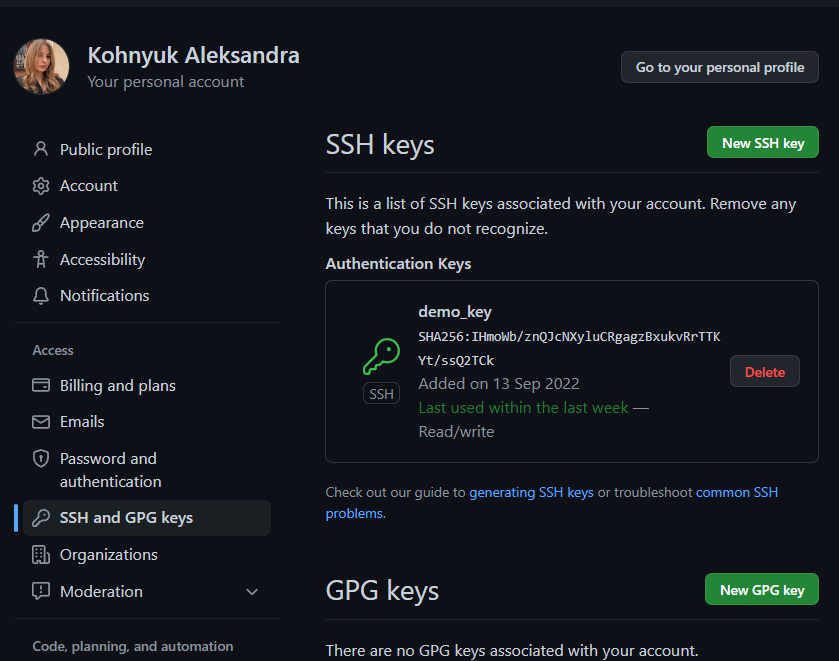
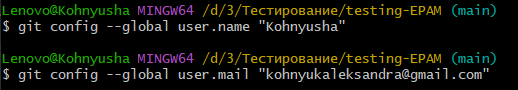
**Лабораторная работа № 1**

**I Can Win**

1. Установили, уже хорошо.
2. ssh-keygen -t rsa -C "kohnyukaleksandra@gmail.com"

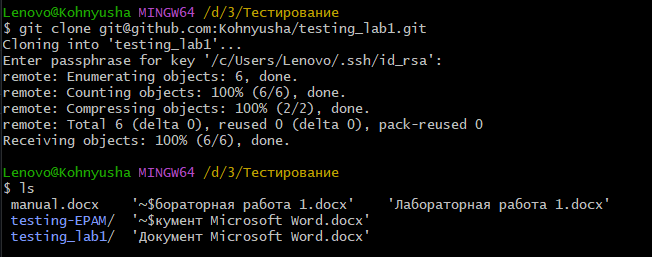


1. Указание имени и мыла юзера

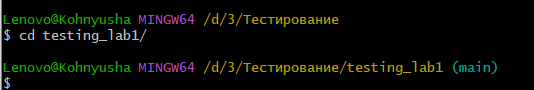


1. Переходим в нужную папочку, куда будем добавлять свой репозиторий с гита

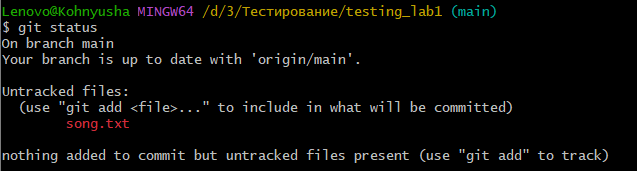




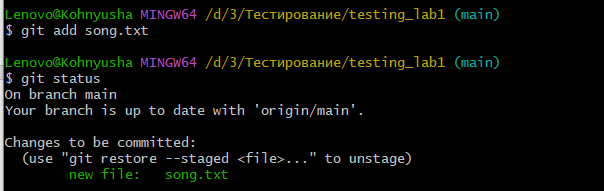
Все, супер

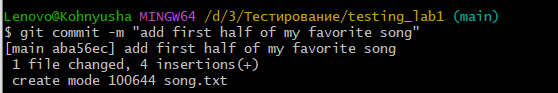


1. Создаем файлик НЕ через консоль, а в папке
2. Git говорит, что появился новый файл

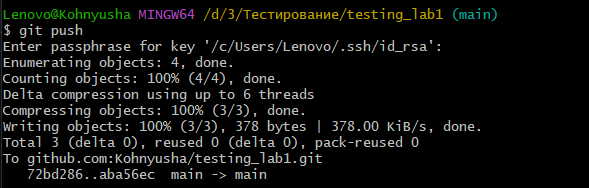


индексируем

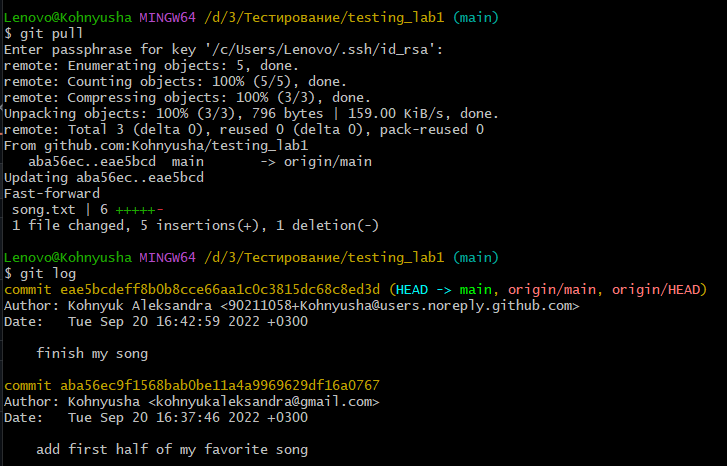




Отправляем на сервер

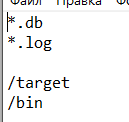


8. Подтягиваем изменения с гита на локальный репозиторий

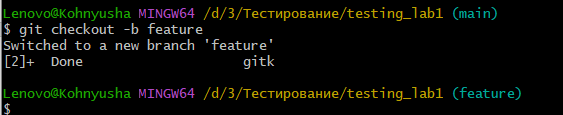


**Bring It On**

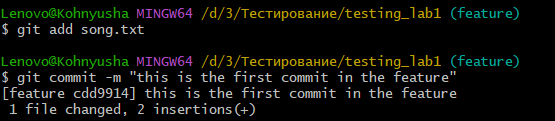
1. Добавить файл .gitignor. Он позволяет нам отфильтровать какие-то вещи, о которых будет говорить гит.

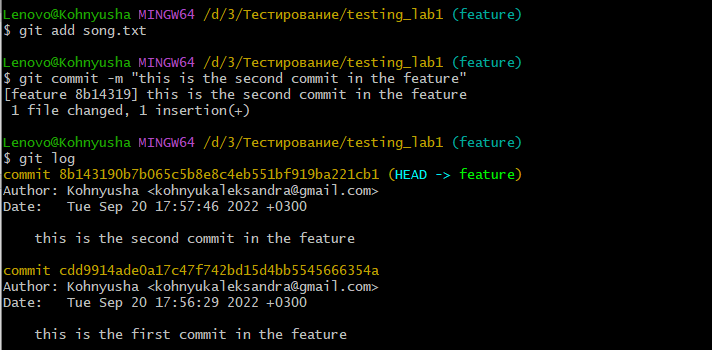


1. Создаем ветку и сразу в нее переходим



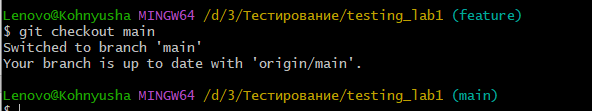
Делаем 2 коммита

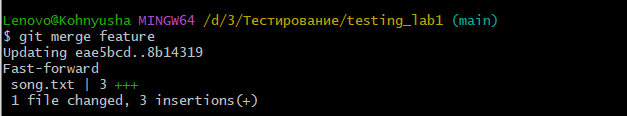


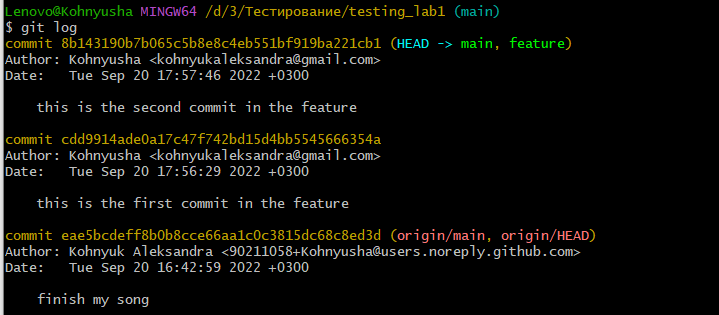


1. Мержим ветки

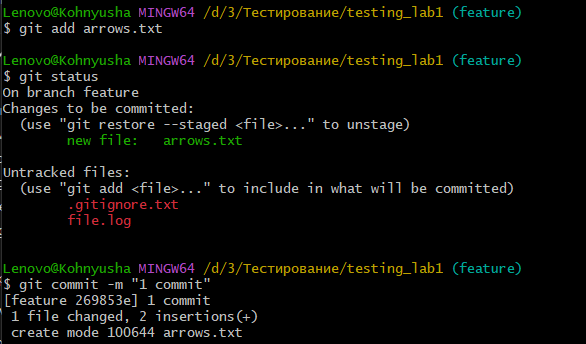
Перейдем в main



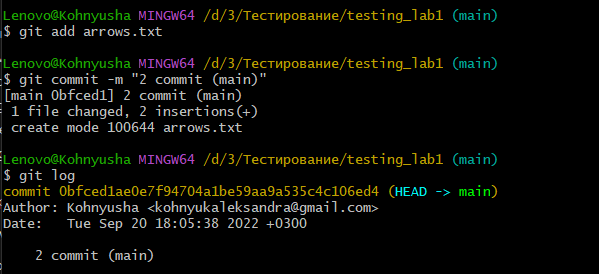




1. Возвращаемся в feature, создаем файл

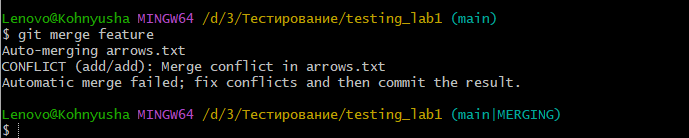


1. Переходим в main и создаем файл с таким же именем, но др содержимым

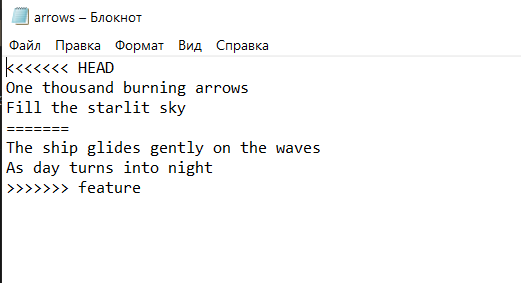


1. Смержим две ветки и соединим содержимое файликов

Появляется конфликт

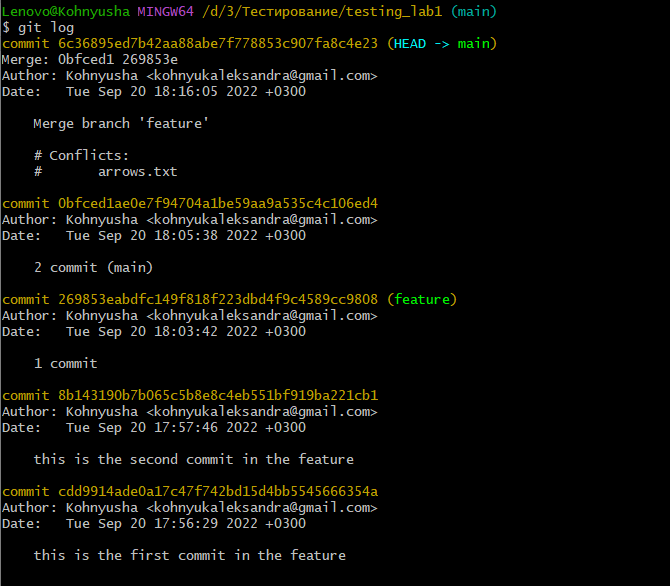


Head – головной коммит, то есть коммит мастера



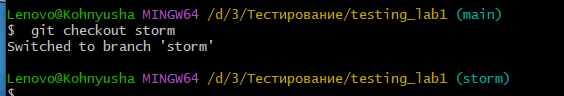
Удаляем флажки

Все супер



**Hurt Me Plenty (и правда)**

1. Создайте ветку storm и добавьте коммит в файл storm.txt:

Twenty ships with Norsemen braves  
Riding the northern wind

2. Добавьте еще 2 строки в storm.txt и сделайте еще один коммит:

They left their shores at early dawn  
As a red sun was rising in the east

3. Вернитесь в master и создайте файл pursuit.txt с текстом ниже:

The warming sun returns again  
And melts away the snow  
The sea is freed from icy chains  
Winter is letting go

Выполните коммит.

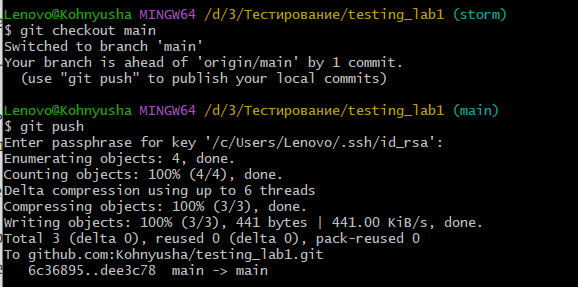
4. Отметьте коммит тегом session1 и перейдите в ветку storm

5. Сделайте rebase ветки storm так чтобы она содержала последний коммит из мастера.

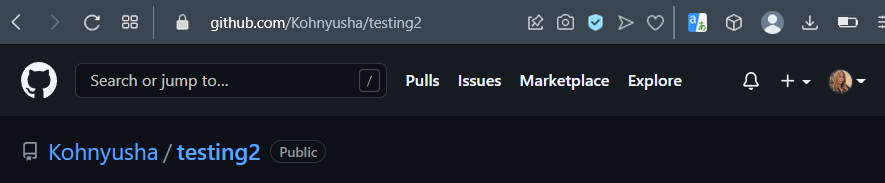


**Hardcore**

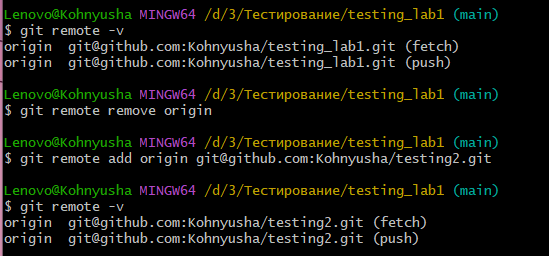
1. Запушить репозиторий

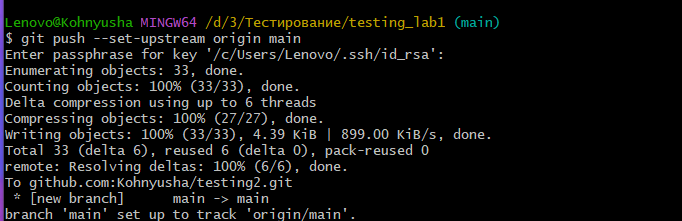


1. Создаем новый репозиторий

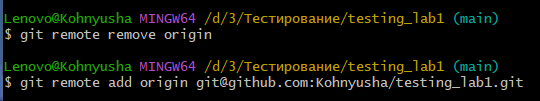


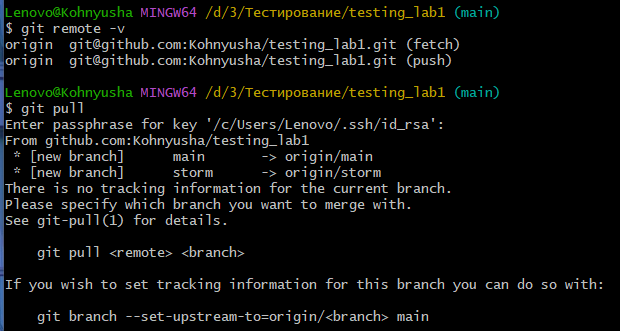
1. Смените remote в локальном репозитории так, чтобы fetch и push шел на новый репозиторий который был создан в предыдущем шаге.



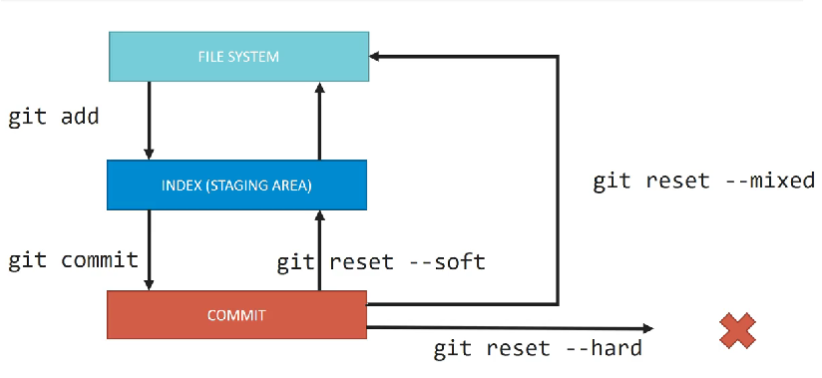


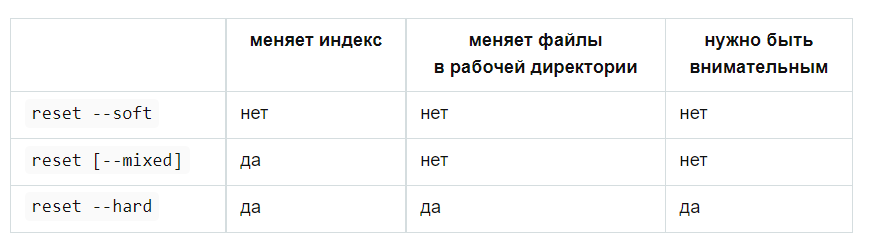
обратно





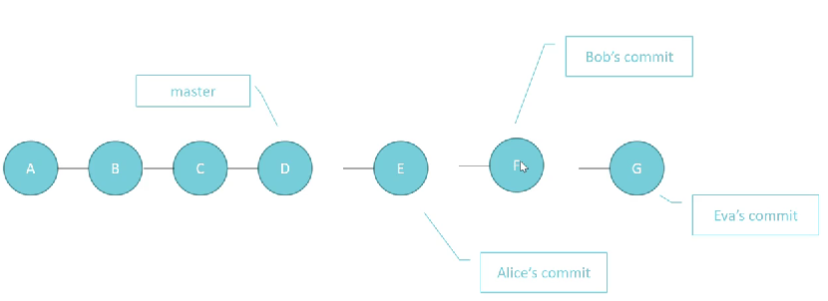
**Версии reset (mixed, soft, hard)**



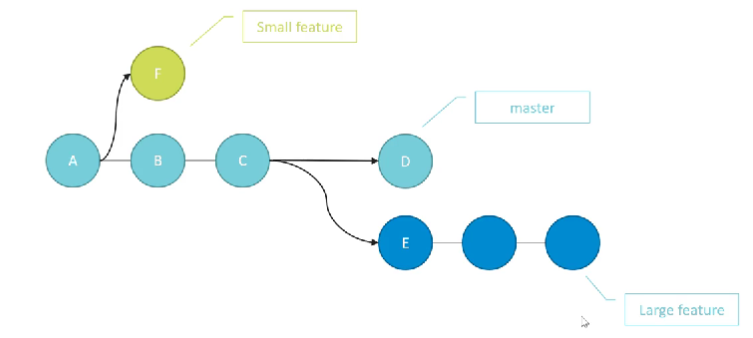


**Стратегии branch**

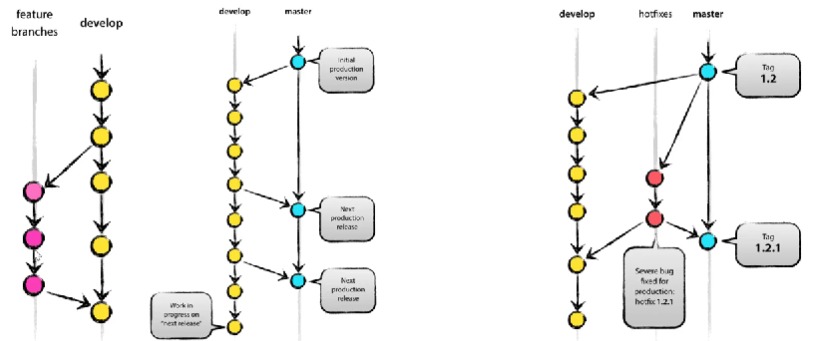
* **Centralized strategy** – все работают в 1 бранче.



* **Feature-branch workflow** – создаем ветку, в ней делаем фитчу и мерджим ее в мастер.



* **Gitflow** – 2010г использование Feature-branch workflow, но с какими-то опред бранчами, обозн в задаче.
* В Feature-branc делается какая-то фитча, или закр дефект и они напрямую попадают в девелоп.
* местер-бранче мы храним только тот .код, который отдаем потребителю (гарантия правильного кода в мастере)
* хот-фиксес, гит флоу это си-ма с исп фитчей-бранчей, но есть возможность сделать ветку и для себя

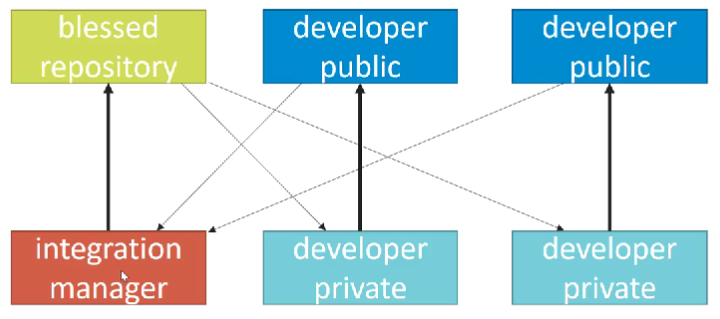


**Integration manager workflow**: можем пушить код в 1 реп, а пуллить из другого.

Есть некоторый **“blesse repository”** – *священный репозиторий,* в который имеет право пушить только **“integration manager” –** *человек, наделенный определенными полномочиями* (крч он просто пуллит код у девелоперов, и если считает его хорошим, то пушит его в репозиторий)

**“developer private”** пуллят свой код из *священного репозитория*, а пушают в свой собственный **“developer public”**.

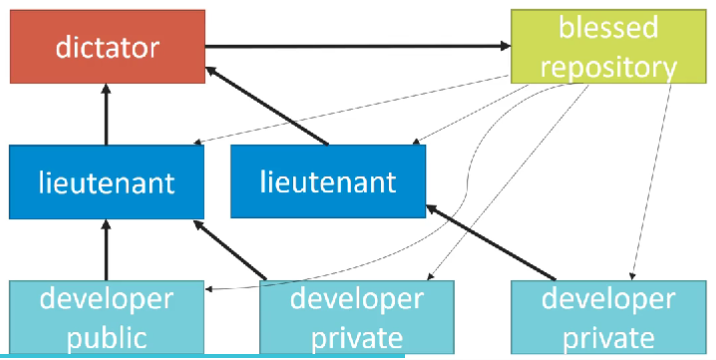
**“integration manager”** пуллит код у конкретных девелоперов, и смотрит, если код хороший, то пушит его в **“blesse repository”.**



**Dictator and Lieutenants workflow**: испол ядро линукса. Это что-то похожее на **“integration manager”**, только их больше.

Есть некоторый **“blesse repository”**, в него имеет право пушить только 1 человек **– “dictator”.** Он решает что пойдет в репозиторий.

Когда **developer** отправляет свои изменения в ветки, их подтягивают **lieutenant.** Если они считают, что изменения хорошие, то предоставляют их **dictator.**



**Forking workflow**: решает проблему доступа «чтение-запись» для конкретных девелоперов.

**“original developer”** – типо главный. В одиночку он ничего не сделает и ему нужны помощники, но доверять право на запись работы в **“original repository”** он не хочет.

**Fork** – *это копия ориг репа*. Например **“fork N developer”** - *разраб* хочет добавить вещь в проект, и он делает форк. Добавляет коммиты и создается фитча и публикуется в их форке.

Дальше **“original developer”** пуллит форк разработчиков, смотрит, и если посчитает нужным, пушит в оригинальный реп.

