |
|  | | | | | | | | | |
| **Отделение Информационных технологий и туризма** | | | | | | | |  |
| (наименование структурного подразделения (кафедра / отделение)) | | | | | | | |  |
|  | | | | | | | | | |
| УТВЕРЖДАЮ  заведующий отделения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / И.В. Пронин/  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г. | | | | | | | | | |
|  | | | | | | | | | |
| **ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ** | | | | | | | | | |
|  | | | | | | | | | |
| **Общие сведения** | | | | | | | | | |
|  | | | | | | | | | |
| ФИО обучающегося | | | Зенков Данил Константинович | | | | | | |
| Курс | | | 3 | | | | | | |
| Форма обучения | | | Очная | | | | | | |
| Направление подготовки / специальность / профессия | | | 09.02.07 Информационные системы и программирование | | | | | | |
| Наименование структурного подразделения (кафедра / отделение) | | | Отделение информационных технологий и туризма | | | | | | |
| Группа | | | КИСП-22(1) | | | | | | |
| Вид практики | | | Учебная | | | | | | |
| Тип практики | | | По модулю | | | | | | |
| Способ проведения практики | | | Стационарно | | | | | | |
| Форма проведения практики | | | Дискретно | | | | | | |
| Место прохождения практики | | | НПОУ «ЯКИТ» | | | | | | |
| Период прохождения практики | | | с 06.05.24  по 18.05.24 | | | | | | |
| Реквизиты договора о прохождении практики (при проведении практики в профильной организации) | | |  | | | | | | |
|  | | | | | | | | | |
| **Содержание индивидуального задания**   * Git. Введение. Установка и настройка * Git. Внутренняя реализация. Создание изменений: индексация и коммиты. Команды: init, status, add, commit. * Git. Ветвление. Создание и удаление веток, переключение между ними. Команды: branch, checkout, status, log, diff. * Перемещение курсора и отмена изменений. Команды git restore, git rm, git reset, git checkout, git commit, git revert. * Слияние изменений и продвинутая работа с ветками. Команды: merge, cherry-pick, rebase. * Git. Работа с удаленным репозиторием. Модель ветвления. Команды: clone, fetch, push, pull. | | | | | | | | | |
|  | | | | | | | | | |
| Задание на практику составил: | | | | | | | | | |
| руководитель практики от образовательной организации | | | | | | | | | |
|  |  |  | |  | Т.А. Тронь. |  | «06» мая 2024 г. | | |
| (уч. степень, уч. звание, должность) |  | (подпись) | |  | (И.О. Фамилия) |  | (дата) | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
| Задание на практику принял: | | | | | | | | | |
| обучающийся |  |  | |  | Д.К. Зенков |  | «06» мая 2024 г. | | |
|  |  | (подпись) | |  | (И.О. Фамилия) |  | (дата) | | |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc166618836)

[1. ПРАКТИЧЕСКАЯ ЧАСТЬ 6](#_Toc166618837)

[1.1. Установка и настройка Git 6](#_Toc166618838)

[1.2. Создание изменений: индексация и коммиты. Команды: init, status, add, commit 7](#_Toc166618839)

[1.3. Ветвление. Создание и удаление веток, переключение между ними. Команды: branch, checkout, status, log, diff. 9](#_Toc166618840)

[1.4. Перемещение курсора и отмена изменений. Команды git restore, git rm, git reset, git checkout, git commit, git revert. 13](#_Toc166618841)

[1.5. Слияние изменений и продвинутая работа с ветками. Команды: merge, cherry-pick, rebase. 17](#_Toc166618842)

[1.6. Работа с удаленным репозиторием. Модель ветвления. Команды: clone, fetch, push, pull. 19](#_Toc166618843)

[ЗАКЛЮЧЕНИЕ 22](#_Toc166618844)

# ВВЕДЕНИЕ

Данная учебная практика посвящена системе управления Git, которая позволяет разработчикам работать совместно над проектами и отслеживать изменения в коде.

Основные особенности Git:

* Каждый разработчик имеет полную копию истории проекта, поскольку полная копию репозитория лежит у вас на машине.
* Git позволяет отслеживает все изменения в коде и позволяет пользователям видеть, кто, когда и что изменил.
* Разработчики могут создавать ветки от основной ветки, чтобы экспериментировать с изменениями без влияния на основной код.
* После внесения изменений в ветку пользователи могут объединить их с основной веткой, что позволяет легко интегрировать изменения от разных участников.
* Git сохраняет подробную историю всех изменений, что позволяет пользователям легко отслеживать эволюцию проекта и откатывать изменения при необходимости.

Git широко используется в разработке программного обеспечения, но также находит применение в других областях, таких как управление конфигурациями и совместная работа над документами.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

Практической частью работы является работа с Git, которая состоит из следующих задач:

* Установка и настройка Git
* Изучение особенностей системы управления Git
* Применение команд системы управления Git

# Установка и настройка Git

Для начала установки нужно скачать загрузчик с официального сайта Git и выполнить установку приложения. В процессе выбора настроек установки нужно изменить некоторые важные моменты.

В моём случае загрузка Git будет происходить на операционную систему Windows.

Первое что требуется изменить, основное название ветки, поскольку Git по стандарту может менять название веток при обновлениях. В моём случае я изменил название основной ветки на «development» (Рис.1).

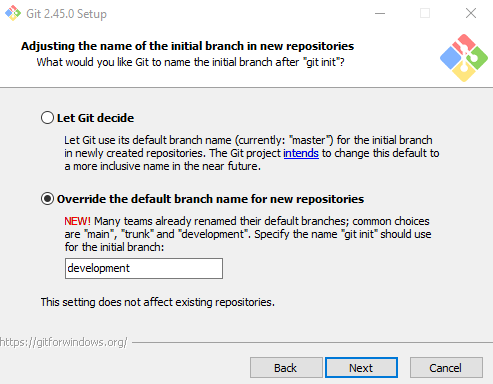


Рис.1. Переименование основной ветки

Второе изменение это доступ к запуску Git из консоли, чтобы позволить запускать Git из других ПО (Рис.2). Это позволит запускать Git напрямую из консолей.

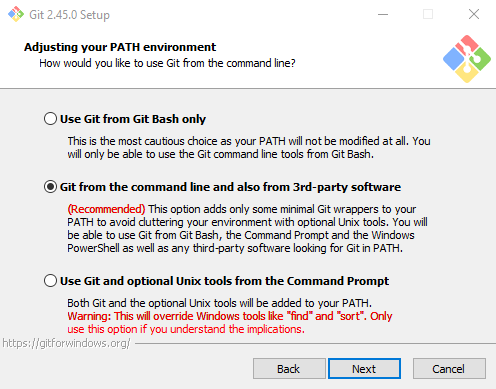


Рис.2. Настройка запуска Git из консоли

После настройки этих основных моментов, идут расширенные настройки которые не понадобятся в этой работе. Все остальные настройки остаются без изменений. Остаётся только установить Git и запустить его.

# Создание изменений: индексация и коммиты. Команды: init, status, add, commit

Команда git init – создаёт пустой репозиторий в директории, откуда была вызвана.

Репозиторий – это папка проекта, отслеживаемого Git, содержащая древо изменений проекта в хронологическом порядке. Все файлы истории хранятся в специальной скрытой папке .git внутри папки проекта.

Для начала работы, создадим папку GitFirst, в которой будем выполнять работу. Далее, запускаем программу Git Bash, и с помощью команды cd <директория к папке> войдём в неё (Рис.3). После того как мы вошли в неё, пропишем команду git init (Рис.3), и таким образом создадим пустой репозиторий.

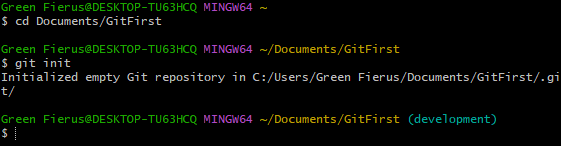


Рис.3. Создание пустого репозитория

Команда git status – выводит информацию о статусе файлов, находящихся в репозитории.

В нашем репозитории в данный момент ничего нет, он пустой. Если мы выполним команду git status сейчас, то он будет пуст. Создадим в нём два текстовых файла torn и horn. Теперь, при выполнении команды git status мы увидим наши файлы помеченные красным цветом. Это значит, что файлы ещё не были добавлены в репозиторий (Рис.4).

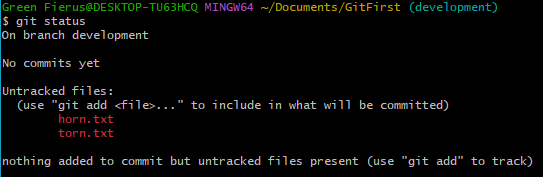


Рис.4. Выполнение команды git status.

Команда git add – добавляет файлы в индекс.

Индекс – файл, в котором содержатся изменения, подготовленные для добавления в коммит.

Добавим файлы torn и horn в репозиторий. Для этого, мы можем прописать команду git add <название файла> или для того чтобы добавить все файлы из папки написать точку вместо названий файлов (Рис.5).

Теперь, при выполнении команды status мы увидим немного иную картину, наши файлы станут зелёного цвета и будут считаться новыми (Рис.5).

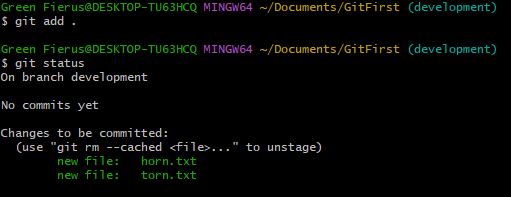


Рис.5. Добавление файлов.

Теперь остаётся выполнить команду git commit, чтобы сохранить состояние этих файлов в git.

Команда git commit – создаёт новый коммит с файлами из индекса.

Коммит – фиксация изменений, внесенных в индекс.

Выполним команду git commit чтобы зафиксировать состояние файлов (Рис 6.).

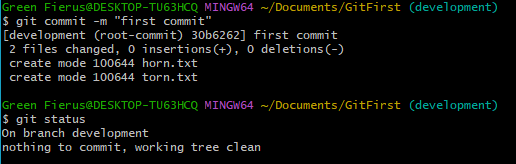


Рис.6. Выполнение коммита.

Мы выполнили коммит, а это значит, что мы сохранили текущее состояние файлов в древе, или же графе и создали объект коммита, и направили текущую ветку на новый коммит.

# Ветвление. Создание и удаление веток, переключение между ними. Команды: branch, checkout, status, log, diff.

Команда git branch – создание новой ветки.

Ветка – это последовательность коммитов. Ветки независимы, что к примеру позволяет проверять в них работу кода, без угроз поломки основной части проекта. Создадим ветку с названием feature (Рис.7).

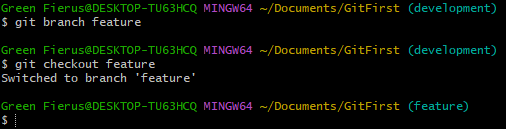


Рис.7. Создание и переход на другую ветку.

Чтобы переключится на другую ветку, необходимо использовать команду git checkout.

Команда git checkout – позволяет переключится на другую ветку или создать ветку и сразу переключится на неё (Рис.7).

В результате выполнения, у нас поменялась основная ветка development на feature.

Если нужно проверить количество веток в проекте, нужно написать команду git branch <ключ> (Рис.8).

Ключи для этой команды выглядят следующим образом:

* Ключ -a выведет все ветки.
* Ключ -r выведет только удалённые ветки.
* Команда без ввода ключа выведет только локальные ветки.

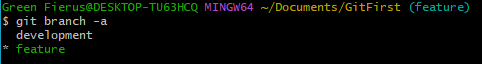


Рис.8. Вывод существующих веток.

Здесь мы видим две ветки:

* development – основная ветка;
* feature – созданная ветка.

Звёздочка и зелёный текст сигнализируют о ветке, которая сейчас выбрана.

Добавим в новую ветку какой-нибудь файл. Для этого создадим новый текстовый файл, под названием feather и напишем команду git status. Мы видим только один файл на сохранение, файлы torn и horn не видны, и если мы вернёмся в ветку development, мы будем видеть так же только файл feather. (Рис.9).

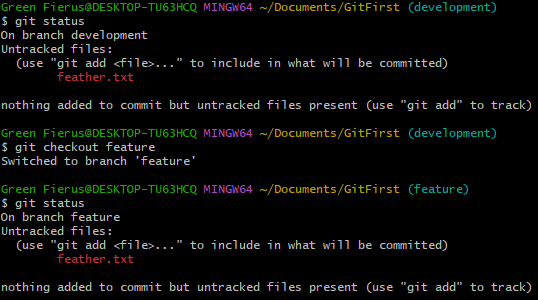


Рис.9. Отображение файла в среде.

Сохраним файл в новой ветке feature. Для этого нам нужно дублировать всё тоже самое, что делали при коммите в основной ветке. Добавить файл и выполнить коммит.

Команда git log – выводит историю коммитов. Работает через ключи:

* -<число>, -n <число> – выводит последние несколько коммитов. Их число указывается в параметре к ключу;
* -p – показывает изменения, сделанные в данном коммите;
* -all – выводит историю всех коммитов на всех ветках;
* --pretty=<значение> – красивый вывод истории. Доступные значения: oneline, short, medium, full;

Выполним команду git log (Рис.10).

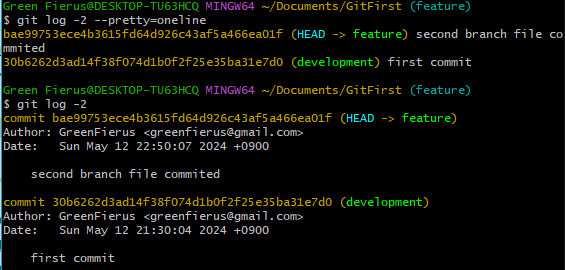


Рис.10. Выполнение команды log.

Вызов был осуществлён в двух вариациях. Pretty и обычный. Сверху показаны последние коммиты, а наоборот, те которые были до последнего.

Команда git diff – помогает просмотреть изменения между файлами, коммитами, ветками.

Команда работает по принципу: git diff <ключи> <путь до файла> <путь до файла>. Имеет следующие ключи:

* --diff-filter=<метка> – Позволяет выбирать, изменения между какими файлами показывать. Возможные значения метки: A – добавить файлы, добавленные в индекс, D – удалить файлы, M – файлы, в которые были внесены изменения с момента последнего коммита, и некоторые другие.
* --word-diff=color – Подсвечивает добавленные слова зеленым цветом, а удаленные – красным. По умолчанию словом считается неразрывная последовательность символов между пробелами.

Выполним команду (Рис.11).

Для выполнения команды нам нужны будут хэши веток, которые мы собираемся сравнивать. Их можно увидеть через использование команды git log. Хэшами являются тегом ветки, они выглядят как хаотичный набор символов. Для них применения необязательно выписывать их полностью, хватит пяти-шести первых символов.

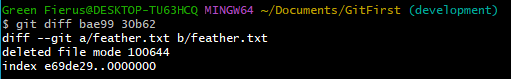


Рис.11. Выполнение git diff.

Данная команда показывает разницу между версиями файлов, или веток. В данном случае разницей является наличие текстового файла feather.

Если нам потребуется удалить ветку, нужно прописать git branch с ключом -d, что означает удалить, но будет удалена только в том случае, если она была полностью слита с другой веткой. В ином случае, git не позволит удалить её. Для игнорирования этого ограничения, есть ключ -D.

Создадим новую ветку под названием todel, которую мы удалим (Рис.12).

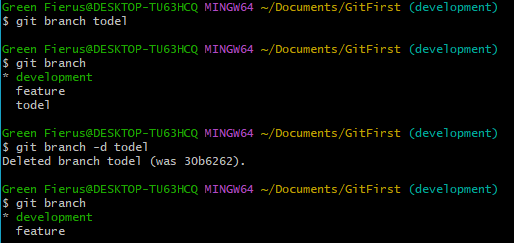


Рис.12. Удаление ветки.

Поскольку ветка была пустой, git не мешал её удалению.

# Перемещение курсора и отмена изменений. Команды git restore, git rm, git reset, git checkout, git commit, git revert.

Команда git rm – Удаляет файл из рабочей копии только из индекса. Данная команда не может удалить файл только из рабочей копии.

Удалим файл feather: для этого нам нужно написать команду git rm feather, то есть название файла, который требуется удалить (Рис.13).

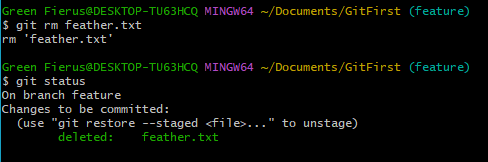


Рис.13. Удаление файлов.

В статусе видно, что feather считается удалённым, но это ещё не всё, ведь требуется выполнить коммит для фиксации изменения. Прописываем коммит и оставляем комментарий для наглядности.

Теперь восстановим этот файл. Для этого есть команда git restore.

Команда git restore – Восстанавливает указанный файл из переданного источника. По умолчанию источником является индекс. Если файла нет в указанном источнике, файл будет удален. Имеет следующие ключи:

* Ключ -s, --sourse=<название\_древа> нужен для того, чтобы передать команде путь к коммиту, откуда мы будет восстанавливать файл. По умолчанию берется из области индекса.
* Ключ – --worktree – восстанавливает файл в рабочей копии.
* Ключ – --staged – восстанавливает файл в области индекса.

Проверим работу функции в новой ветке delfunc, в которую будут добавлены три текстовых файла cold, hot, weather. Добавим все три файла в ветку, и восстановим файл cold в области индекса (Рис.14).

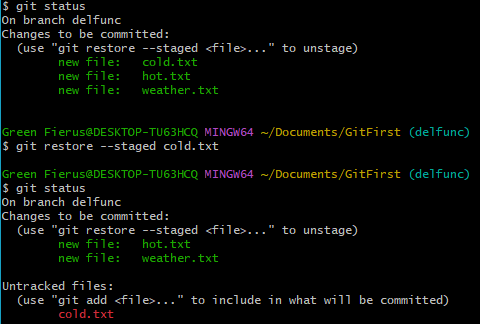


Рис.14. Возвращение файла.

Пока изменения не были закоммитчены, их можно удалять и возвращать без потери файла.

Используя команду log, мы могли заметить указатель HEAD возле хэшей коммита. HEAD – указатель, который показывает текущий выбранный хэш, с его помощью можно перемещаться на какое-то определённое число коммитов назад.

Используя знак ^ можно спуститься на один предыдущий коммит стоящий перед указателем HEAD. Ставится вплотную после написания HEAD.

Используя знак ~ можно спуститься на указанное количество коммитов стоящих перед указателем HEAD. Ставится вплотную после написания HEAD с числом, которое будет указывать на количество нужных спусков.

Для применения мы пишем команду git checkout. С помощью этой команды можно переходить на определённые ветки, коммиты через хэши и спускаться на несколько коммитов ниже с использованием HEAD.

У меня в системе выполнено пять коммитов. Я хочу переместить указатель HEAD на три коммита тому назад (Рис.15).

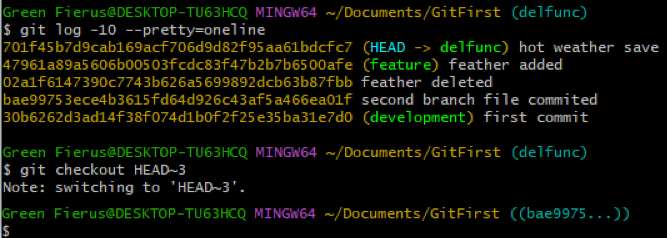


Рис.15. Перемещение указателя HEAD.

Если нам нужно вернуться на последний коммит, нам потребуется перейти по нему с помощью хэша и уже оттуда можно будет использовать все функции, которые будут находиться в нём.

Команда git revert – отменяет изменения, внесенные в переданном коммите. Имеет следующие ключи:

* Ключ -n – не делает коммит. С данным ключом изменения коснутся только рабочей копии.
* Ключ --abort – данный ключ используется только при разрешении конфликтов. Прерывает выполнение команды после разрешения конфликтов.
* Ключ --continue – данный ключ используется только при разрешении конфликтов. Продолжает выполнение команды после разрешения конфликтов.

Если нам нужно вернуться на последний коммит, нам потребуется перейти по нему с помощью хэша и уже оттуда можно будет использовать все функции, которые будут находиться в нём.

Команда git revert так же позволяет использование указателя HEAD. Ревертируем коммиты (Рис.16).

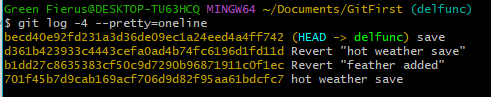


Рис.16. Реверт коммита.

По логу видно, что два коммита было инвертировано, а это значит, что их сохранения не вступали в силу, а это выходит, что файлы hot, weather и feather не были добавлены в них.

Команда git reset – позволяет перемещать указатель ветки вместе с указателем HEAD. Переносит указатель ветки на переданный коммит. Имеет следующие ключи:

* Ключ – --soft – С этим ключом, команда не отменяет изменения ни в индексе, ни в рабочей копии. Все ваши файлы останутся в том же состоянии, в котором были, но указатель ветки будет передвинут.
* Ключ – --hard – С этим ключом команда удалит все изменения так, чтобы состояние индекса и рабочей копии полностью соответствовали коммиту, к которому мы сделали reset.

Сделаем reset последнего коммита, который мы сделали по ошибке. При этом оставим файлы в том же состоянии (Рис.17).

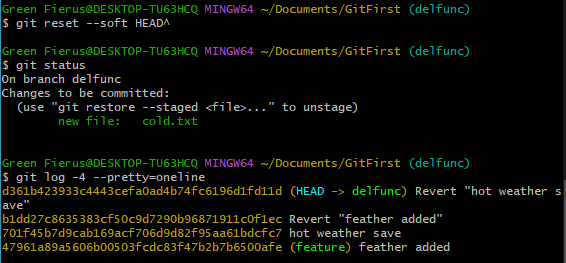


Рис.17. soft reset коммита.

До применения ресета, у нас был коммит с комментарием «save», в нём был добавлен файл cold в delfunc. После применения команды видно, что в статусе файл cold считается добавленным, но не сохранённым.

Теперь применим hard reset (Рис.18).

В случае hard reset, коммит save cold полностью отменяется, из-за чего файл cold просто исчезнет.

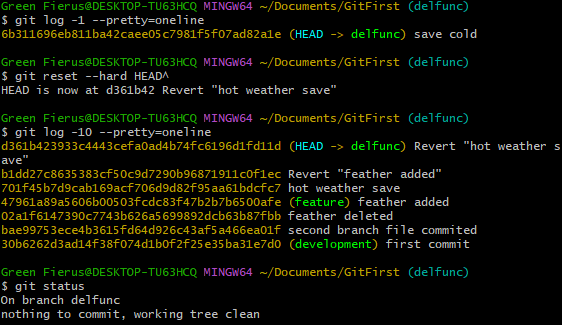


Рис.18. hard reset коммита.

# Слияние изменений и продвинутая работа с ветками. Команды: merge, cherry-pick, rebase.

Команда git merge – сливаем изменения с переданной ветки в текущую.

Для этого нам понадобится две ветки, сливаемая ветка и целевая. В целевую ветку мы будет отправлять данные сливаемой.

У команды merge есть следующие ключи:

* Ключ -ff позволяет при возможности включить fast-forward;
* Ключ --no-ff отключает fast-forward;
* Ключ --ff-only останавливает merge, если fast-forward невозможно сделать;
* Ключ --abort использующийся только при разрешении конфликтов. Позволяет прервать слияние и вернуть все к моменту начала операции;
* Ключ --continue использующийся только при разрешении конфликтов. Позволяет продолжить слияние после разрешения всех конфликтов.

Неявное слияние – во время неявного слияния не создается новых коммитов: используются только уже существующие. Суть этого слияния заключается в том, что из вливаемой ветки извлекаются несколько коммитов, а затем они применяются к последнему коммиту целевой ветки. Такое слияние называется fast-forward.

Соединим ветку feature с основной веткой development (Рис.19).

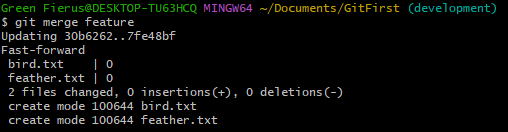


Рис.19. Соединение веток.

Для соединения ветки сперва нужно перейти на ту ветку, в которою мы хотим сделать соединение, после выполнить merge. То, что мы соединили ветку, не значит что мы от неё избавились. Она до сих пор числится как ветка, только её функции были переданы в основную ветку.

Команда git rebase – Перемещает все коммиты: от общего коммита двух веток до последнего коммита текущей ветки на вершину переданной ветки.

Git rebase переносит коммиты текущей ветки на вершину переданной.

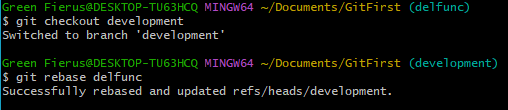


Рис.20. Применение rebase.

Команда работает так, что в примере перенесены коммиты ветки development на верхушку ветки delfunc.

Git rebase можно выполнять для одной и той же ветки.

Команда git cherry-pick – Берёт переданный коммит и создаёт в текущей ветке его точную копию.

Создадим новую ветку под названием cherry, и сделаем в ней два коммита, в каждом из которых будет добавлен по одному файлу.

Добавим в основную ветку один из этих коммитов (Рис.21).

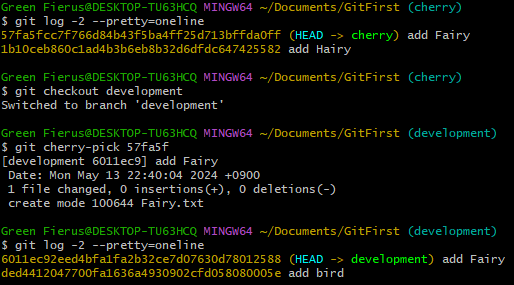


Рис.21. Применение cherry-pick.

Применение cherry-pick очень простое, требуется хэш коммита чтобы выбрать определённое действие и перенести его. Текстовый файл Fairy теперь является частью основной ветки development, файл Hairy был проигнорирован, ибо не изменялся или добавлялся в коммите добавления Fairy.

# Работа с удаленным репозиторием. Модель ветвления. Команды: clone, fetch, push, pull.

Удалённый репозиторий – это версия вашего проекта, сохранённая на удалённом сервере. Доступ к репозиторию на таком сервере может осуществляться по интернету или по локальной сети.

Для работы с удалённым репозиторием нам нужно создать аккаунт на Github.

Github – это сервис облачного хранилища, в котором можно сохранять проекты или какие то части проекта, для переноса с одного устройства на другое. Часто используется для групповых разработок.

Команда git remote add <Название> <Ссылка> – подключает удалённый репозиторий к вашему под переданным именем.

Имя удалённого репозитория можно придумать самому.

Ссылка на удалённый репозиторий получается в самом Github, через проект который мы собираемся загружать на наше устройство.

Команда git remote remove <название> – отключает переданный удалённый репозиторий от вашего.

Команда git remote rename <старое> <новое> – меняет имя переданного удалённого репозитория.

Команда git remote show [имя удалённого репозитория] – Выводит список всех подключенных удаленных репозиториев. Если передано имя репозитория, то выводит информацию об этом репозитории.

Команда git clone <ссылка на удалённый репозиторий> – клонирует переданный репозиторий на ваш компьютер (Рис.22).

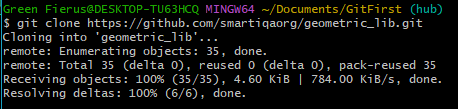


Рис.22. Клонирование по ссылке.

Для клонирования нам нужно иметь только ссылку на проект, который нам нужен.

Команда git fetch – получает изменения из переданного удалённого репозитория. Если не было передано ни одного удалённого репозитория, команда пытается получить изменения из репозитория с именем origin.

Команда git pull – получает изменения из переданного удаленного репозитория и обновляет рабочую копию в соответствии с удаленным репозиторием. По умолчанию слияние удаленной ветки с локальной происходит именно в fast-forward режиме, так что включать его специально не требуется.

Команда git push – Загружает изменения в удаленный репозиторий. Если слияние изменений в удаленном репозитории нельзя сделать в режиме fast-forward, и при этом не был передан ключ force, выполнение закончится с ошибкой.

Создадим удалённый репозиторий на Gtihub.

Для его создания нужно найти кнопку его создания, дать ему имя, описание если это нужно и выбрать дополнительные опции если они нужны.

Мой удалённый репозиторий будет иметь название GitFirstUse и по стандарту в него был добавлен файл README.md.

Создание форка репозитория на Github. Пулл-рекветы.

Форк – точная копия репозитория, но в вашем аккаунте. Форки нужны чтобы вносить свои изменения в проект, к репозиторию которого у вас нет прямого доступа.

Пулл-реквест – функция Github, позволяющая попросить владельца репозитория, от которого мы сделали форк, загрузить наши изменения обратно в свой репозиторий.

В Github если мы зайдём на страницу проекта, который мы хотим скопировать есть кнопка с названием Fork. Нажав на неё, на нашем аккаунте появится точная копия этого проекта.

Далее его можно будет клонировать у себя на устройстве, и если в нём были сделаны изменения, можно командой git push загрузить эти изменения на удалённый репозиторий.

# ЗАКЛЮЧЕНИЕ

В результате учебной практики были изучены инструменты совместной разработки Git и Github. Был получен опыт работы использования команд Git, которые позволяют сохранять состояния программ, локально на вашем компьютере и при надобности в облачном хранилище Github.

В ходе учебной практики были успешно освоены основные понятия и термины, связанные с Git и Github, такие как:

* Установка и настройка;
* Внутренняя реализация и создание изменений;
* Создание и удаление веток, переключение между ними;
* Перемещение курсора и отмена изменений;
* Слияние веток, их перемещение и выбор определённой версии сохранения;
* Работа с удалённым репозиторием.

Данная учебная практика позволит избежать проблем непонимания работы с Git и Github, что позволит сразу быть подготовленным к работе с этими инструментами в будующем.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **НПОУ «Якутский колледж инновационных технологий»** | | | | | | | | | | |  |
| (наименование образовательной организации) | | | | | | | | | | |  |
|  | | | | | | | | | | | | |
| **Отделение Информационных технологий и туризма** | | | | | | | | | | |  |
| (наименование структурного подразделения (кафедра / отделение)) | | | | | | | | | | |  |
|  | | | | | | | | | | | | |
| УТВЕРЖДАЮ  заведующий отделения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / И.В. Пронин/  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г. | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **РАБОЧИЙ ГРАФИК (ПЛАН) ПРОВЕДЕНИЯ ПРАКТИКИ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Общие сведения** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| ФИО обучающегося | | | | Зенков Данил Константинович | | | | | | | | |
| Курс | | | | 3 | | | | | | | | |
| Форма обучения | | | | Очная | | | | | | | | |
| Направление подготовки / специальность / профессия | | | | 09.02.07 Информационные системы и программирование | | | | | | | | |
| Наименование структурного подразделения (кафедра / отделение) | | | | Отделение информационных технологий и туризма | | | | | | | | |
| Группа | | | | КИСП-22(1) | | | | | | | | |
| Вид практики | | | | Учебная | | | | | | | | |
| Тип практики | | | | По модулю | | | | | | | | |
| Способ проведения практики | | | | Стационарно | | | | | | | | |
| Форма проведения практики | | | | Дискретно | | | | | | | | |
| Место прохождения практики | | | | НПОУ «ЯКИТ» | | | | | | | | |
| Период прохождения практики | | | | с 06.05.24  по 18.05.24 | | | | | | | | |
| Реквизиты договора о прохождении практики (при проведении практики в профильной организации) | | | |  | | | | | | | | |
|  | | | | | | | | | | | | |
| **Планируемые работы** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| № п/п | Содержание работы | | | | | | Срок выполнения | | | Отметка о выполнении | | |
|  | Оформление документов по прохождению практики | | | | | | до начала практики | | |  | | |
|  | Проведение медицинских осмотров (обследований) в случае выполнения обучающимся работ, при выполнении которых проводятся обязательные предварительные и периодические медицинские осмотры (обследования) в соответствии с законодательством РФ | | | | | | до начала практики | | |  | | |
|  | Вводный инструктаж по правилам охраны труда, технике безопасности, пожарной безопасности, оформление временных пропусков для прохода в профильную организацию (при необходимости). | | | | | | в первый день практики | | |  | | |
|  | Выполнение индивидуального задания практики | | | | | | в период практики | | |  | | |
|  | Консультации руководителя(-ей) практики о ходе выполнения заданий, оформлении и содержании отчета, по производственным вопросам | | | | | | в период практики | | |  | | |
|  | Подготовка отчета по практике | | | | | | за два дня до промежуточной аттестации | | |  | | |
|  | Проверка отчета по практике, оформление характеристики руководителя(-ей) практики | | | | | | за два дня до промежуточной аттестации | | |  | | |
|  | Промежуточная аттестация по практике | | | | | | в последний день практики | | |  | | |
|  | | | | | | | | | | | | |
| Рабочий график (план) составил: | | | | | | | | | | | | |
| руководитель практики от образовательной организации | | | | | | | | | | | | |
|  | |  |  | |  | Т.А. Тронь. | |  | «06» мая 2024 г. | | | |
| (уч. степень, уч. звание, должность) | |  | (подпись) | |  | (И.О. Фамилия) | |  | (дата) | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| С рабочим графиком (планом) ознакомлен: | | | | | | | | | | | | |
| обучающийся | |  |  | |  | Д.К. Зенков | |  | «06» мая 2024 г. | | | |
|  | |  | (подпись) | |  | (И.О. Фамилия) | |  | (дата) | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **НПОУ «Якутский колледж инновационных технологий»** | | | | | | | |
|  | (наименование образовательной организации) | | | | | | | |
|  | | | | | | | | | |
|  | **Отделение Информационных технологий и туризма** | | | | | | | |
|  | (наименование структурного подразделения (кафедра / отделение)) | | | | | | | |
|  | | | | | | | | | |
| **ДНЕВНИК ПРАКТИКИ** | | | | | | | | | |
|  | | | | | | | | | |
| **Общие сведения** | | | | | | | | | |
|  | | | | | | | | | |
| ФИО обучающегося | | | | Зенков Данил Константинович | | | | | |
| Курс | | | | 3 | | | | | |
| Форма обучения | | | | Очная | | | | | |
| Направление подготовки / специальность / профессия | | | | 09.02.07 Информационные системы и программирование | | | | | |
| Наименование структурного подразделения (кафедра / отделение) | | | | Отделение информационных технологий и туризма | | | | | |
| Группа | | | | КИСП-22(1) | | | | | |
| Вид практики | | | | Учебная | | | | | |
| Тип практики | | | | По модулю | | | | | |
| Способ проведения практики | | | | Стационарно | | | | | |
| Форма проведения практики | | | | Дискретно | | | | | |
| Место прохождения практики | | | | НПОУ «ЯКИТ» | | | | | |
| Период прохождения практики | | | | с 06.05.24  по 18.05.24 | | | | | |
| Реквизиты договора о прохождении практики (при проведении практики в профильной организации) | | | |  | | | | | |
|  | | | | | | | | | |
| **Учет выполняемой работы** | | | | | | | | | |
| |  |  |  |  | | --- | --- | --- | --- | | № п/п | Содержание работы | Дата выполнения | Отметка о выполнении | |  | Ознакомился с заданием практики | 06.05. 2024 |  | |  | Начал выполнение работы | 07.05. 2024 |  | |  | Установил Git на свой компьютер | 09.05. 2024 |  | |  | Начал изучение внутренней реализации | 10.05. 2024 |  | |  | Разобрался как работают ветвления в Git | 11.05. 2024 |  | |  | Научился работать с перемещением курсора и удалением лишних действий | 13.05. 2024 |  | |  | Научился передавать данные из веток | 14.05. 2024 |  | |  | Ознакомился с Github и работой с ним | 15.05. 2024 |  | |  | Начал писать отчёт | 16.05. 2024 |  | |  | Редактировал отчёт | 17.05. 2024 |  | |  | Сдал отчёт | 18.05.2024 |  | | | | | | | | | | |
| Дневник заполнил: | | | | | | | | | |
| обучающийся | |  |  | |  | Д.К Зенков |  | «18» мая 2024 г. | |
|  | |  | (подпись) | |  | (И.О. Фамилия) |  | (дата) | |
|  | | | | | | | | | |
| Дневник проверил: | | | | | | | | | |
| руководитель практики от образовательной организации | | | | | | | | | |
|  | |  |  | |  | Т.А. Тронь. |  | «18» мая 2024 г. | |
| (уч. степень, уч. звание, должность) | |  | (подпись) | |  | (И.О. Фамилия) |  | (дата) | |