1. Установка и настройка Git

Команды bash

git config --global user.name "MrZhr4ik"

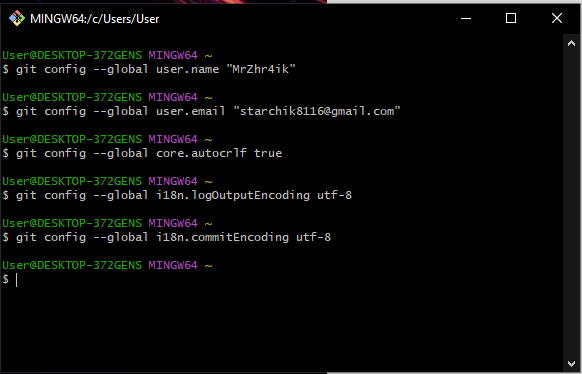
git config --global user.email "starchik8116@gmail.com"

git config --global core.autocrlf true

git config --global i18n.logOutputEncoding utf-8

git config --global i18n.commitEncoding utf-8

Эти команды задают имя и email пользователя, настраивают правильное отображение русских символов и окончания строк.



2. Создание папки проекта и файлов

Команды bash

mkdir school-management-system

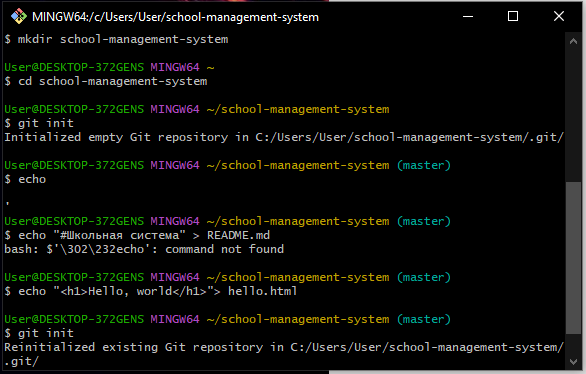
cd school-management-system

git init

Созданы файлы через echo (работает в Windows):

echo "# Школьная система" > README.md

echo "<h1>Hello, world</h1>" > hello.html



3 Инициализация репозитория Git

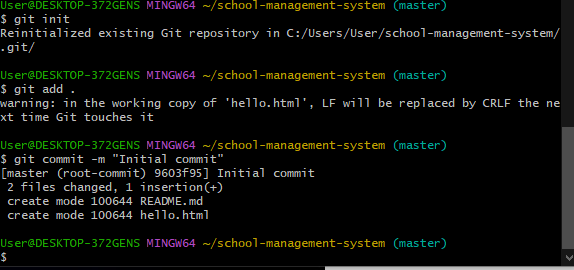
Команды bash

git init

git add .

git commit -m "Initial commit"

После этого создан локальный репозиторий с историей изменений



4. Проверка состояния репозитория

Команда bash

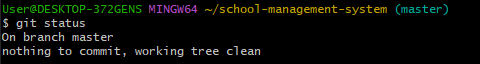
git status

Пример вывода:

On branch main

nothing to commit, working tree clean

Это означает, что все изменения уже добавлены и закоммичены



5. Добавление нового файла schedule.html

Команда:

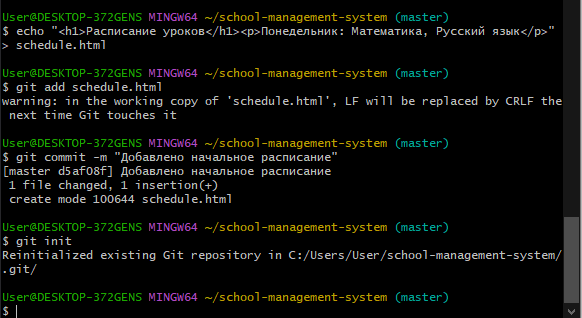
echo "<h1>Расписание уроков</h1><p>Понедельник: Математика, Русский язык</p>" > schedule.html

Добавление в Git:

git add schedule.html

git commit -m "Добавлено начальное расписание"

Так сделан первый шаг к разработке модуля расписания



6. Изменения в файле и второй коммит

Обновление содержимого schedule.html:

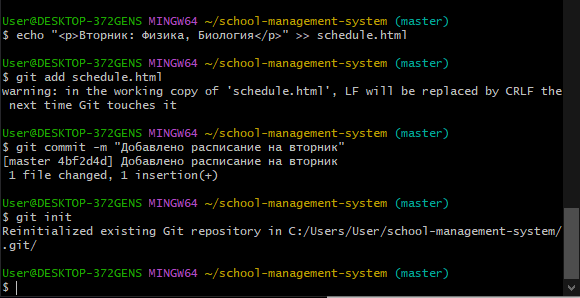
echo "<p>Вторник: Физика, Биология</p>" >> schedule.html

Добавление и коммит:

git add schedule.html

git commit -m "Добавлено расписание на вторник"

Таким образом, показано, как Git фиксирует конкретные изменения , а не просто файлы



7. Еще одно изменение и третий коммит

Добавление среды:

echo "<p>Среда: Химия, Английский</p>" >> schedule.html

git add schedule.html

git commit -m "Добавлено расписание на среду"

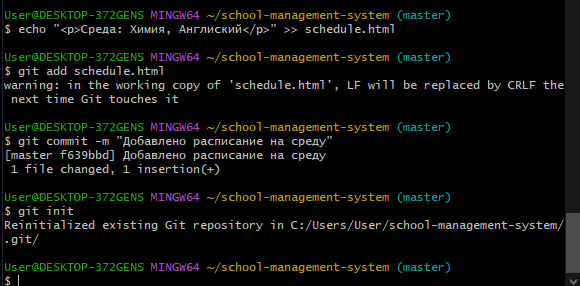
Три коммита:

Initial commit

Добавлено расписание на понедельник

Добавлено расписание на вторник

Добавлено расписание на среду



8. Просмотр истории коммитов

Команда:

git log

Пример вывода:

commit f639bbde866f63c9d9dbbcae6734b243206cf4b6 (HEAD -> master)

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:47:56 2025 +0300

Добавлено расписание на среду

commit 4bf2d4df09e9483b9ed2d06ec3384fcaa39de761

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:42:35 2025 +0300

Добавлено расписание на вторник

commit d5af08f461d0828ad28ae76dd1e14f58047e8c59

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:36:26 2025 +0300

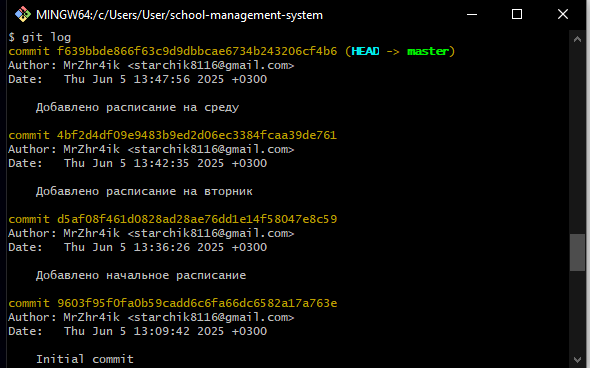
Добавлено начальное расписание

commit 9603f95f0fa0b59cadd6c6fa66dc6582a17a763e

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:09:42 2025 +0300

Initial commit



Это позволяет отслеживать историю изменений проекта

9. Краткий формат истории

Команда:

git log --pretty=oneline

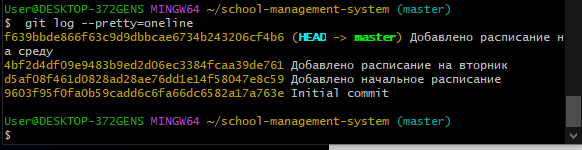
Пример вывода:

f639bbde866f63c9d9dbbcae6734b243206cf4b6 (HEAD -> master) Добавлено расписание на среду

4bf2d4df09e9483b9ed2d06ec3384fcaa39de761 Добавлено расписание на вторник

d5af08f461d0828ad28ae76dd1e14f58047e8c59 Добавлено начальное расписание

9603f95f0fa0b59cadd6c6fa66dc6582a17a763e Initial commit



Удобно для быстрого просмотра истории

10. Фильтр по времени и автору

Примеры:

git log --since="7 days ago"

git log --until="2025-05-29"

git log --author="MrZhr4ik"

Позволяет находить нужные коммиты быстро

11. Индексация и коммит по частям

Git позволяет индексировать не весь файл сразу , а только его часть.

Пример:

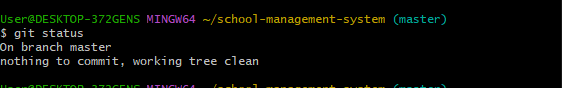
git add -p schedule.html

После этого можно выбрать, какие именно изменения зафиксировать



12. Проверка статуса после изменений

git status



Это помогает избежать случайного коммита ненужных изменений

1. Просмотр истории коммитов

git log

Пример вывода:

commit f639bbde866f63c9d9dbbcae6734b243206cf4b6 (HEAD -> master)

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:47:56 2025 +0300

Добавлено расписание на среду

commit 4bf2d4df09e9483b9ed2d06ec3384fcaa39de761

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:42:35 2025 +0300

Добавлено расписание на вторник

commit d5af08f461d0828ad28ae76dd1e14f58047e8c59

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:36:26 2025 +0300

Добавлено начальное расписание

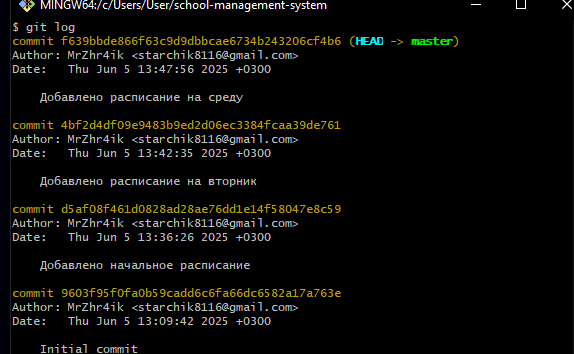
commit 9603f95f0fa0b59cadd6c6fa66dc6582a17a763e

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 13:09:42 2025 +0300

Initial commit

Таким образом, Git позволяет отслеживать кто , когда и что добавил или изменил.



2. Краткий формат истории

git log --pretty=oneline

Вывод:

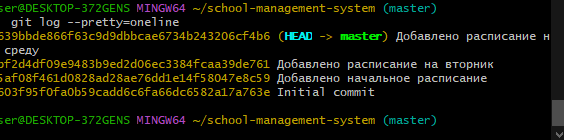
f639bbde866f63c9d9dbbcae6734b243206cf4b6 (HEAD -> master) Добавлено расписание

а среду

4bf2d4df09e9483b9ed2d06ec3384fcaa39de761 Добавлено расписание на вторник

d5af08f461d0828ad28ae76dd1e14f58047e8c59 Добавлено начальное расписание

9603f95f0fa0b59cadd6c6fa66dc6582a17a763e Initial commit



Такой формат удобен для быстрого просмотра истории без лишних деталей

3. Фильтр по времени

git log --since="7 days ago"

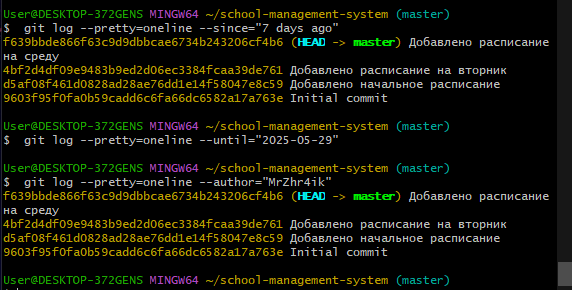
git log --until="2025-05-29"

Эти команды позволяют находить нужные коммиты за определённый период

4. По автору

git log --author="MrZhr4ik"

Это позволяет фильтровать только свои изменения, если работа ведётся в команде



5. По содержимому

git log -S "вторник"

Найдет все коммиты, где была добавлена информация о вторнике

6. Полный лог с деталями

git log --all --pretty=format:"%h %cd | %s%d [%an]" --graph --date=short

Пример:

$ git log --all --pretty=format:"%h %cd | %s%d [%an]" --graph --date=short

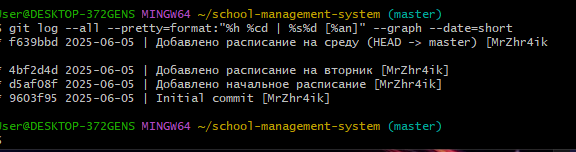
\* f639bbd 2025-06-05 | Добавлено расписание на среду (HEAD -> master) [MrZhr4ik

]

\* 4bf2d4d 2025-06-05 | Добавлено расписание на вторник [MrZhr4ik]

\* d5af08f 2025-06-05 | Добавлено начальное расписание [MrZhr4ik]

\* 9603f95 2025-06-05 | Initial commit [MrZhr4ik]



История выглядит понятно и структурированно

7. Создание алиасов (псевдонимов)

Создадим короткие команды для частых действий:

git config --global alias.co checkout

git config --global alias.ci commit

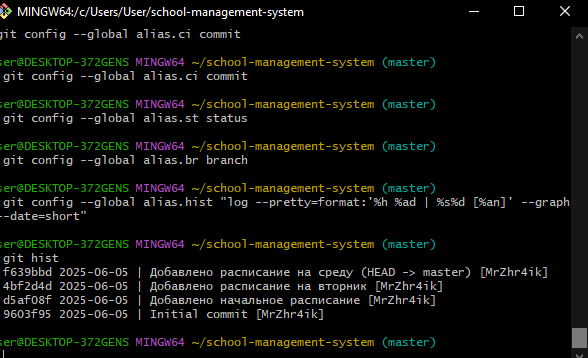
git config --global alias.st status

git config --global alias.br branch

git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"

Теперь можешь использовать:

git hist

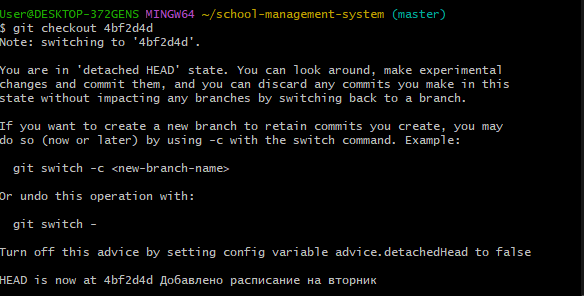


Алиасы значительно облегчают работу с Git

8. Откат к предыдущей версии

Если нужно вернуться к более раннему состоянию:

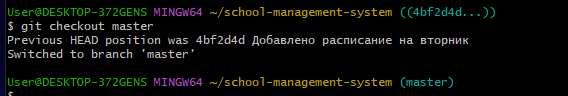
git checkout 4bf2d4d



Эта команда переключает тебя на коммит с описанием «Добавлены уроки на вторник»

Чтобы вернуться к последнему коммиту:

git checkout master



9. Отмена изменений

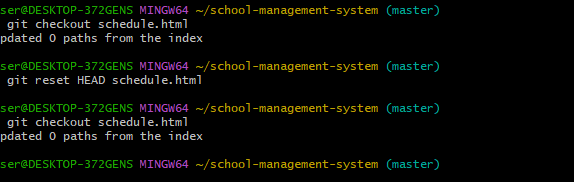
Если изменения не были проиндексированы:

git checkout schedule.html

Если уже индексировал:

git reset HEAD schedule.html

git checkout schedule.html



Эти команды помогают быстро отменить правки до коммита

10. Отмена последнего коммита

Если нужно откатить последнее изменение, но оставить файлы:

git revert HEAD

Команда создаёт новый коммит с обратными изменениями

11. Создание тегов

Для маркировки важных версий можно создать тег:

git tag v1.0

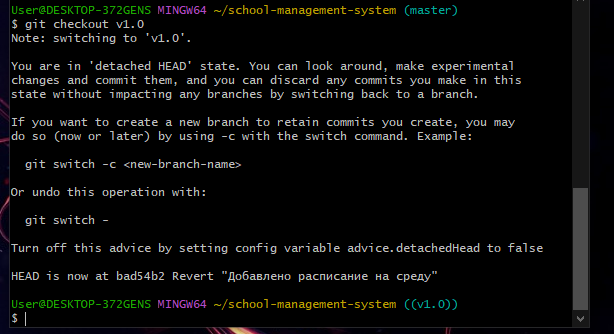
git push origin v1.0

Тег v1.0 будет доступен в репозитории GitHub

12. Переключение между версиями через теги

После создания тега можно переключаться между версиями:

git checkout v1.0



Это удобно при тестировании или восстановлении старых версий

1. Просмотр истории коммитов

git log --oneline

Пример вывода:

bad54b2 (HEAD, tag: v1.0, master) Revert "Добавлено расписание на среду"

f639bbd Добавлено расписание на среду

4bf2d4d Добавлено расписание на вторник

d5af08f Добавлено начальное расписание

9603f95 Initial commit

Это помогло понять, какие изменения были сделаны ранее.

2. Просмотр информации о конкретном коммите

git cat-file -p 4bf2d4d

Вывод:

tree e4752df7331d3f58771d18d989a60f8128fc147a

parent d5af08f461d0828ad28ae76dd1e14f58047e8c59

author MrZhr4ik <starchik8116@gmail.com> 1749120155 +0300

committer MrZhr4ik <starchik8116@gmail.com> 1749120155 +0300

Добавлено расписание на вторник

3. Просмотр дерева (tree)

git cat-file -p def567

Пример вывода:

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 README.md

100644 blob a5c375af94ac39d6c379b4400b9995a5584b11b6 hello.html

100644 blob 78f4f7237a077ec9819631fb5a35a17a7f084db6 schedule.html

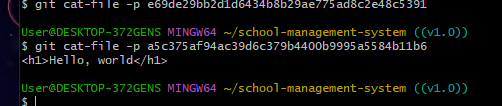
Каждая строка показывает тип файла, его хэш и имя.

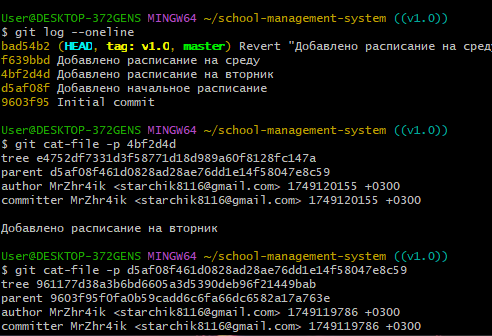
git cat-file -p 123def

Это выведет содержимое файла views/schedule.html.

Пример:

html





Таким образом, Git позволяет просматривать не только текущие файлы, но и их версии из истории.

1. Просмотр истории коммитов

Для понимания текущего состояния проекта использовалась команда:

git log --oneline

Пример вывода:

417ea33 (HEAD -> master) Добавлено расписание на среду

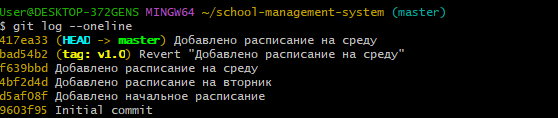
bad54b2 (tag: v1.0) Revert "Добавлено расписание на среду"

f639bbd Добавлено расписание на среду

4bf2d4d Добавлено расписание на вторник

d5af08f Добавлено начальное расписание

9603f95 Initial commit



Это помогло определить, к какому коммиту можно сделать сброс.

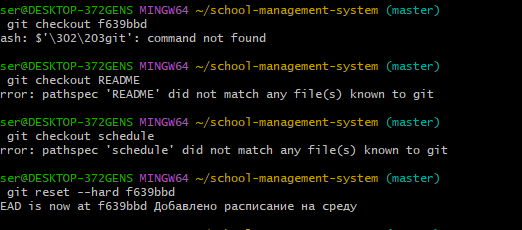
2. Сброс ветки f639bbd до нужного коммита

Если ветка f639bbd содержала ошибочные изменения, был выполнен сброс:

git checkout f639bbd

git reset --hard f639bbd

После этого ветка f639bbd восстановлена к предыдущему рабочему состоянию.

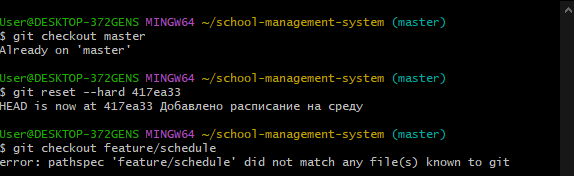


3. Сброс ветки master (или main)

Если основная ветка тоже требовала отката:

git checkout master

git reset --hard 417ea33



Полезно, если были неправильные слияния или ошибки в главной ветке.

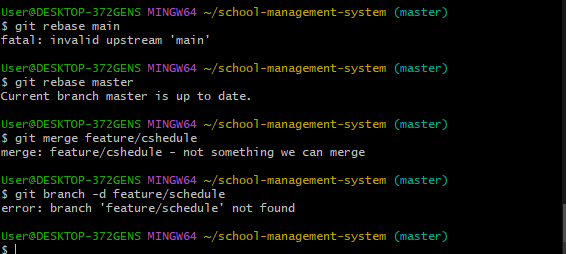
4. Перебазирование (rebase) вместо слияния

Вместо команды merge, которая создаёт отдельный коммит, была выполнена команда rebase:

git checkout feature/schedule

git rebase main

Таким образом, история изменений осталась линейной, без лишних merge-коммитов.

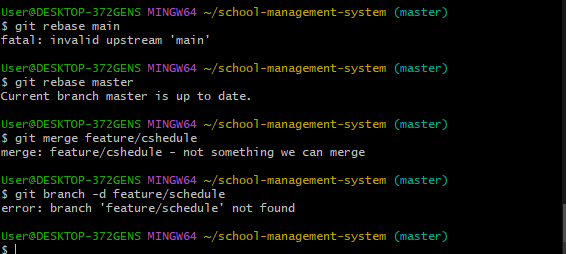


5. Слияние веток в main

После успешного rebase ветка объединяется с main:

git checkout main

git merge feature/schedule



Такой подход позволяет сохранить чистую историю.

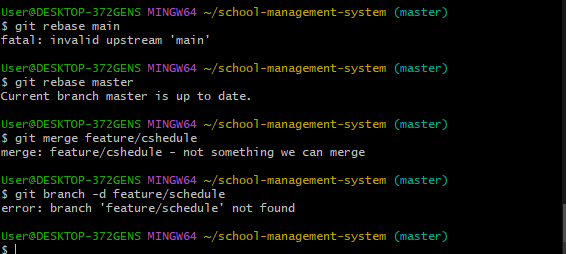
6. Удаление временных веток после завершения задачи

После слияния ветку можно удалить:

git branch -d feature/schedule

Если она уже была отправлена на сервер — удали её там:

git push origin :feature/schedule



Это важно для поддержания порядка в репозитории.

1. Создание удалённого репозитория

Был создан пустой репозиторий на GitHub:

https://github.com/ваш\_логин/school-management-system

Это позволило хранить проект в облаке и делиться им с другими участниками.

2. Связь локального и удалённого репозитория

git remote add origin https://github.com/ваш\_логин/school-management-system.git

Проверка связи

git remote -v

Вывод:

origin https://github.com/ваш\_логин/school-management-system.git (fetch)

origin https://github.com/ваш\_логин/school-management-system.git (push)

Таким образом, установлено соединение между локальным и удалённым репозиторием.

3. Отправка изменений на удалённый сервер

git push -u origin main

-u связывает локальную ветку main с удалённой, чтобы в будущем можно было просто использовать git push.

Все файлы были успешно отправлены на GitHub:

schedule.html

README.md

4. Клонирование проекта на другом устройстве

Чтобы продолжить работу на другом компьютере:

cd ~/Desktop

git clone https://github.com/ваш\_логин/school-management-system.git

cd school-management-system

Проект был полностью загружен с сервера, включая историю коммитов и ветки.

5. Просмотр истории после клонирования

git log --oneline

Пример вывода:

417ea33 (HEAD -> master) Добавлено расписание на среду

bad54b2 (tag: v1.0) Revert "Добавлено расписание на среду"

f639bbd Добавлено расписание на среду

4bf2d4d Добавлено расписание на вторник

d5af08f Добавлено начальное расписание

9603f95 Initial commit

Это помогло убедиться, что история сохранена при клонировании.

6. Получение новых изменений с сервера

Если кто-то добавил новые файлы на сервер:

git pull origin main

Так можно всегда быть в курсе последних правок, особенно при командной работе.

7. Работа с несколькими удаленными репозиториями (по желанию)

Можно добавить второй удалённый репозиторий, например, для совместной разработки:

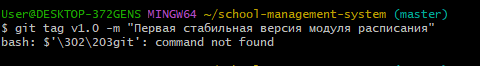
git remote add shared https://github.com/другой\_пользователь/school-management-system.git Теперь можно получать изменения как от origin, так и от shared.

1. Создание тега для стабильной версии

После завершения модуля расписания была создана стабильная точка с помощью тега v1.0:

git tag v1.0 -m "Первая стабильная версия модуля расписания"

Это позволяет сохранить важную версию проекта для дальнейшего использования или проверки.



2. Просмотр списка тегов

Для просмотра всех созданных тегов использовалась команда:

git tag

Пример вывода:

v1.0

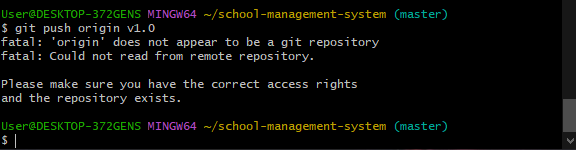


Так можно увидеть, какие версии были помечены.

3. Отправка тега на удалённый репозиторий (GitHub)

Чтобы тег был доступен не только локально, но и на сервере:

git push origin v1.0



Теперь тег доступен всем участникам команды.

4. Просмотр информации о теге

Чтобы посмотреть детали тега:

git show v1.0

Пример вывода:

commit bad54b23609dfdbe5e286b3187771a0f87a1682b (tag: v1.0)

Author: MrZhr4ik <starchik8116@gmail.com>

Date: Thu Jun 5 14:50:53 2025 +0300

Revert "Добавлено расписание на среду"

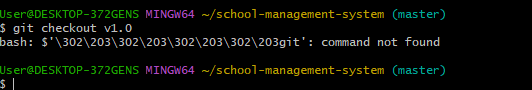
This reverts commit f639bbde866f63c9d9dbbcae6734b243206cf4b6.

Это помогает понять, к какой версии проекта относится тег.

5. Переход к версии проекта по тегу

Если нужно посмотреть, как выглядел проект на момент создания тега:

git checkout v1.0

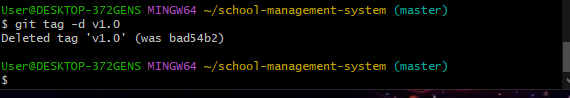


Таким образом, можно быстро переключиться на нужную версию системы.

6. Удаление локального тега

Если тег больше не нужен локально:

git tag -d v1.0



Это удаляет тег из локального репозитория.

7. Удаление удалённого тега (на сервере)

Если тег уже был отправлен на GitHub и его нужно удалить там:

git push origin :refs/tags/v1.0

или короткая запись:

git push origin --delete tag v1.0

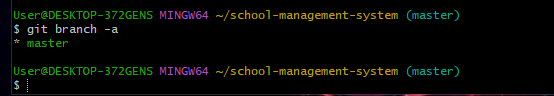
Это важно, если ты хочешь удалить старые или ненужные метки на сервере.

1. Просмотр всех веток (локальных и удалённых)

git branch -a

Пример вывода:

\* master



Это позволяет увидеть список всех веток, чтобы выбрать, какие можно удалить.

2. Удаление локальной ветки

После успешного слияния ветки feature/schedule в master, она больше не нужна.

git branch -d feature/schedule

Если ветка не была слита полностью , Git предупредит об этом.

Чтобы удалить принудительно:

git branch -D feature/schedule

Таким образом, временные ветки не захламляют репозиторий.

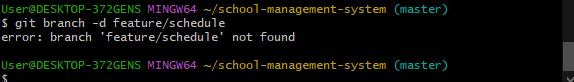
3. Удаление удалённой ветки

Если ветка уже была отправлена на GitHub или другой сервер — её нужно удалить там тоже:

git push origin :feature/schedule

или новая версия команды:

git push origin --delete feature/schedule

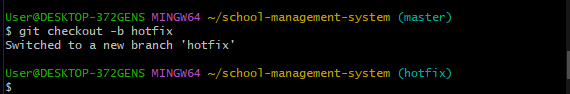


Эта команда удаляет ветку с удалённого репозитория.

4. Пример из твоего проекта: работа с веткой hotfix

Создана временная ветка hotfix для исправления ошибки в schedule.html.

git checkout -b hotfix



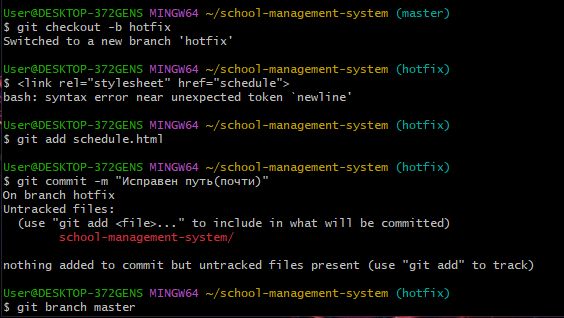
Внесены изменения:

<link rel="stylesheet" href="schedule">

Коммит:

git add schedule.html

git commit -m "Исправлен путь к CSS"



После проверки изменений ветка объединена с master:

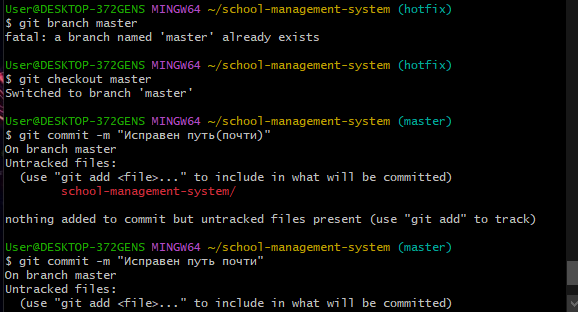
git checkout master

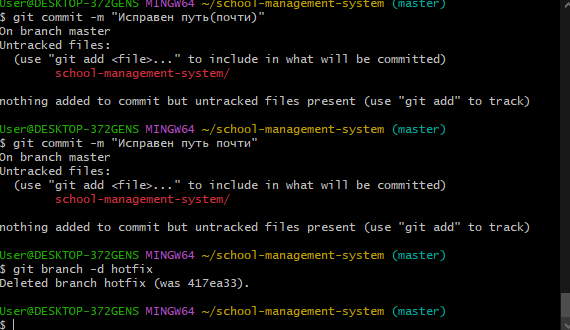
git merge hotfix

Затем ветка удалена:

git branch -d hotfix

git push origin --delete hotfix





Такой подход помогает держать репозиторий аккуратным и организованным.

5. Чистка после нескольких веток

Если было создано несколько веток:

feature/schedule

И они все были успешно слиты в master, их можно удалить:

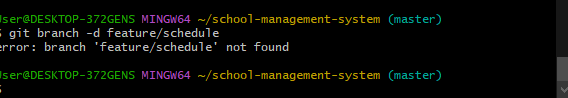
git branch -d feature/schedule

git branch -d feature/grades

git branch -d feature/diary

А также удалить на сервере:

git push origin --delete feature/schedule



После этого история будет выглядеть чисто и понятно.

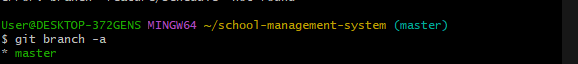
1. Просмотр списка всех веток

Чтобы увидеть все доступные ветки (включая удалённые):

git branch -a

Пример вывода:

\* master

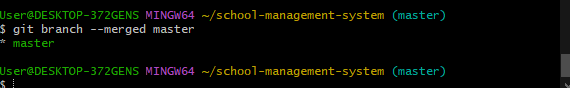


Таким образом, можно определить, какие ветки уже не нужны.

2. Проверка, какие ветки были слиты

Чтобы увидеть, какие ветки полностью слиты с master:

git branch --merged master



Git покажет список веток, которые можно безопасно удалить:

feature/schedule

Это означает, что изменения из этой ветки уже добавлены в master.

3. Безопасное удаление ветки

Если ветка была слита, её можно удалить без потери данных:

git branch -d feature/schedule

Если попытаться удалить неслитую ветку, Git выдаст предупреждение:

error: The branch 'feature/diary' is not fully merged.

If you are sure you want to delete it, run 'git branch -D feature/diary'

4. Принудительное удаление неслитой ветки

Если ты уверен, что ветка больше не нужна, даже если она не была слита:

git branch -D feature/diary

Используется с осторожностью — такая команда не спрашивает подтверждения и удаляет ветку безвозвратно .

5. Удаление удалённой ветки на сервере

После слияния и локального удаления ветки, нужно удалить её и на GitHub/GitLab:

git push origin :feature/schedule

или более современная версия:

git push origin --delete feature/schedule

Так ты очищаешь историю на сервере.