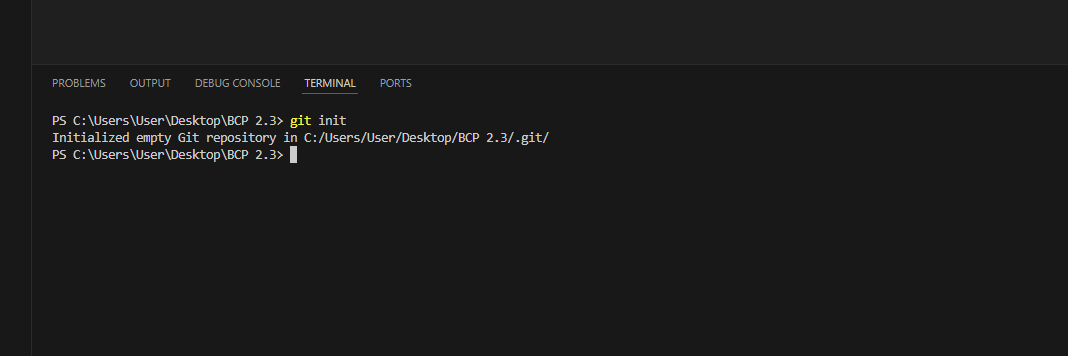
**ВСР 2.3. Встроенные средства IDE**

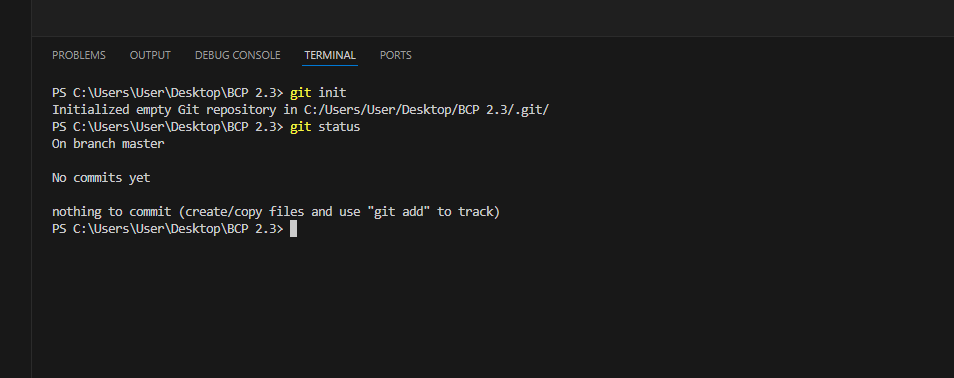
**Работа с командами Git:** Visual Studio Code

**1. Инициализация репозитория**

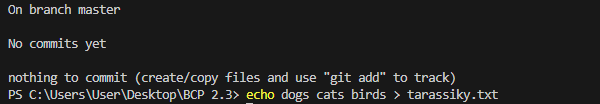
Для создания нового репозитория в системе контроля версий Git необходимо выполнить команду git init. Данная команда инициирует новый репозиторий, создавая скрытую директорию .git, в которой хранится вся информация о версиях, объектах и ссылках, необходимых для отслеживания истории изменений в проекте. Эта директория служит основой для работы с системой контроля версий, обеспечивая сохранение всех данных о коммитах и состоянии файлов.

1. **Статус репозитория**

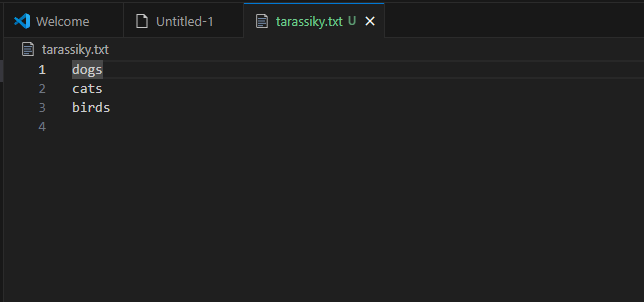
Для получения информации о текущем состоянии репозитория следует использовать команду git status. Эта команда предоставляет сведения о подготовленных, неподготовленных и неотслеживаемых файлах. Она позволяет разработчикам быстро оценить, какие изменения были внесены, какие файлы находятся в стадии подготовки к коммиту, а также выявить новые файлы, которые еще не добавлены в систему контроля версий.



Создание текстового файла с каким-нибудь текстом

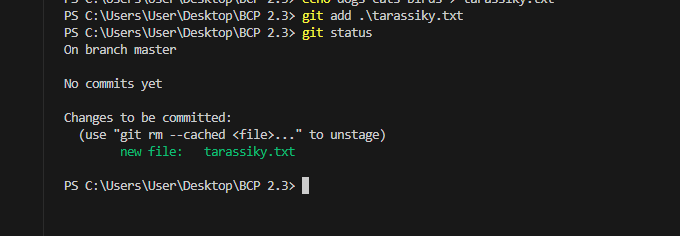


Получаем файл



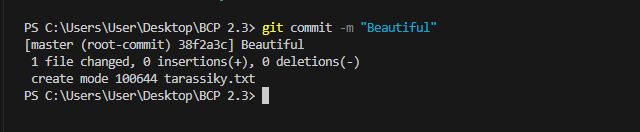
1. **Добавление файлов в область подготовленных файлов**

Для добавления конкретного файла в область подготовленных файлов используется команда git add, за которой следует указание имени файла. После выполнения данной команды целевой файл будет помещен в индекс (область подготовленных файлов).



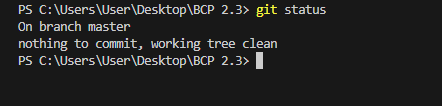
Рекомендуется сразу после этого проверить статус репозитория с помощью git status, чтобы убедиться, что файл был успешно добавлен в область подготовки.

1. **Внесение изменений однострочным сообщением или через редактор**



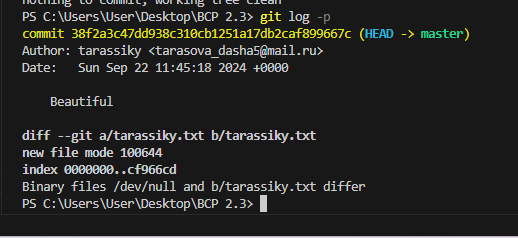
Для создания коммита с описанием изменений можно воспользоваться командой git commit с параметром -m, за которым следует текст сообщения в кавычках. Это позволяет документировать внесенные изменения непосредственно во время создания коммита.

И сразу проверяем статус репозитория



Файл успешно зафиксирован

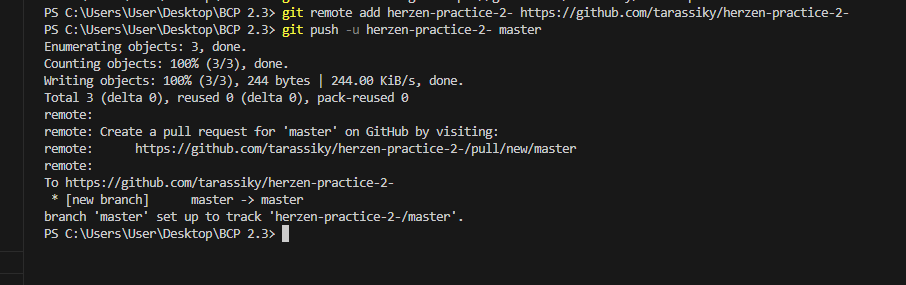
1. **Просмотр истории коммитов с изменениями**

Для анализа истории изменений в репозитории используется команда git log. Она отображает список последних коммитов в порядке их выполнения, что позволяет разработчикам отслеживать прогресс проекта и изменения, внесенные на каждом этапе разработки. Добавление флага -p к команде git log позволяет получить более детальную информацию о внесенных изменениях в каждом файле, что способствует более глубокому пониманию эволюции проекта.

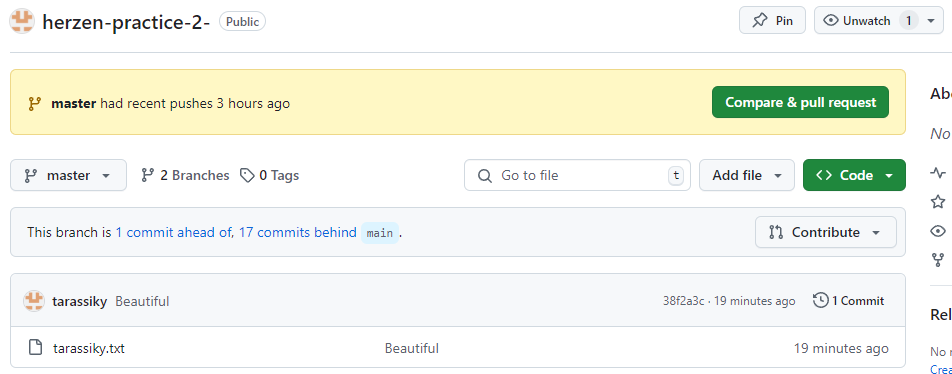
1. **Добавление удаленного репозитория**

Чтобы связать локальный репозиторий с удаленным, необходимо использовать команду git remote add, указав имя и URL удаленного репозитория. Это действие создает ссылку на удаленный репозиторий, что позволяет выполнять операции синхронизации между локальной и удаленной версиями проекта.2024-09-22_11-49-22

1. **Отправка изменений в удаленный репозиторий**

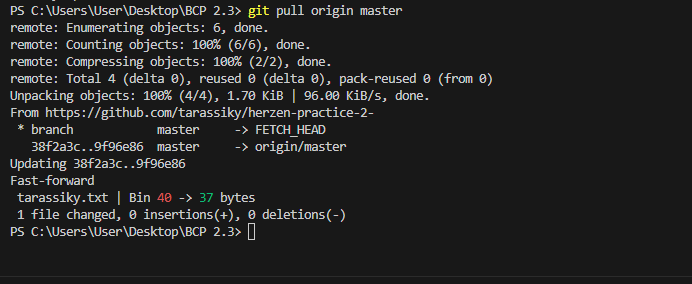
Для отправки локальных изменений в удаленный репозиторий используется команда git push, к которой добавляются имя удаленного репозитория и ветка, в которую будут отправлены изменения. Если отправляется новая ветка, необходимо использовать флаг -u, который устанавливает связь между локальной и удаленной веткой, упрощая последующие операции синхронизации.

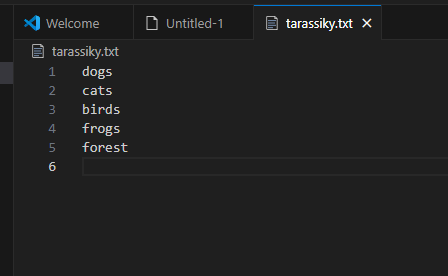
Далее, перейдя в GitHub, видим, что наш файл сохранился



1. **Получение изменений из удалённого репозитория**

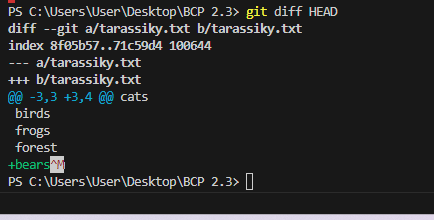
Команда git pull применяется для загрузки изменений из удаленного репозитория. Она выполняет две операции: сначала загружает обновления с указанного удаленного репозитория, а затем объединяет их с текущей локальной веткой. Это позволяет поддерживать актуальность локальной копии проекта относительно его удаленной версии и гарантирует синхронизацию всех внесенных изменений.





1. **Просмотр изменений до коммита**

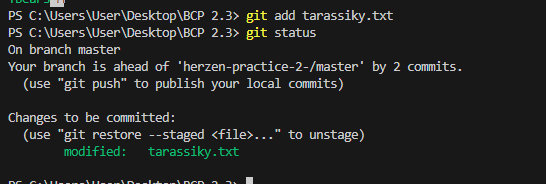
Для анализа изменений, внесённых в репозиторий до их коммита, можно воспользоваться командой git diff. Этот инструмент позволяет детально рассмотреть список модификаций, произведённых с момента последнего коммита.



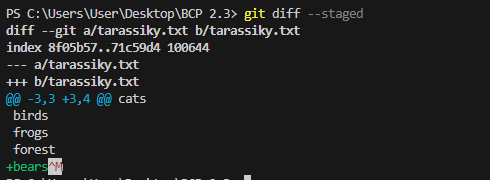
Например, если в файл tarassiky.txt была добавлена новая строка (bears), но в текущем состоянии репозитория этот файл не содержит указанной строки, то при выполнении команды git diff в терминале отобразятся все изменения, касающиеся данного файла.

Для просмотра изменений, которые были подготовлены к коммиту, следует использовать флаг --staged. Таким образом, если мы добавим файл tarassiky.txt в область подготовленных файлов с помощью команды git add, а затем проверим статус репозитория с помощью git status, мы увидим, что файл находится в состоянии "подготовлен к коммиту". Далее, применив команду git diff --staged, мы сможем увидеть изменения, которые будут зафиксированы в следующем коммите.

Перед применением команды я добавила в файл строчку, а в репозитории этот же файл не содержит данную строчку, поэтому в терминале мы можем увидеть изменения в файлах с момента последнего коммита.

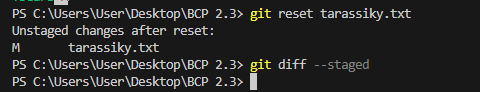


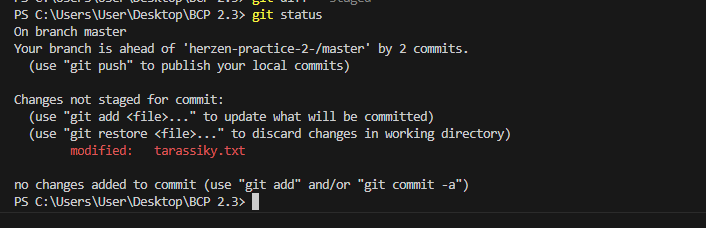
Для просмотра подготовленных изменений необходимо добавить флаг ”–staged”.

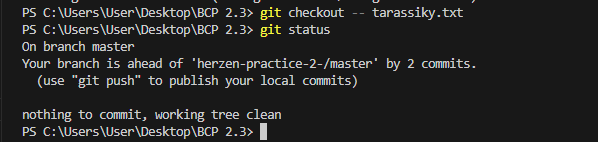


1. **Отмена подготовленных и неподготовленных изменений**

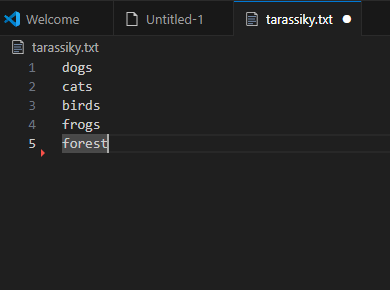
Восстановление состояния подготовленного файла в рабочем дереве возможно с использованием команды git reset. Если после выполнения этой команды мы снова применим git diff --staged, терминал вернёт пустое значение, указывая на отсутствие подготовленных изменений.



Чтобы отменить неподготовленные изменения, можно воспользоваться командой git checkout -- <имя\_файла>. Прежде чем выполнять эту операцию, рекомендуется проверить статус репозитория с помощью git status. 

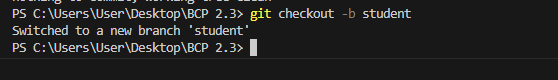
В результате выполнения команды git checkout изменённое состояние файла будет возвращено к состоянию последнего коммита. 

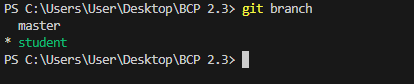
После этого, если мы снова проверим статус репозитория, изменённый файл tarassiky.txt перестанет отображаться, а слово "bears" исчезнет из его содержимого.



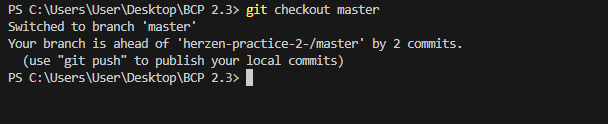
1. **Создание новой ветки и переход в неё**

Создание новой ветки осуществляется с помощью команды git branch <имя\_ветки>. Однако Git не переключается на новую ветку автоматически. Для этого необходимо использовать команду git checkout -b <имя\_ветки>, что создаст новую ветку и сразу же переключит на неё контекст.



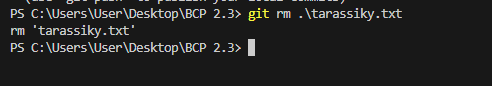
Чтобы определить текущую ветку, достаточно выполнить команду git branch, которая отобразит список всех веток с указанием активной.

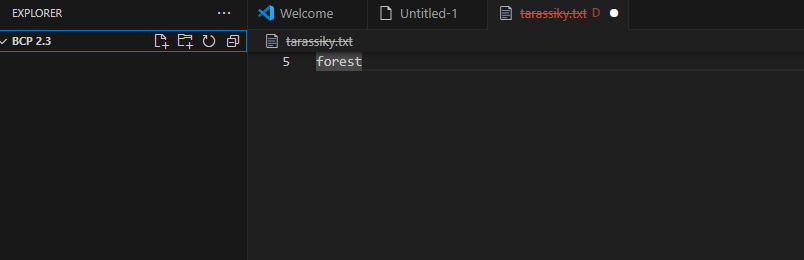
Переключение между существующими ветками осуществляется с помощью команды git checkout <имя\_ветки> без дополнительных флагов. При создании новой ветки она наследует все файлы и изменения от ветки, на основе которой она была создана (например, от ветки master).



1. **Удаление файлов**

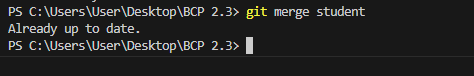
Удаление файлов из текущего рабочего дерева выполняется с использованием команды git rm <имя\_файла>. Также можно применять шаблоны для удаления нескольких файлов одновременно; например, использование маски \*.txt приведёт к удалению всех файлов с расширением .txt. После выполнения команды для удаления файла tarassiky.txt, он будет исключён из рабочего дерева и подготовлен к коммит.





Мы успешно удалили файл tarassiky.txt.

1. **Слияние двух веток**

Объединение двух веток осуществляется с помощью команды git merge <имя\_ветки>, что приводит к слиянию указанных изменений с текущей активной веткой. Эта операция позволяет интегрировать изменения из одной ветки в другую, обеспечивая актуальность и согласованность кода.

1. **Удаление ветки**

Удаление ненужной ветки выполняется с использованием команды git branch -d <имя\_ветки>. Перед выполнением данной операции рекомендуется проверить текущую активную ветку с помощью команды git branch. Например, если у нас есть две ветки — master и student, и мы решаем удалить ветку student, после выполнения команды удаления останется только ветка master. Это позволяет поддерживать порядок в репозитории и избегать накопления неиспользуемых веток.



