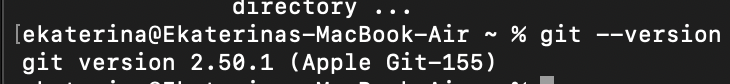
**Лабораторная работа 6**

**Системы контроля версий. Основы работы в Git**

**Цель работы:** ознакомление с основными возможностями системы контроля версий Git.

**1)** Проверьте установленную версию и зафиксируйте ее в отчете:

**2)** Создайте локальный репозиторий. Опишите последовательность действий в отчете.

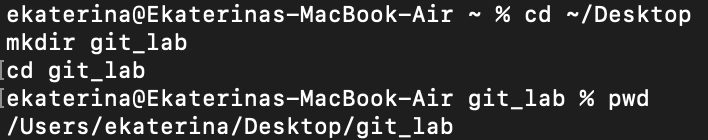
Создаю специальную папку для работы:

cd ~/Desktop

mkdir git\_lab

cd git\_lab

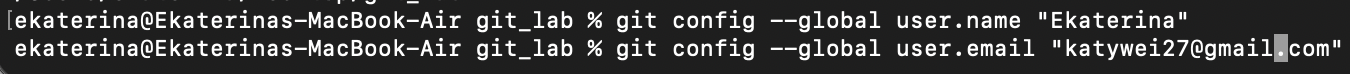
Проверю правильно ли я создала с помощью команды pwd.

Я должна находится: /Users/ekaterina/Desktop/git\_lab

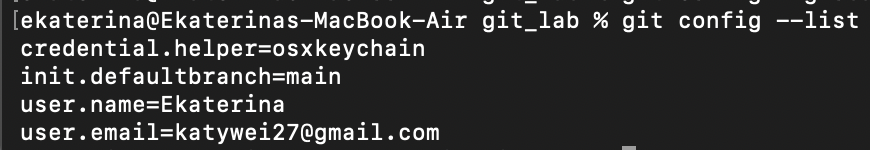
Все сделано правильно.

**3)** Выполните конфигурацию гит. Для этого настройте конфигурационный файл, используя:

$ git config --global user.name "XXX"  
$ git config --global user.email XXX@XXX.com



**4)** Проверить настройки конфигурационного файла:



**5)** Получите справку по **Git**. Ознакомьтесь со списком командам. Какой командой для получения справки вы воспользовались?

**Основные команды** Git

git init – создание репозитория

git add <имена файлов> – добавляет файлы в индекс

git commit – выполняет коммит проиндексированных файлов в репозиторий

git status – показывает какие файлы изменились между текущей стадией и HEAD. Файлы разделяются на 3 категории: ***новые*** файлы, ***измененные*** файлы, ***добавленные*** новые файлы

git checkout < SHA1 или метка> – получение указанной версии файла

git push – отправка изменений в удаленный репозиторий

git fetch – получение изменений из удаленного репозитория

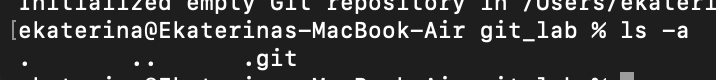
git clone <remote url> – клонирование удаленного репозитория себе

Для получения справки была использована команда git help. Она выводит общее описание Git и список основных команд, а также позволяет открыть подробную справку по любой команде.  


**6)** Инициализируйте локальный репозиторий в рабочей папке (см. лекцию), где будет он будет размещен. Исследуйте изменения, которые произошли в папке. Зафиксируйте их в отчете.

В папке git\_lab: прописала команду git init

Затем проверила появление скрытой папки: ls -a (скрытая служебная директория .git, содержит: настройки Git (config), данные о ветках (refs), историю объектов (objects), информацию о состоянии репозитория). Эта директория отвечает за хранение всей истории версий проекта.

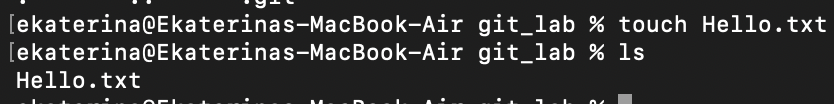


**7)** С помощью блокнота создайте в папке текстовый файл Hello.txt.

В той же папке прописываю: touch Hello.txt

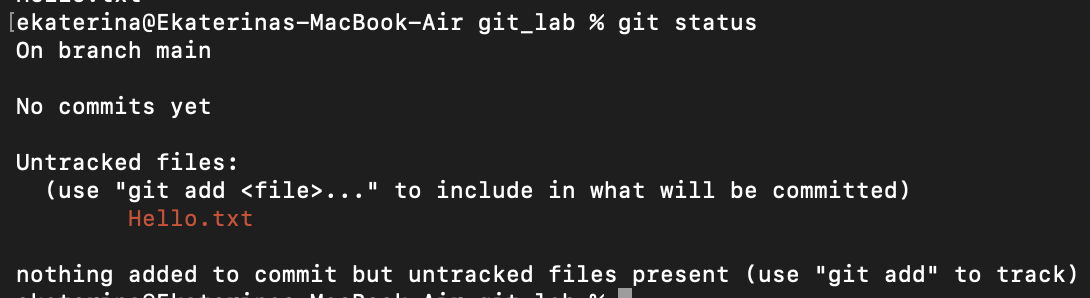
Проверяю с помощью команды ls

Должно появиться: Hello.txt



**8)** Научитесь получать информацию о статусе (status) репозитория. Какая информация отображается в терминальном окне GitBash? Прокомментируйте вывод команды в отчете.

Выполняю: git status ­— команда показывает текущее состояние рабочей директории и индекса.



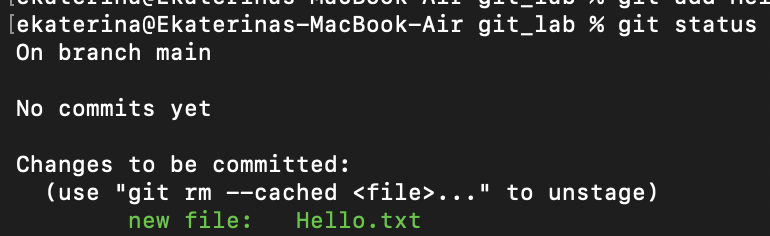
— В выводе отображается, что файл Hello.txt находится в разделе *Untracked files*, то есть Git его видит, но ещё не отслеживает.  
— Git также рекомендует использовать команду git add для индексации.

**9)** Проиндексируйте файл Hello.txt. Какие изменения произошли в репозитории?

Это делается командой git add Hello.txt, которая добавляет файл в индекс (staging area) —

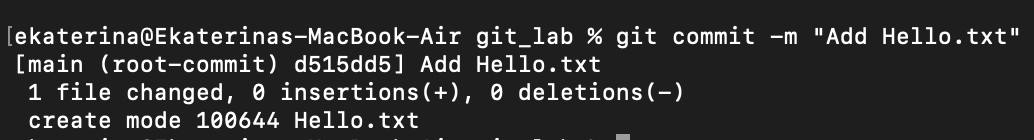
Git начнёт отслеживать его изменения.

Проверяю состояние: git status

  
Теперь файл находится в разделе “Changes to be committed”, то есть он подготовлен для коммита, но ещё не сохранён в истории репозитория.

**10)** Зафиксируйте изменения индексированных файлов в репозитории. Какие изменения произошли в репозитории? Как Git информирует об успешном создании нового коммита? В какую ветку добавлен файл?

Выполняю команду: git commit -m "Add Hello.txt"



**Ответы на вопросы задания**

**1️. Какие изменения произошли в репозитории?**

Создан **новый коммит**, который сохраняет текущее состояние файла **Hello.txt**.  
Этот файл теперь стал частью истории Git.  
Репозиторий больше не “пустой” — в нём появился первый коммит.

**2. Как Git информирует об успешном создании нового коммита?**

После команды git commit Git выводит сообщение:

[main (root-commit) d515dd5] Add Hello.txt

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 Hello.txt

Git сообщает:

* **ветку**, куда добавлен коммит (main)
* **тип коммита** (root-commit — первый коммит)
* **хэш коммита** (уникальный код)
* **сколько файлов изменено**
* **что создан файл Hello.txt**

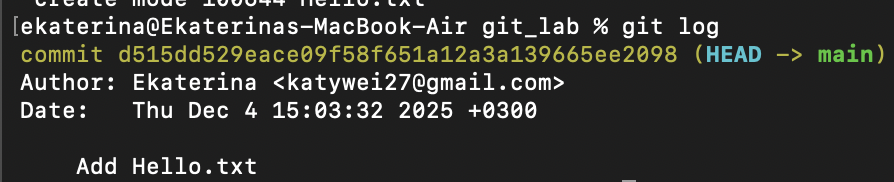
**3. В какую ветку добавлен файл?**

Git пишет: [main (root-commit) ...]

Это значит, что коммит создан в **ветке main**  
И файл Hello.txt добавлен именно туда

**11)**  Просмотрите журнал с помощью команды git log

Я выполнила: git log.

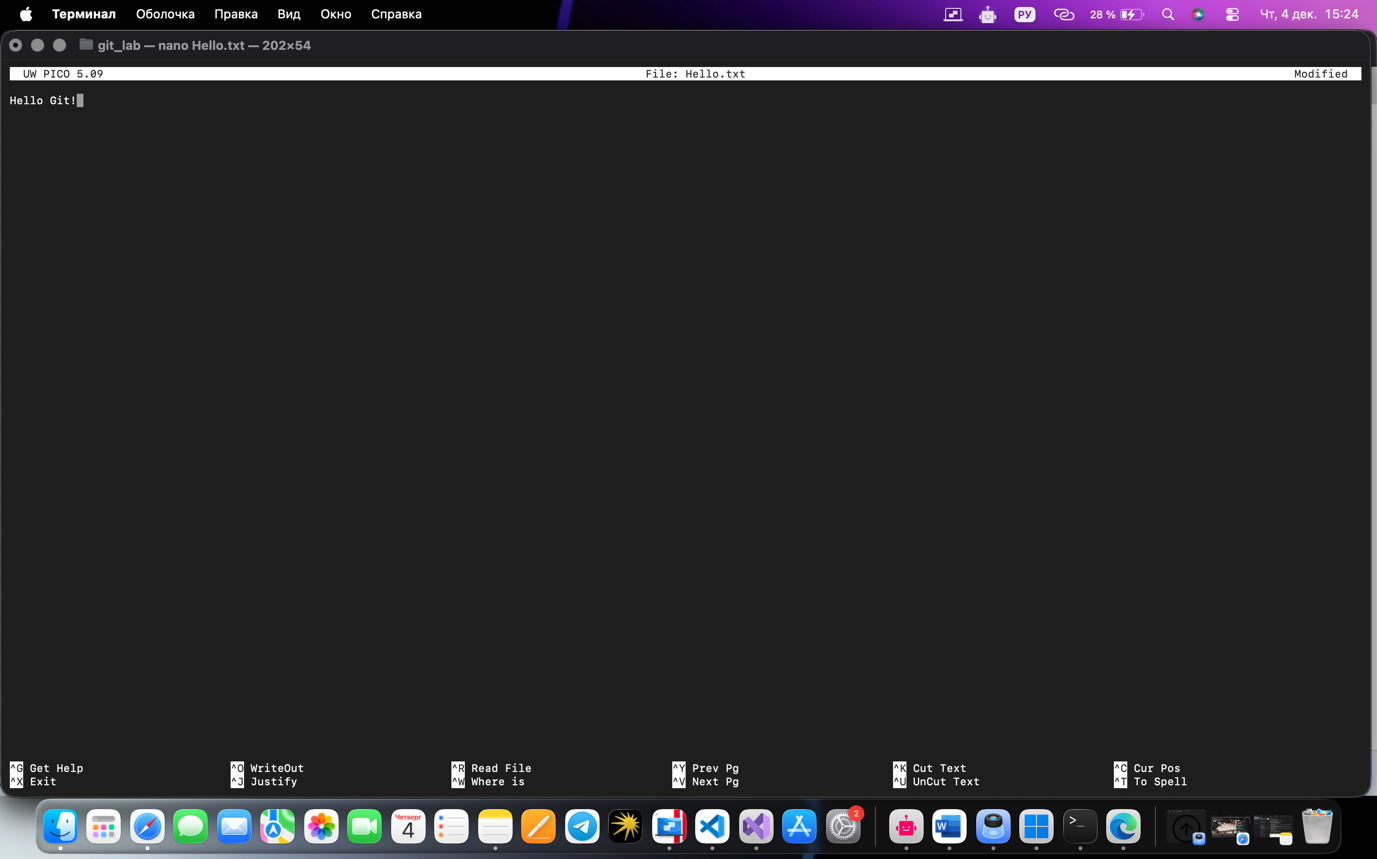


* **commit d515dd5...** — это уникальный хэш коммита (его “ID”).
* **HEAD -> main** — текущая ветка **main**, и HEAD указывает на этот коммит.
* **Author** — имя и email, которые я настроила ранее.
* **Date** — когда создан коммит.
* **Add Hello.txt** — сообщение коммита.

**12)** Научитесь получать информацию о проиндексированных и неиндексированных файлах. Для этого измените существующий файл Hello.txt, добавьте в репозиторий новый файл и исследуйте состояние репозитория. Зафиксируйте изменения в репозитории.

Нужно:

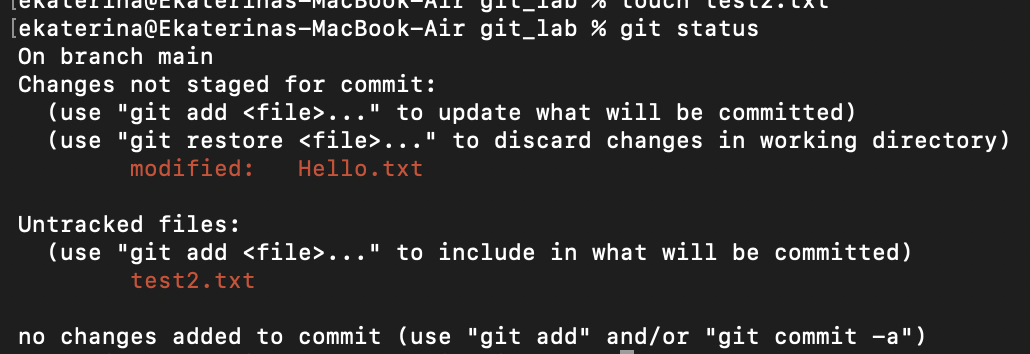
1. **Изменить существующий файл Hello.txt**
2. **Создать новый файл**
3. **Посмотреть статус (git status)**
4. **Зафиксировать изменения**

1) Выполняю: nano Hello.txt и пишу текст Hello Git!

2) Создаю ещё один файл, например test2.txt



3) Просматриваю статус



Сообщение: Changes not staged for commit: modified: Hello.txt — значит, что я **изменила файл Hello.txt**, но **Git ещё не добавил эти изменения в индекс** (staging area).

Сообщение: Untracked files: test2.txt — файл **test2.txt** существует в папке, но Git ещё **не отслеживает** его.

Дальше добавлю файлы с помощью команд: git add Hello.txt; git add test2.txt.

Затем создаю коммит:  
— Файлы добавлены в ветку **main**

**Git подтвердил успешный коммит**

Вот эта строка: [main b658946] Update Hello.txt and add test2.txt

— говорит, что коммит сделан в ветке **main**  
— b658946 — это хеш коммита  
— сообщение коммита: *Update Hello.txt and add test2.txt*

**«2 files changed»**Git сообщает, что в коммите участвуют **2 файла**:

* Hello.txt → изменён
* test2.txt → добавлен

**«1 insertion(+)»**Это означает, что в изменения добавлена **1 строка**.

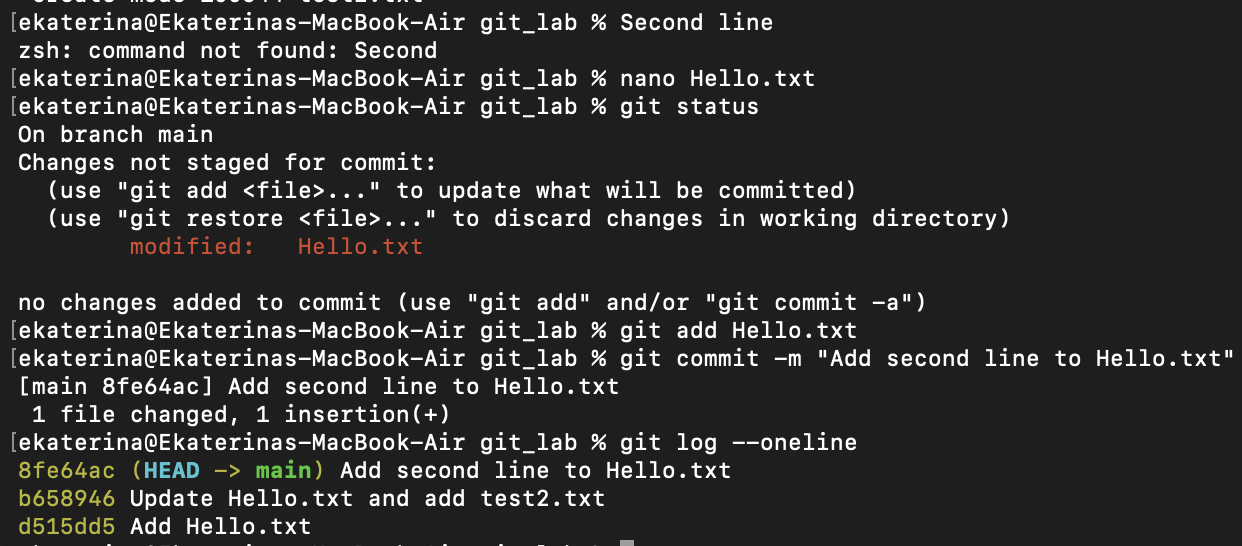
**«create mode 100644 test2.txt»**Это значит, что новый файл **test2.txt** создан и добавлен в репозиторий.

**13)** Научитесь пользоваться командами управления и фиксации изменений (add, commit, status, log).

Внесу небольшое изменение в файл Hello.txt. Например, добавлю строку с помощью команды: Second line

Затем добавлю дополнительный текст: nano Hello.txt

Потом: git status; git add Hello.txt; git commit -m "Add second line to Hello.txt"; git log –oneline;



**Я показала владение командами:**

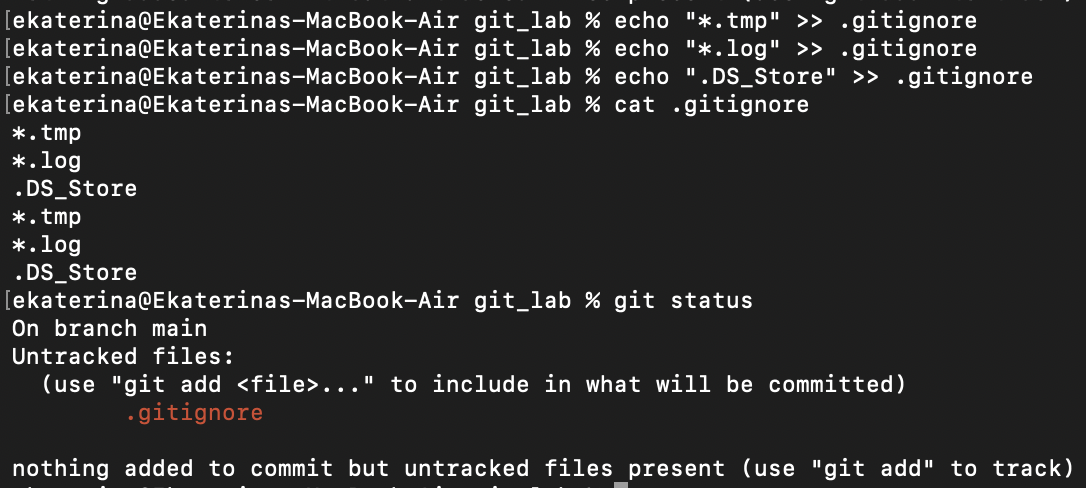
* git status — проверка состояния; git add — индексирование; git commit — фиксация, создание коммита; git log — просмотр истории.

**14)**  Ознакомиться с шаблонами .gitignore и настройте этот файл для вашего репозитория.

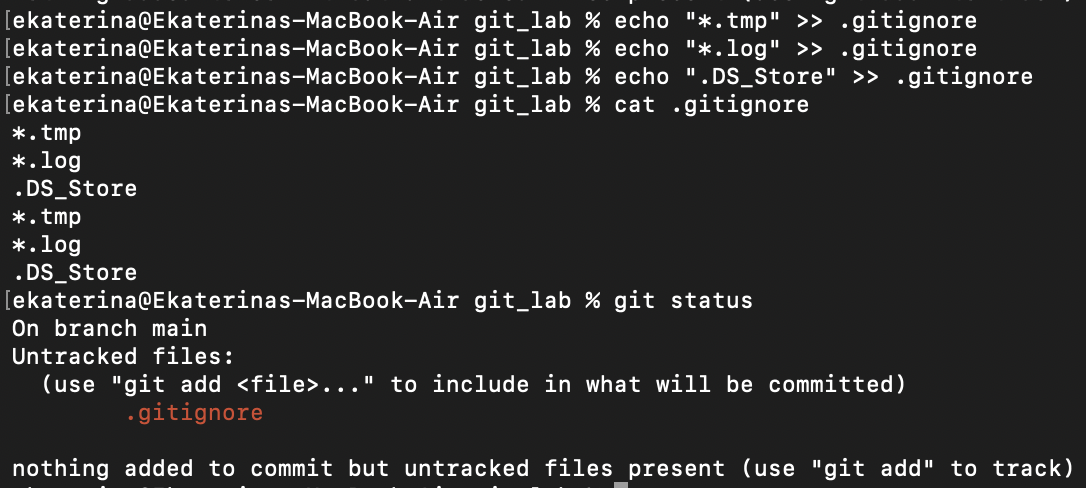
Я создаю файл **.gitignore**, который говорит гиту, какие файлы не отслеживать.

Так как текстовый редактор nano вызывал сложности, файл был создан через терминал с помощью команд:

echo "\*.tmp" >> .gitignore  
echo "\*.log" >> .gitignore  
echo ".DS\_Store" >> .gitignore

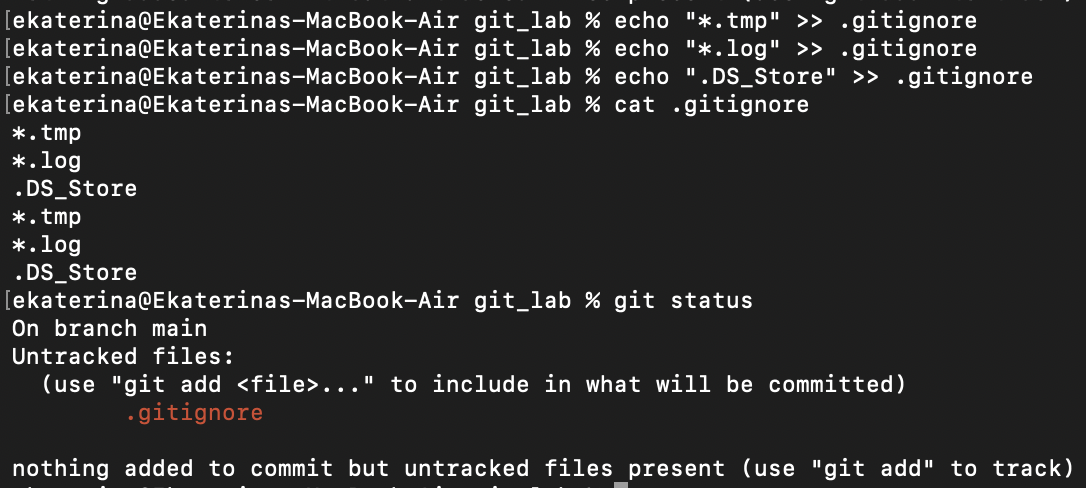


Эти команды автоматически добавили в файл список шаблонов, которые Git должен игнорировать.

Проверка содержимого файла: 

Команда вывела добавленные строки, что подтверждает правильность создания файла.

Статус репозитория после добавления файла:

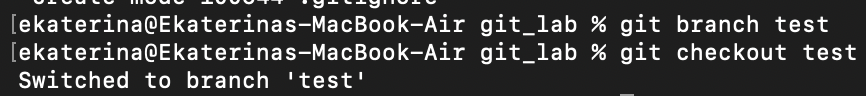


Git показал .gitignore как *untracked file*, что означает, что файл создан, но ещё не добавлен в индекс.

Остался последний шаг — зафиксировать .gitignore  
Для этого ввела: git add .gitignore  
Затем: git commit -m "Add .gitignore file"

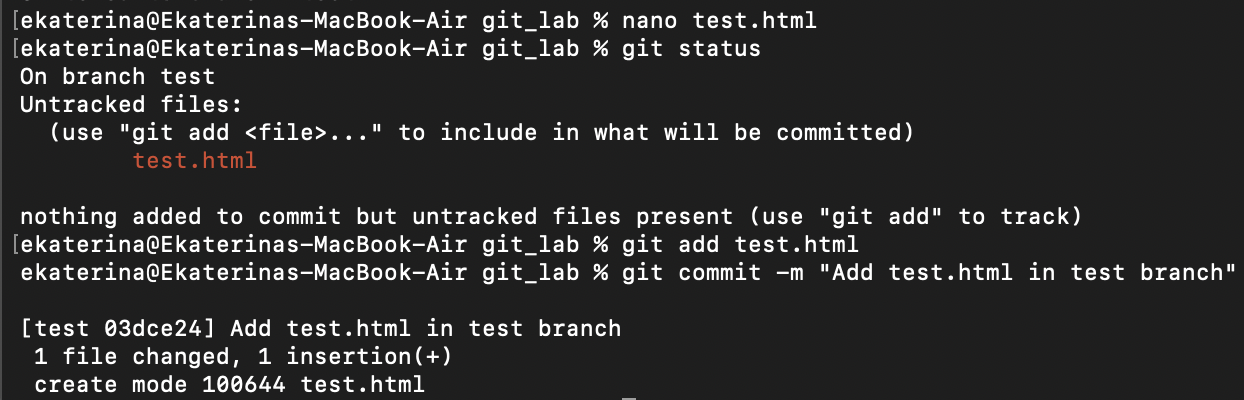
**15)** Создайте (branch) новую ветку, например, test и переключитесь (checkout) в нее.

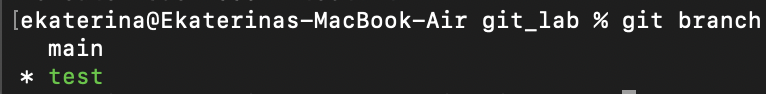
**1.** Создать ветку test:git branch test **2.** Переключиться в неё:git checkout test

Это означает, что я теперь работаю в новой ветке — **test**.

**16)** Создайте новый текстовый файл test.html и зафиксируйте его (commit).

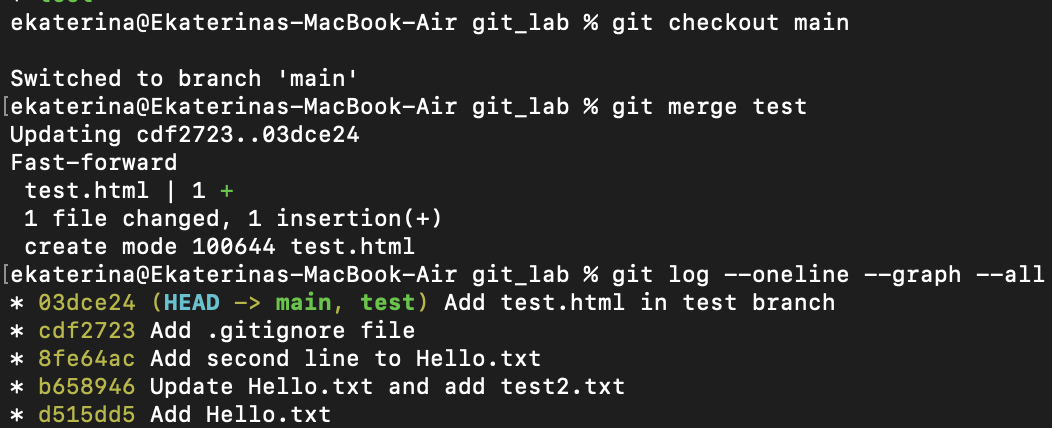
Открою файл в nano и напишу текст: <h1>This is test.html</h1>, затем проверю статус, добавлю файл в индекс и создам коммит:

**17)** Просмотрите список всех веток и определите текущую ветку.

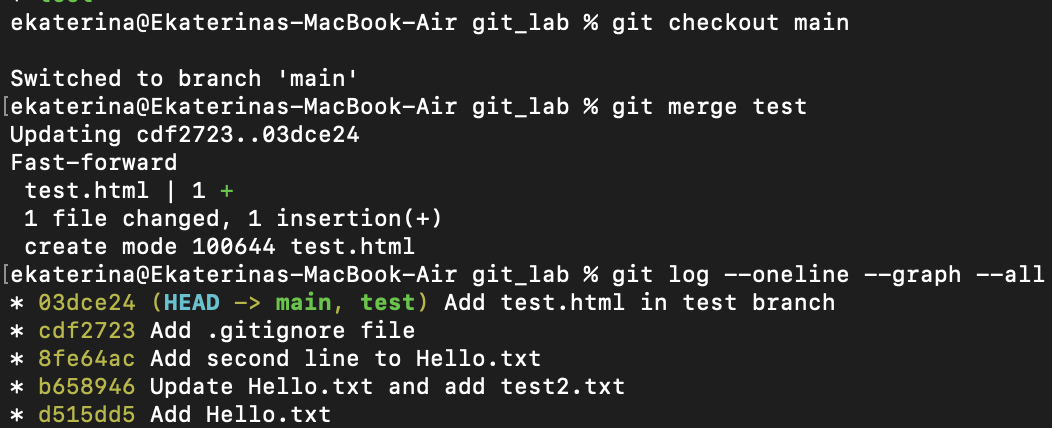
Для этого нужно ввести: git branch

две ветки: main и test, текущая отмечена звёздочкой \*

**18)** Выполните слияние (merge) веток.

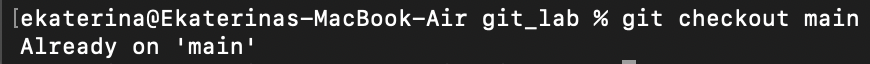
Переключаемся на ветку, куда будем сливать.Обычно сливают **в main**, значит:git checkout main. Затем выполняем слияние: git merge test.

Если нет конфликтов — коммит не нужен, Git делает его сам.  
Проверяем историю: git log --oneline --graph –all

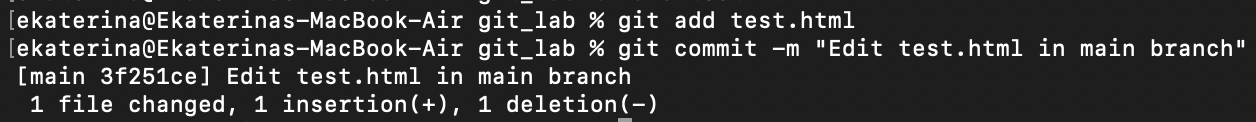


**19)** По очереди отредактируйте файл test.html в ветках master и test. Выполните commit. Выполните слияние в одной и второй ветках. Разрешите ситуацию конфликта слияний.

Изменяем test.html в ветке main, но для начала нужно убедиться, что я нахожусь в main: git checkout main.

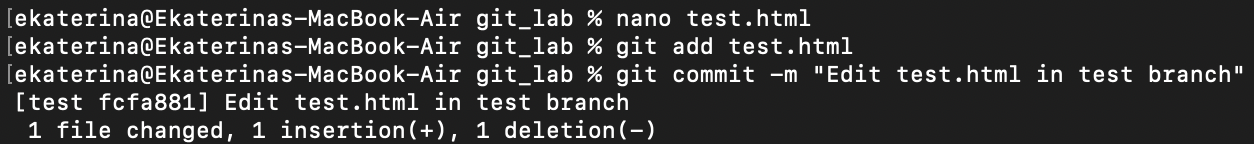


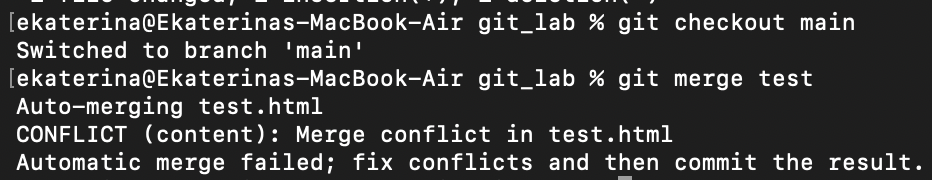
Открываю файл с помощью: nano test.html. Изменяю содержимое, например: <p>Version from MAIN branch</p>. Создаю коммит: git add test.html; git commit -m "Edit test.html in main branch"

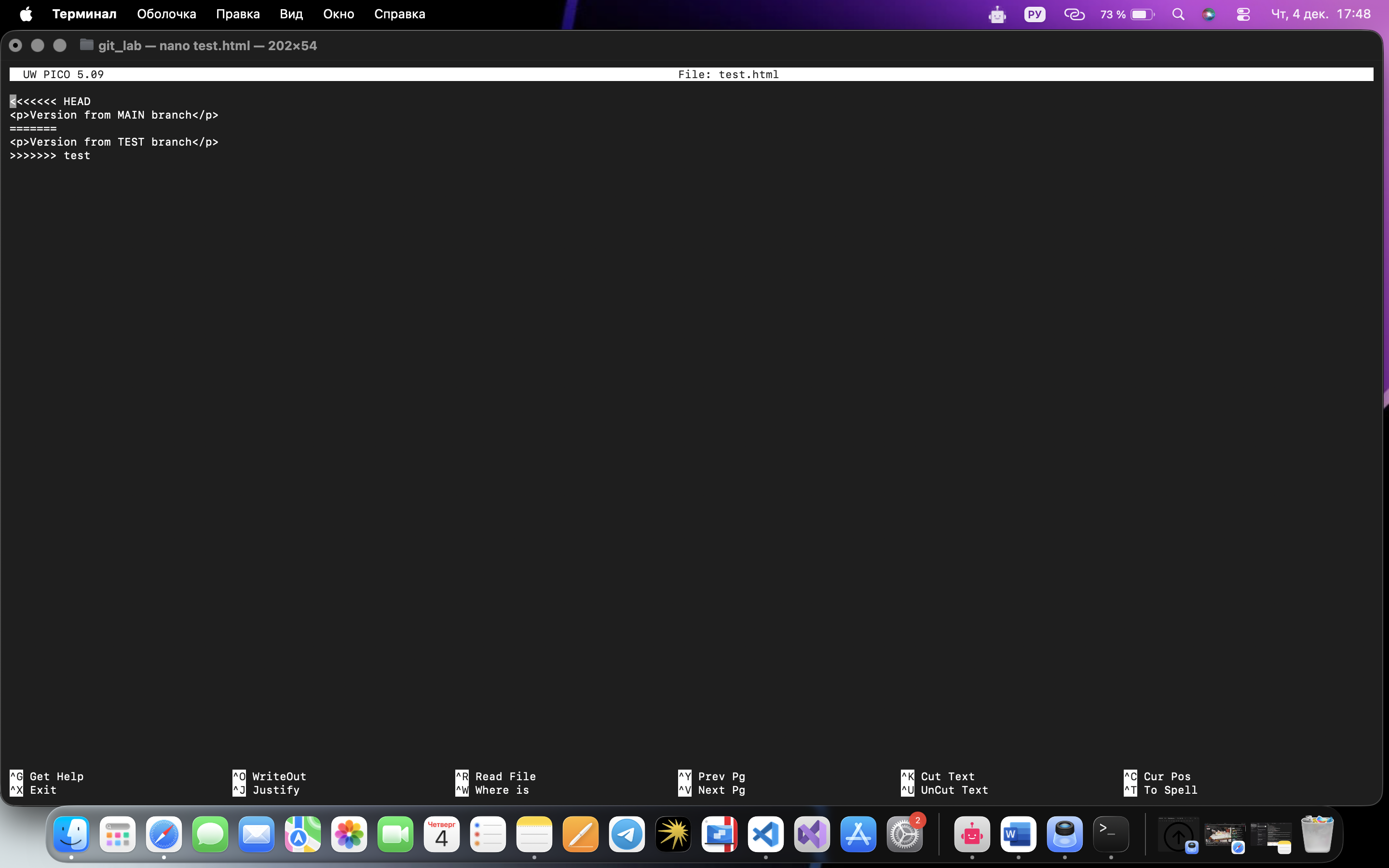


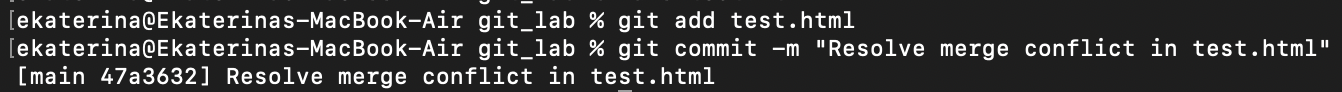
Дальше меняем test.html в ветке test:

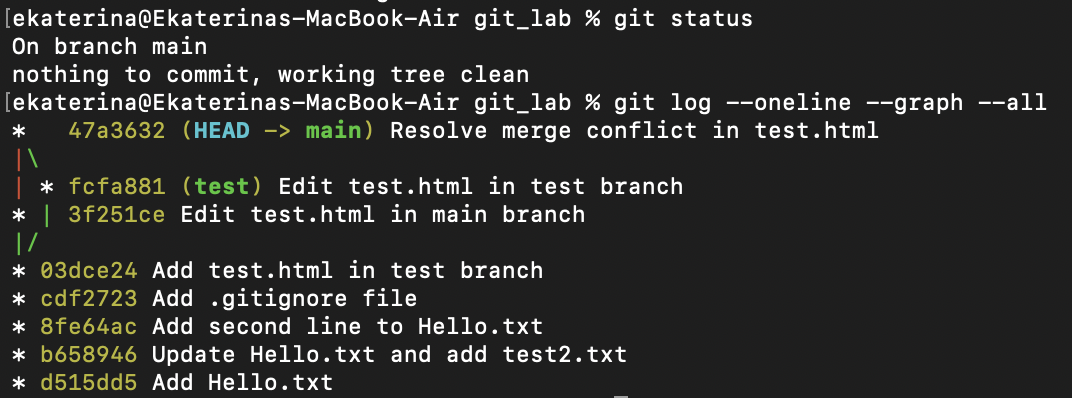
Переход в ветку: git checkout test  
Открываю файл снова: nano test.html  
Меняю **ту же строку**, но по-другому, например: <p>Version from TEST branch</p>. Создаю коммит: git add test.html; git commit -m "Edit test.html in test branch"



Создаю конфликт слияния: теперь пробую слить test → main  
Разрешаю конфликт:

Открою файл test.html: nano test.html, изменяю на <p>Combined version: MAIN + TEST</p>

Добавляю исправленный файл: git add test.html; завершаю merge

Проверю историю (что конфликт реально решён)  
Ввожу:git log --oneline --graph --all

Появился commit — все правильно.

**20)** Основные команды Git для работы с ветками **1. Создать новую ветку:** git branch имя\_веткиПример:git branch test

**2. Переключиться на ветку:** git checkout имя\_веткиПример:git checkout test

**3. Создать и сразу перейти в новую ветку:** git checkout -b имя\_ветки

**4. Посмотреть список веток:** git branchВывод покажет все ветки, текущая помечена \*.

**5. Удалить ветку (когда она больше не нужна):** git branch -d имя\_ветки

**6. Слияние веток (merge):** перейти в ветку, куда нужно влить изменения:git checkout main**;** git merge test