

Вариант 4. Терминал (Terminal) или Командная строка (или Command Shell Git)

1. Выбор инструмента

В качестве клиента для работы с системой контроля версий Git был выбран терминал или командная строка. Данный инструмент представляет собой базовый и универсальный способ взаимодействия с Git, предоставляющий полный доступ ко всем функциям и командам, который поддерживается и одинаково работает во всех операционных системах (Windows, Linux, macOS) и позволяет реализовать полный цикл операций с репозиторием – от инициализации и ведения истории изменений до работы с ветвлением и удаленными репозиториями. Выбор данного варианта обусловлен его универсальностью, независимостью от сторонних приложений и тем фактом, что терминал – это стандартный инструмент, который применяется как в учебной, так и в профессиональной практике.

2. Краткий обзор

Командная строка является базовым инструментом, позволяющим выполнять все операции с репозиторием при помощи текстовых команд. Такой способ работы предоставляет полный контроль над процессом, обеспечивает доступ ко всем возможностям Git и не зависит от наличия дополнительных графических приложений.

К преимуществам использования командной строки можно отнести универсальность, так как данный инструмент доступен в любой операционной системе, а также гибкость, позволяющую точно управлять всеми стадиями работы с файлами и коммитами. В то же время основными недостатками являются отсутствие пользовательского интерфейса (UI – user interface), кнопок для легкого управления проектом и явной визуализации, а следовательно необходимо знать синтаксис команд и разбираться в их назначении, что может создавать определенные трудности для начинающих пользователей.

3. Подготовка репозитория

На первом этапе работы необходимо создать локальный репозиторий, в котором будут храниться все версии файлов проекта. Для этого в терминале используется команда **git init**. Она инициализирует новый репозиторий в выбранной директории, создавая скрытую папку **.git**, где сохраняется вся служебная информация для отслеживания изменений.

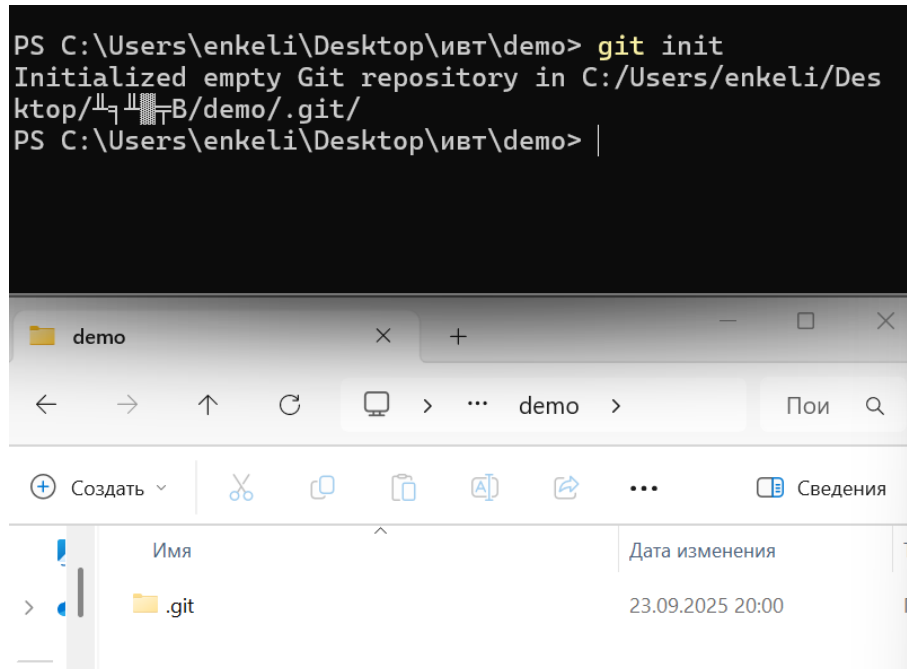


Рисунок 1 – Инициализация нового локального репозитория командой **git init**

Альтернативным вариантом является клонирование уже существующего удаленного репозитория с помощью команды **git clone <URL_удаленного_репозитория>**. В этом случае на локальный компьютер копируется весь проект вместе с историей изменений. Такой подход удобен, когда требуется подключиться к совместной разработке или продолжить работу с уже подготовленным проектом.

4. Работа с файлами и коммитами

После инициализации или клонирования репозитория следующим этапом является работа с файлами. Git позволяет отслеживать изменения, фиксировать их и сохранять историю в виде последовательности коммитов.

Для начала проверим текущее состояние репозитория с помощью команды **git status**. В выводе отображаются все файлы, которые находятся в рабочей директории, а также их текущее состояние: неотслеживаемые, измененные или подготовленные к коммиту. На рисунке видно, что файлов нет, добавлять в staging area нечего и коммитить, собственно, тоже, что нам и отвечает Терминал.

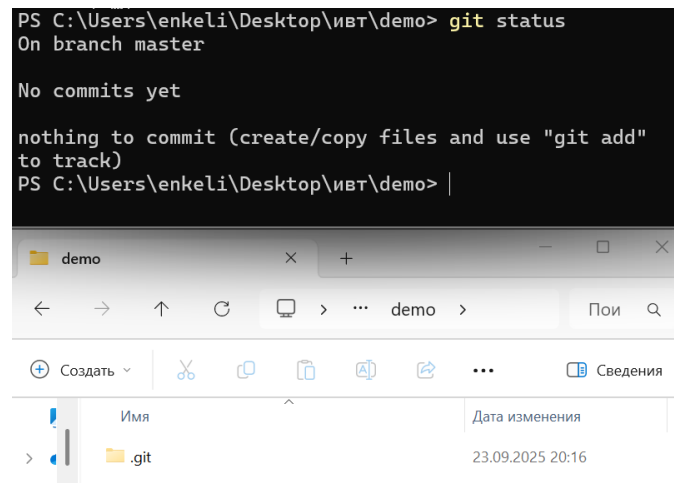


Рисунок 2 – Проверка состояния репозитория командой `git status`

Далее создадим новый текстовый файл, изменим его и снова выполним команду **git status**. Видим, что появился новый файл, и он не отслеживаемый (untracked), так как еще не добавлен в staging area.

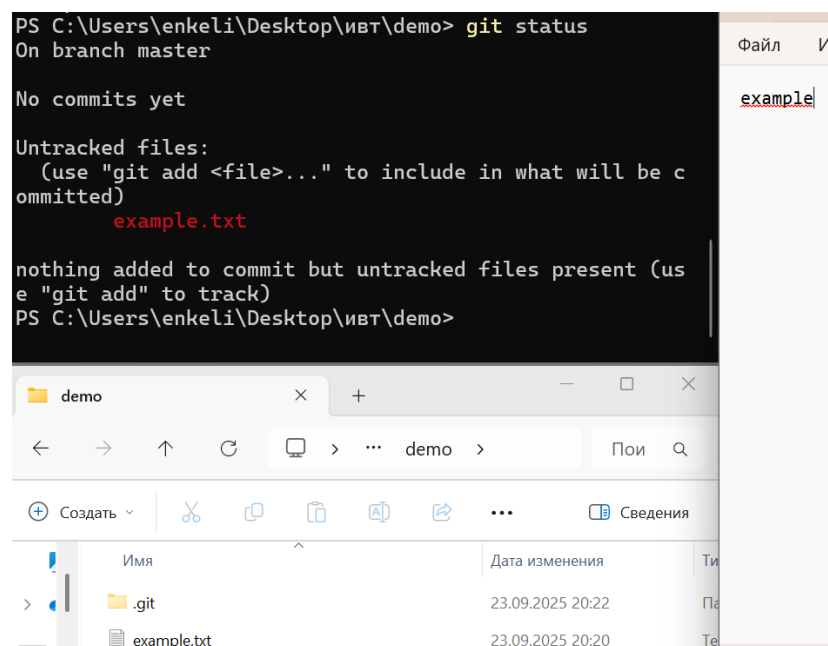


Рисунок 3 – Отображение нового неотслеживаемого файла `example.txt` в результате выполнения команды `git status`

Добавим его в область подготовки (staging area). Для этого используется команда **git add <имя_файла>**. При необходимости можно добавить несколько файлов сразу, указав маску, например **git add '*.txt'**. Для добавления всех измененных файлов (чтобы не перечислять имена файлов или маски) можно выполнить команду **git add .** (с точкой в конце). После выполнения этой команды выбранные файлы будут отмечены как готовые к фиксации.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git add .
PS C:\Users\enkeli\Desktop\ивт\demo> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   example.txt

PS C:\Users\enkeli\Desktop\ивт\demo> |
```

Рисунок 4 – Добавление файла example.txt в область подготовки (staging area) командой git add

Чтобы сохранить изменения в истории проекта, выполняется команда **git commit -m "сообщение"**. В сообщении указывается краткое описание произведенных изменений, что позволяет в дальнейшем легко ориентироваться в истории коммитов. Видим, что изменения успешно сохранены: 1 файл изменен, коммитить нечего.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git commit -m "Создан файл example.txt"
[master (root-commit) e0d0ac1] 1 file changed, 1 insertion(+)
 create mode 100644 example.txt
PS C:\Users\enkeli\Desktop\ивт\demo> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\enkeli\Desktop\ивт\demo> |
```

Рисунок 5 – Фиксация изменений в репозитории командой git commit -m

Просмотреть зафиксированные изменения можно с помощью команды **git log**. Она выводит список всех коммитов в обратном хронологическом порядке. Более подробная информация, включая перечень затронутых файлов, отображается при использовании команды **git log --summary**. Таким образом, с помощью этих команд обеспечивается полный цикл

фиксации изменений: от подготовки файлов до сохранения и анализа истории. На рисунке видим id коммита, автора, дату и сообщение к коммиту, во втором случае еще видим, что был создан файл example.txt.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git log
commit e0d0ac1b193fdc87865ce4c2faf0b9a17df71f3b (HEAD
-> master)
Author: Трофимцова Екатерина Евгеньевна <enkelilu@yandex.ru>
Date: Tue Sep 23 20:34:08 2025 +0300

    Создан файл example.txt
PS C:\Users\enkeli\Desktop\ивт\demo> git log --summary
commit e0d0ac1b193fdc87865ce4c2faf0b9a17df71f3b (HEAD
-> master)
Author: Трофимцова Екатерина Евгеньевна <enkelilu@yandex.ru>
Date: Tue Sep 23 20:34:08 2025 +0300

    Создан файл example.txt

    create mode 100644 example.txt
PS C:\Users\enkeli\Desktop\ивт\demo> |
```

Рисунок 6 – Просмотр истории коммитов с помощью команд `git log` и `git log --summary`

5. Работа с удаленным репозиторием

Git позволяет не только хранить историю изменений локально, но и синхронизировать ее с удаленным репозиторием. Это необходимо для совместной работы и резервного копирования проекта.

Для начала требуется связать локальный репозиторий с удаленным. Такая привязка выполняется командой `git remote add origin <URL>`, где `<URL>` – адрес удаленного репозитория (например, на GitHub или GitLab). Вместо URL-адреса можно написать SSH-ключ, скопировать который можно на странице удаленного репозитория. По соглашению имя `origin` используется по умолчанию для основного удаленного репозитория.

После этого можно выгрузить локальные коммиты в удаленный репозиторий с помощью команды `git push -u origin master`. Опция `-u` устанавливает связь между локальной и удаленной веткой, что позволяет в дальнейшем использовать более короткую команду `git push`. На рисунке ниже видно, что локальный и удаленный репозитории были связаны и

изменения с локального выгружены в удаленный (терминал ответил, что удаленный репозиторий уже существует, так как эта команда была выполнена мной чуть выше).

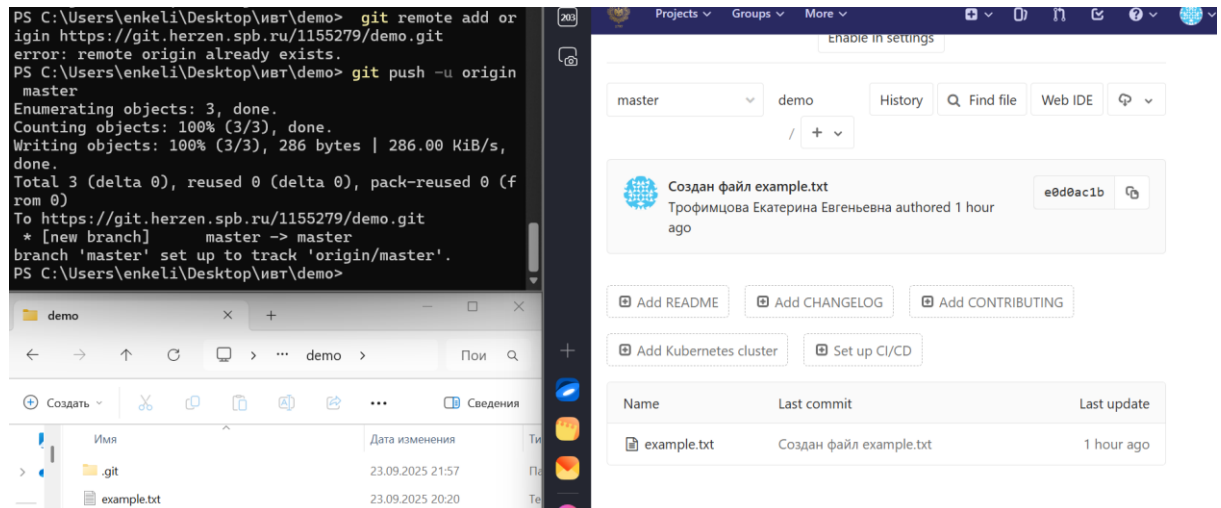


Рисунок 7 – Связывание локального и удалённого репозитория и выгрузка изменений командой `git push -u origin master`

Проверить список удаленных репозиториях можно командой `git remote -v`.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git remote -v
origin https://git.herzen.spb.ru/1155279/demo.git (fetch)
origin https://git.herzen.spb.ru/1155279/demo.git (push)
```

Рисунок 8 – Проверка списка удаленных репозиториях

Внесем изменения в файл `example.txt`, сделаем коммит и выгрузим на удаленный репозиторий. Видим, что на GitLab появилась информация о новом коммите, а также в терминале написано, что изменен один файл, добавлено две строчки.

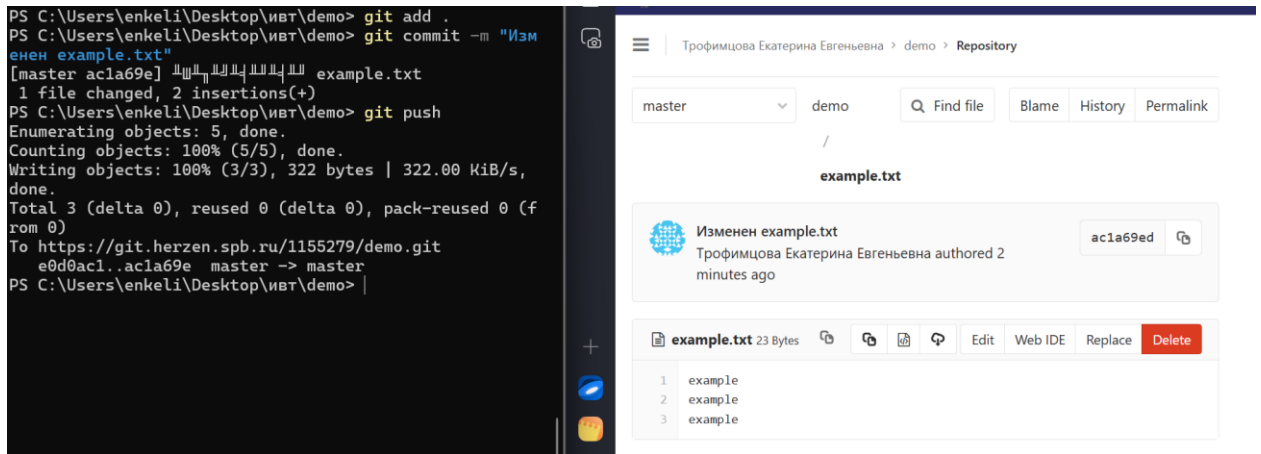


Рисунок 9 – Фиксация изменений в файле example.txt и выгрузка нового коммита в удалённый репозиторий

Чтобы загрузить последние изменения из удаленного репозитория и объединить их с локальной копией, используется команда **git pull origin master**. Для демонстрации изменим файл example.txt прямо на GitLab и сделаем git pull.

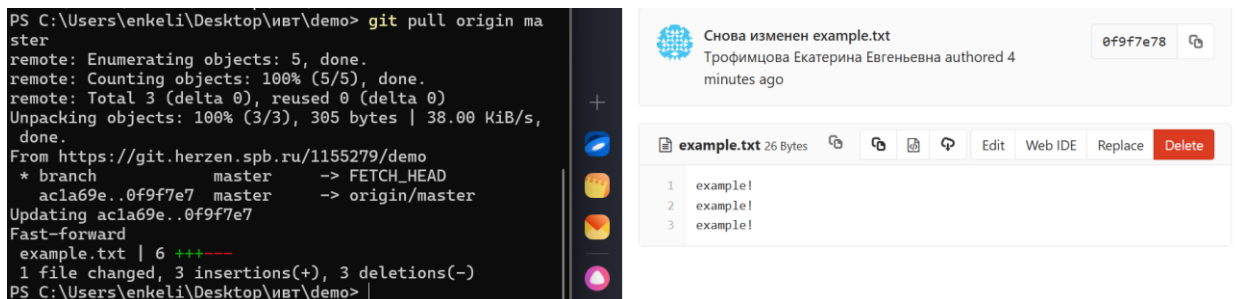


Рисунок 10 – Получение изменений из удалённого репозитория командой git pull origin master

Таким образом, локальный и удаленный репозиторий синхронизируются между собой.

Для анализа различий между версиями применяются команды:

- **git diff HEAD** – отображает изменения в рабочих файлах по сравнению с последним коммитом;

```

PS C:\Users\enkeli\Desktop\ивт\demo> git diff HEAD
PS C:\Users\enkeli\Desktop\ивт\demo> git diff HEAD
diff --git a/example.txt b/example.txt
index 692dd53..09244dc 100644
--- a/example.txt
+++ b/example.txt
@@ -1,3 +1,3 @@
-example!
+example
 example!
 example!
\ No newline at end of file

```

Рисунок 11 – Сравнение рабочего каталога с последним коммитом командой `git diff HEAD`

На рисунке выше первый раз команда была выполнена до внесения изменений, поэтому никакого ответа от терминала не последовало. Далее файл был изменен (а именно удален один восклицательный знак), и выполнение команды показало это изменение.

- **`git diff --staged`** – показывает различия между подготовленными к коммиту файлами и последним коммитом.

Добавим внесенное выше изменение в staging area и посмотрим различия между подготовленным к новому коммиту файлом и последним коммитом. Видим те же различия.

```

PS C:\Users\enkeli\Desktop\ивт\demo> git add .
PS C:\Users\enkeli\Desktop\ивт\demo> git diff --staged
diff --git a/example.txt b/example.txt
index 692dd53..09244dc 100644
--- a/example.txt
+++ b/example.txt
@@ -1,3 +1,3 @@
-example!
+example
 example!
 example!
\ No newline at end of file

```

Рисунок 12 – Просмотр различий между подготовленными файлами и последним коммитом командой `git diff --staged`

Эти команды позволяют контролировать изменения и точно понимать, какие данные будут зафиксированы или уже были зафиксированы.

6. Управление изменениями

В процессе работы с репозиторием иногда возникает необходимость отменить подготовку файлов к коммиту, вернуть файл в предыдущее состояние или полностью удалить его из репозитория. Git предоставляет для этого отдельный набор команд.

Если файл был добавлен в область подготовки (staging area), но необходимо отменить это действие, используется команда **git reset <имя_файла>**. Она исключает файл из staging area, при этом сами изменения в рабочей директории остаются. Таким образом, изменения не теряются, но не попадут в ближайший коммит. Для демонстрации был создан и добавлен в область подготовки файл temp.txt. По выполнении команды git status в терминале можно увидеть, что оба файла готовы к коммиту.

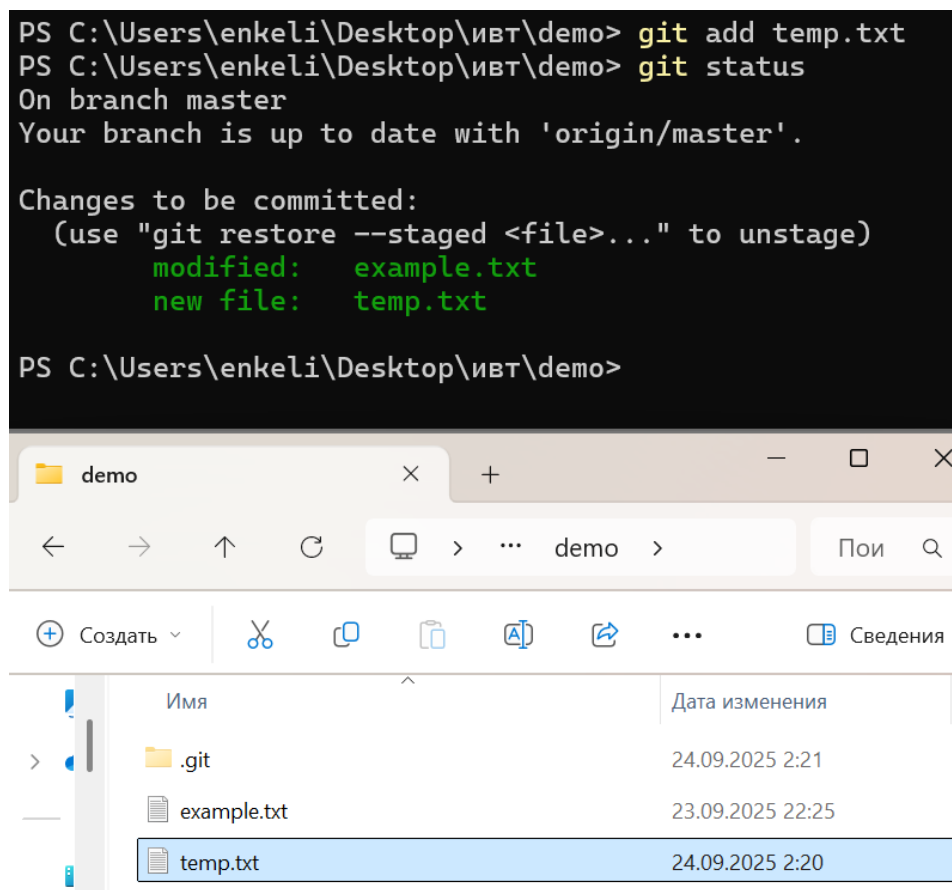


Рисунок 13 Добавление файла temp.txt в staging area и отображение подготовленных файлов командой git status

Выполним теперь команду git reset. Как видим, файл temp.txt был удален из области подготовки.

```

PS C:\Users\enkeli\Desktop\ивт\demo> git reset temp.txt

PS C:\Users\enkeli\Desktop\ивт\demo> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   example.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        temp.txt

```

Рисунок 14 – Отмена добавления файла temp.txt в staging area командой git reset

Для возврата содержимого файла к состоянию последнего коммита применяется команда **git checkout -- <имя_файла>**. В результате все несохраненные изменения в рабочей директории будут потеряны, а файл примет вид, зафиксированный в последнем коммите. Эта команда полезна, если нужно отменить локальные изменения и восстановить «чистую» версию. Команда работает только с неотслеживаемыми файлами (untracked), то есть с теми, которые не добавлены в staging area. На скриншотах выше видно, что файл example.txt находится в состоянии modified, так как в нем был убран один восклицательный знак. Сначала уберем этот файл из области подготовки, что проверим через команду git status. А затем вернем этот файл к состоянию последнего коммита, что тоже проверим через git status, а также увидим наглядно, открыв файл.

Файл	Измен
example.txt	example!
example.txt	example!
example.txt	example!

```

PS C:\Users\enkeli\Desktop\ивт\demo> git reset example.txt
Unstaged changes after reset:
M       example.txt
PS C:\Users\enkeli\Desktop\ивт\demo> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        temp.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\enkeli\Desktop\ивт\demo> git checkout -- example.txt
PS C:\Users\enkeli\Desktop\ивт\demo> git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        temp.txt

nothing added to commit but untracked files present (use "git add" to track)

```

Рисунок 15 – Возврат файла example.txt к состоянию последнего коммита командой git checkout --

Удаление файла из репозитория выполняется командой `git rm <имя_файла>`. В этом случае файл удаляется и из рабочей директории, и из индекса. Чтобы завершить удаление, необходимо выполнить коммит. Таким образом, история изменений будет отражать факт удаления файла.

```
PS C:\Users\enkelil\Desktop\ивт\demo> git add temp.txt
PS C:\Users\enkelil\Desktop\ивт\demo> git commit -m "Добавлен тем
p.txt"
[master c096ace] 1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 temp.txt
PS C:\Users\enkelil\Desktop\ивт\demo> git rm temp.txt
rm 'temp.txt'
PS C:\Users\enkelil\Desktop\ивт\demo> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    temp.txt

PS C:\Users\enkelil\Desktop\ивт\demo> git commit -m "Удален temp.
txt"
[master e0fffb2] 1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 temp.txt
PS C:\Users\enkelil\Desktop\ивт\demo> git log --summary
commit e0fffb20b2d6e2d1e88d3194fd821481ab89bf6b (HEAD -> master)
Author: Трофимцова Екатерина Евгеньевна <enkelilu@yandex.ru>
Date:   Wed Sep 24 02:42:50 2025 +0300

    Удален temp.txt

delete mode 100644 temp.txt
```

Рисунок 16 – Удаление файла temp.txt из репозитория с помощью команд `git rm` и `git commit`

7. Работа с ветками

Ветвление в Git позволяет параллельно развивать разные направления работы, изолировать экспериментальные изменения и безопасно интегрировать их в основную линию развития проекта. По умолчанию после инициализации репозитория используется основная ветка (в данной работе – master). Создание новой ветки фиксирует текущее состояние истории и позволяет вносить изменения, не затрагивая основную ветку.

Создать ветку можно командой `git branch <имя_ветки>`. Переключение между ветками выполняется командой `git checkout <имя_ветки>` (или эквивалентной современной командой `git switch <имя_ветки>`). Часто удобно совмещать создание и переключение: `git`

checkout -b <имя_ветки>. Текущий список веток отображается командой **git branch**, при этом активная ветка помечается символом *****.

На скриншоте ниже был отображен текущий список веток, который состоял только из ветки **master**. Затем одной командой была создана новая ветка и произведено переключение на нее, что опять же было проверено командой **git branch**.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git branch
* master
PS C:\Users\enkeli\Desktop\ивт\demo> git checkout -b readme-update
Switched to a new branch 'readme-update'
PS C:\Users\enkeli\Desktop\ивт\demo> git branch
  master
* readme-update
PS C:\Users\enkeli\Desktop\ивт\demo> |
```

Рисунок 17 – Создание новой ветки и переключение на неё командой **git checkout -b**

Далее в ветке **readme-update** был создан и закоммичен файл **README.md**, и выполнено обратное переключение на ветку **master**.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git add README.md
PS C:\Users\enkeli\Desktop\ивт\demo> git commit -m "Добавлен README.md"
[readme-update a4c7a45] 1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
PS C:\Users\enkeli\Desktop\ивт\demo> git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)
PS C:\Users\enkeli\Desktop\ивт\demo> git branch
* master
  readme-update
```

Рисунок 18 – Создание и коммит файла **README.md** в ветке **readme-update** и переключение обратно на ветку **master**

Слияние выполняется из той ветки, которую требуется обновить: находясь в **master**, команда **git merge <имя_ветки>** переносит изменения из указанной ветки в текущую.

```
PS C:\Users\enkeli\Desktop\ивт\demo> git merge readme-update
Updating e0fffb2..a4c7a45
Fast-forward
 README.md | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```

Рисунок 19 – Слияние ветки **readme-update** с основной веткой **master** командой **git merge**

После успешного слияния и при отсутствии необходимости в отдельной линии разработки ветку можно удалить: **git branch -d <имя_ветки>** (безопасное удаление, откажет, если ветка не слита) или **git branch -D <имя_ветки>** (принудительное удаление).

```
PS C:\Users\enkeli\Desktop\ивт\demo> git branch -d readme-update
Deleted branch readme-update (was a4c7a45).
PS C:\Users\enkeli\Desktop\ивт\demo> git branch
* master
```

Рисунок 20 – Удаление вспомогательной ветки readme-update после её слияния с основной

Таким образом, ветвление обеспечивает изоляцию изменений, упрощает командную работу и делает процесс интеграции управляемым и прозрачным.

8. Итоги работы в терминале с Git

Использование терминала для работы с системой контроля версий Git предоставляет полный доступ ко всем возможностям данного инструмента. Командная строка является первоочередным способом взаимодействия с Git: новые функции становятся доступными именно здесь, а не в графических клиентах. Такой подход обеспечивает максимальную гибкость и контроль над всеми этапами работы с репозиторием. Недостатком можно считать отсутствие наглядной визуализации истории коммитов и ветвлений без использования дополнительных опций (**git log --graph**) или сторонних инструментов, однако для большинства практических задач этого достаточно.

В процессе выполнения работы были освоены основные команды Git: инициализация репозитория (**git init**), проверка состояния (**git status**), добавление файлов (**git add**), фиксация изменений (**git commit**), просмотр истории (**git log**), работа с ветками (**git branch**, **git checkout**, **git merge**), взаимодействие с удаленным репозиторием (**git push**, **git pull**), а также команды для управления изменениями (**git reset**, **git checkout --**, **git rm**). Это позволило на практике изучить базовые принципы функционирования Git, понять логику отслеживания изменений и закрепить навыки работы с системой контроля версий.

Таким образом, терминал можно охарактеризовать как универсальный и надежный инструмент для работы с Git. Он подходит пользователям, которые стремятся к полному контролю над процессом, готовы осваивать

синтаксис команд и хотят использовать все возможности системы без ограничений, накладываемых графическими оболочками.