У нас есть директория где хранятся файлы

Создание репозитория – просто делает гит инит в директории – рабочей зоне, тогда изменения в директории и файлах в ней будет отслеживать гит

*# Создаем новый проект*

mkdir hexlet-git

*# Переходим в созданную директорию*

cd hexlet-git

*# Выполняем инициализацию*

git init

git status – показывает изменения в рабочей зоне, файлы которые были добавлены в индекс – (git add README.md) – это нужно для того чтобы накомить все логически связанные изменения для одного комита

комит – окончательное добавление в репозиторий – то есть все изменения были сохранены и их можно будет вернуться к ним, или даже посмотреть как файл выглядел в то время

git commit -m 'add README.md'

Первое соединение с репозиторием на гитхабе:

1. git remote add origin git@github.com:<ИМЯ НА ГИТХАБЕ>/hexlet-git.git
2. git branch -M main
3. git push -u origin main

далее чтобы запушить – просто git push <файл> - сохранить изменения на удаленном репозитории на гитхабе, чтобы другие разработчики могли скачать эти обновления – обновить свой репозиторий

Полностью клонировать репозиторий к себе

git clone git@github.com:<ИМЯ НА ГИТХАБЕ>/hexlet-git.git

 перед началом работы нужно всегда выполнять команду git pull --rebase, которая скачивает из внешнего репозитория новые коммиты и добавляет их в локальный репозиторий.

Если удалить один из файлов – можно либо закомитеть изменения, либо восстановить файл из репозитория – **git restore** <файл>

rm PEOPLE.md

git status

On branch main

Your branch is up to date with 'origin/main'.

Changes not staged **for** commit:

**(**use "git add/rm <file>..." to update what will be committed**)**

**(**use "git restore <file>..." to discard changes **in** working directory**)**

deleted: PEOPLE.md

no changes added to commit **(**use "git add" and/or "git commit -a"**)**

git rm – команда которая удаляет и делает подготовку к комиту

Анализ сделанных изменений

Git diff - выводит какие изменения были сделаны в файлах – добавлена строка, удалена строка

git diff

diff --git a/INFO.md b/INFO.md

index d5225f8..40f51f1 100644

--- a/INFO.md

+++ b/INFO.md

@@ -1 +1,2 @@

git is awesome!

+new line

diff --git a/README.md b/README.md

index ffe7ece..00fd294 100644

--- a/README.md

+++ b/README.md

@@ -1 +1 @@

-Hello, Hexlet!

+Hello, Hexlet! How are you?

Но предыдущая не работает если мы ввели файл в индекс, тогда след команда показывает такие изменения даже если файл в индексе

*# Выведет все изменения сделанные в рабочей директории*

*# которые были добавлены в индекс*

git diff --staged

Анализ истории изменений (коммитов) – чтобы понять кто и когда и какие изменения вносил

След команда выводит все изменения проекта самые последние наверху – номер комита – уникальный хеш, автор и дата изменения

git log

*# Ниже неполный вывод истории проекта*

commit 5120bea3e5528c29f8d1da43731cbe895892eb6d

Author: tirion <tirion@got.com>

Date: Thu Sep 17 18:04:19 2020 -0400

add new content

commit e6f625cf8433c8b1f1aaed58cd2b437ec8a60f27

Author: tirion <tirion@got.com>

Date: Thu Sep 17 16:14:09 2020 -0400

add INFO.md

Позволяет сразу выводить диф для каждого изменения

git log -p

Чтобы посмотреть диф отдельного комита – то есть за один комит делается сохранение изменений в репозиторий всех измененых файлов – изменения логически связанные. Можно не писать полностью весь хеш комита, только 8 чисел

git show 5120bea3

Следующая команда выводит того кто изменял каждую строчку файла

git blame INFO.md

e6f625cf **(**tirion 2020-09-17 16:14:09 -0400 1**)** git is awesome!

5120bea3 **(**tirion 2020-09-17 18:04:19 -0400 2**)** new line

Git grep – ищет совпадения в строках файлов

git grep line

INFO.md:new line

*# Флаг i позволяет искать без учета регистра*

git grep -i hexlet

README.md:Hello, Hexlet! How are you?

*# Поиск в конкретном коммите*

git grep Hexlet 5120bea3

*# Поиск по всей истории*

*# rev-list возвращает список хешей коммитов*

git grep Hexlet $(git rev-list --all)

Отмена изменений в рабочей директории

Удаляет все изменения которые быле не закомичены, но эти изменения потом не вернешь

git clean -fd

Отменить изменения в файле – помогает гит прописано в гит статус

git restore <файл>

Если нужно отменить индексацию – то есть подготовку к комиту, но не удалять сами изменения в файле. Если нужно все-таки удалить все изменения, то + к это й команде git restore <файл>

git restore --staged INFO.md

Отмена комита

Отмена изменений комита

Git revert + уникальный хеш номер комита изменения которого нужно изменить (см его git log)

git revert aa600a43cb164408e4ad87d216bc679d097f1a6c

Команда revert может "отменять" не только последний коммит, но и любой другой коммит из истории проекта. Создает новый коммит, в котором выполнены изменения обратные указанному в команде коммиту

Удалить последний комит с изменениями

git reset --hard HEAD~

Флаг --hard означает полное удаление. Без него git reset отменит коммит, но не удалит его, а поместит все изменения этого коммита в рабочую директорию, так что с ними можно будет продолжить работать. HEAD~ означает "один коммит от последнего коммита". Если бы мы хотели удалить два последних коммита, то могли бы написать HEAD~2.

git reset HEAD~

Изменение последнего комита

Когда забыли добавить некоторые файлы в индекс перед комитом, но сделали уже комит

Додобавляем в индекс и делаем команду

git commit --amend

Для того, чтобы не открывался редактор для ввода описания коммита к команде git commit --amend можно добавить опцию --no-edit. В этом случае описание коммита не изменится.-

Индекс

Одной командой добавить файл в индекс и закомитить

echo 'new data' **>>** INFO.md

*# Не нужно явно вызывать git add*

git commit INFO.md -m 'update INFO.md'

Добавить все измененные файлы в индекс – опасно можем потом закомитить лишнее

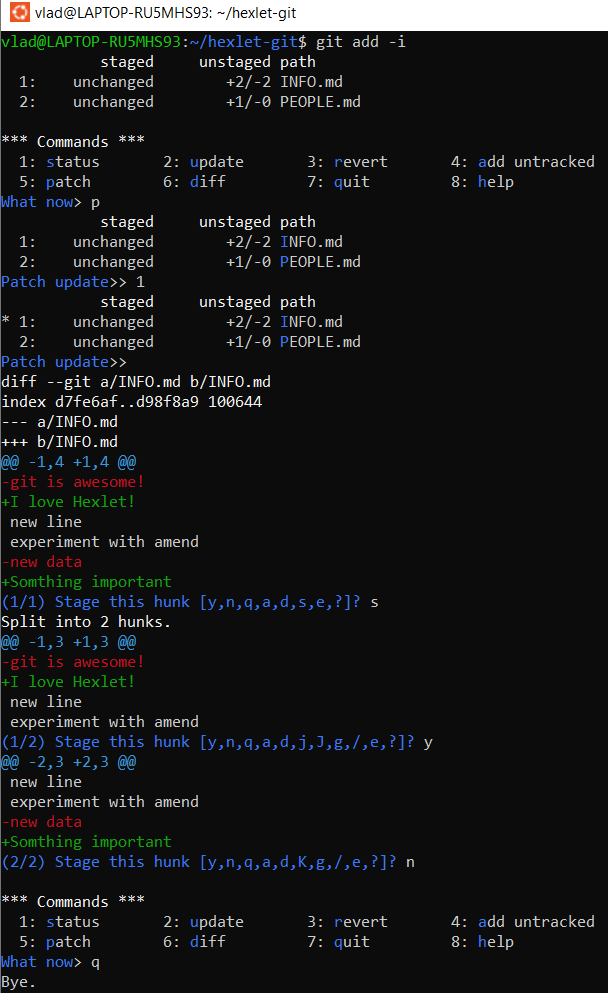
git add .

Комит с одновременным добавлением все в индекс

git commit -am 'do something'

Позволяет отправить в индекс только часть изменений файла, чтобы потом закомитеть эту часть одним сообщением, а потом докомитеть другую часть с другим

git add -i



Перемещение по истории

Показать сокращенный вывод истории коммитов

git log --oneline

Переключиться на определенный комит – теперь мы отматаем историю назад и можем посмотреть какие файлы были на тот момент и скопировать нужные и вернувшись закоммитить

git checkout e6f625c

Вернуться в настоящее – время после последнего комита

git checkout main

показывает в каком месте мы находимся – после какого последнего комита – нужно постоянно проверять где находимся

git branch

если просто main – то в настоящем, если цифры хеша то в прошлом комите

чтобы более себя обезопасить делают проверку прямо в командной строке

*# Если на последнем коммите*

hexlet-git git:**(**main**)**

*# Если на коммите из прошлого*

hexlet-git git:**(**e6f625c**)**

Как добиться такого вывода? Ответ на этот вопрос зависит от используемого командного интерпретатора. В Bash вывод местоположения происходит благодаря редактированию переменной окружения $PS1 – подробнее в материалах

Быстрее всего переключаться между ветками

git switch (например git switch master)

Git checkout – это старая команда, которая использовалась для создания и переключения веток.

Она также может использоваться для восстановления изменений с определенного коммита. Но git checkout делает больше. Она позволяет копировать файлы из любой ветки или коммита прямо в ваше рабочее дерево, не переключая ветки. Если вы не понимаете этого, ничего страшного. Просто помните, что git checkout делает больше, чем простое переключение веток, и дополнительные функции начали создавать путаницу.

Понимание Git

Вывести историю коммитов в виде ветки

git log --graph

Игнорирование файлов (gitignore) – игнорирует изменения в различных логов, в файлах зависимости и прочее

*# В этом файле можно оставлять комментарии*

*# Имя файла .gitignore*

*# Файл нужно создать самостоятельно*

*# Каждая строчка — это шаблон, по которому происходит игнорирование*

*# Игнорируется файл в любой директории проекта*

access.log

*# Игнорируется директория в любой директории проекта*

node\_modules

*# Игнорируется каталог в корне рабочей директории*

/coverage

*# Игнорируются все файлы с расширением sqlite3 в директории db,*

*# но не игнорируются такие же файлы внутри любого вложенного каталога в db*

*# например, /db/something/lala.sqlite3*

/db/**\***.sqlite3

*# игнорировать все .txt файлы в каталоге doc/*

*# на всех уровнях вложенности*

doc/**\*\***/**\***.txt

Но нужно самим добавить этот файл touch .gitignore

*# добавляем в файл правила игнорирования по примеру выше*

git add .gitignore

git commit ...

* У git rm есть опция --cached. Если её использовать, файл будет удалён из репозитория, но не из рабочей директории

Если не удаляется значит не существует в репозитории

Бывают случаи когда во время основной задачи появляется ещё одна которую нужно быстро решить

если ваши изменения не пересекаются с изменениями по срочной задаче, вы можете внести исправления, добавить их в индекс, закоммитить и запушить.

Но если они затрагивают файлы основной задачи, тогда помогает след команда stash она прячет изменения

git stash

*# изменяем файлы*

git stash

*# Вернутся последние изменения*

git stash pop

*# Вернутся предпоследние изменения*

git stash pop