1. Создать новый проект

mkdir Hexlet-Git

2. Перейти в созданный репозиторий

cd Hexlet-Git

3. Выполнить инициализацию

git init

4. Посмотреть статус репозитория

git status

5. Создать файл README.md со строкой текста

echo 'Hello, Hexlet!' > README.md
echo 'Haskell Curry' > PEOPLE.md

6. Добавить файл под git

git add git add README.md

7. Сделать коммит

git commit -m 'add README.md'

8. Добавьте файл PEOPLE.md в репозиторий

git add PEOPLE.md

9. Репозиторий для работы находится в директории code-user. Перейти в неё нужно в терминале. Добавьте в репозиторий два файла двумя разными коммитами:

hexlet.txt с текстом Hello, Hexlet!
index.html с текстом <h1>Мама, я коммичу</h1>

cd code-user
echo 'Hello, Hexlet!'>hexlet.txt
git add hexlet.txt
git commit -m 'add hexlet.txt'
echo '<h1>Мама, я коммичу</h1>'>index.html
git add index.html
git commit -m 'add index.html'

10. Создать репозиторий на GitHub без добавления файлов

Repositories -> New -> Hexlet-Git -> отключить чекбокс «Add a README file»

11. Связать внешний репозиторий с локальным

git remote add origin git@github.com:<vypiemzalyubov/Hexlet-Git.git
git branch -M main
git push -u origin main

12. Удалить локальный репозиторий и склонировать внешний

git clone git@github.com:vypiemzalyubov/Hexlet-Git.git

13. Синхронизировать репозитории

git pull --rebase

14. Добавьте новый файл NEW.md с произвольным содержимым в репозиторий (нужно выполнить коммит) и залейте изменения на Github

echo 'произвольное содержимое'>NEW.md
git add NEW.md
git commit -m 'add NEW.md'
git push

15. Выполните клонирование репозитория. Находясь в той директории, которая загружается автоматически в терминале, первый параметр «откуда», второй — «куда»

git clone repos/git-user code-user
cd code-user

16. В репозитории уже есть два файла. Измените их так:

В hexlet.txt добавьте второй строчкой текст I like to change files
В index.html замените текст на <h1>С помощью гит можно писать книги</h1>

Выполните один коммит, содержащий сразу эти два изменения. Во время коммита git попросит установить емейл и имя пользователя. Сделайте это, используя любые вымышленные данные. Добавьте изменения в основной репозиторий с помощью git push

echo 'I like to change files'>>hexlet.txt
echo '<h1>С помощью гит можно писать книги</h1>'>index.html
git add .
git commit -m 'update hexlet.txt and index.html'
git config --global user.email 'myemail@gmail.com'
git config --global user.name 'vypiemzalyubov'
git commit -m 'update hexlet.txt and index.html'
git push

17. Удалить PEOPLE.md в рабочей директории и затем восстановить его

rm PEOPLE.md
git status
git restore PEOPLE.md

18. Удалить PEOPLE.md и закоммитить

rm PEOPLE.md
git add PEOPLE.md
git commit -m 'remove PEOPLE.md'

19. Удалите файл NEW.md и сделайте коммит

rm NEW.md
git add NEW.md
git commit -m 'remove NEW.md'

20. Добавьте файл INFO.md с текстом *git is awesome!* и сделайте коммит и залейте изменения на GitHub

echo 'git is awesome!'>INFO.md
git add .
git commit -m 'add INFO.md'
git push

21. В рабочей директории два изменения: один файл удален, другой изменен. Отмените эти изменения. Вывод git status должен выглядеть следующим образом:

$ git status
nothing to commit, working tree clean

git restore hexlet.txt
git restore index.html
git status

22. Редактировать файлы INFO.md и PEOPLE.md и посмотреть изменения

```
echo 'new line' >> INFO.md
echo 'Hello, Hexlet! How are you?' > README.md
git diff
```

23. Сделайте коммит с сообщением *add new content* и залейте изменения на GitHub

```
git add .
git commit -m ‘add new content’
git push
```

24. Удалите файл index.html. Переименуйте файл hexlet.txt в hexlet2.txt
После каждого действия добавляйте изменения в отслеживание и изучайте вывод команды git diff --staged и git status. Обратите внимание на различия вывода последней команды до добавления изменений в индекс и после. Закоммитьте каждое изменение отдельно

```
rm index.html
git add index.html
git diff --staged
git status
git commit -m 'remove index.html'
mv hexlet.txt hexlet2.txt
git add .
git diff --staged
git status
git commit -m 'rename hexlet.txt to hexlet2.txt'
```

25. Выполнить команды, анализирующие историю изменений

```
git log
git show 5120bea3
git blame INFO.md
git grep line
```

26. Посмотрите изменения, сделанные в последнем коммите. Внесите обратные изменения в файл hexlet2.txt. Сделайте коммит. Таким образом новым коммитом мы перезаписываем изменения, сделанные в последнем коммите

```
git log -p
echo 'Hello Hexlet!' > hexlet2.txt
git add hexlet2.txt
git commit -m 'update hexlet2.txt'
```

27. Создать папку и файл, а затем отменить эти операции

```
mkdir one
touch two
git clean -fd
```

28. Создать файл и отменить изменения

```
echo 'new text' > INFO.md
git restore INFO.md
```

29. Создать файл, добавить под git и отменить изменения в индексе

```
echo 'new text' > INFO.md
git add INFO.md
git restore --staged INFO.md
```

30. Отмените все изменения, сделанные в рабочей директории и индексе. В результате должен получиться такой вывод:

$ git status
nothing to commit, working tree clean

```
git restore hexlet.txt
git restore --staged index.md index.html
git restore index.html
git clean -fd
```

31. Удалить файл PEOPLE.md, закоммитить, затем отменить коммит

```
rm PEOPLE.md
git add .
git commit -m ‘remove PEOPLE.md’
git revert aa600a43cb164408e4ad87d216bc679d097f1a6c
```

32. Добавить новый коммит и удалить

```
echo 'test' >> INFO.md
git add INFO.md
git commit -m 'update INFO.md'
git reset --hard HEAD~
```

33. Полностью удалите два последних коммита

```
git reset --hard HEAD~2
```

34. Добавить изменения в INFO.md и README.md, добавить INFO.md под Git и закоммитить. Добавить под Git README.md и закоммитить в текущий коммит

```
echo 'experiment with amend' >> INFO.md
echo 'experiment with amend' >> README.md
git add INFO.md
git commit -m 'add content to INFO.md and README.md'
git add README.md
git commit --amend
```

35. Вы сделали коммит, но забыли добавить в него пару файлов. Измените последний коммит так, чтобы он содержал все три файла одновременно

```
git add .
git commit --amend
```

36. В файл hexlet.txt было добавлено несколько строк. Используя интерактивный режим git add, разбейте изменения на две части и затем добавьте в индекс одну из них. Выведите на экран состояние изменений подготовленных к коммиту в интерактивном режиме с помощью status. Сделайте коммит, в комментарии которого напишите значение из столбца staged

```
git status
git diff
git add -i
patch
1
# Подтверждаем выбор нажатием Enter
s
y
n
status
quit
git commit -m '+2/-0'
```

37. Вы сделали несколько коммитов в репозиторий, и в каком-то из них сохранили файл todo.md со списком задач, а потом удалили этот файл. Загвоздка в том, что в комментарии к коммиту добавление этого файла отражено не было.
Найдите в истории коммит, в котором был добавлен файл todo.md, и переключитесь на него с помощью git checkout

```
git log -p --oneline -- todo.md
git checkout 7c2fcf8
```

38. Проигнорируйте и удалите из репозитория, если это необходимо, но не удаляйте из рабочей директории, файлы notes.txt и todo.md, а затем сделайте коммит со всеми изменениями

```
echo 'notes.txt' > .gitignore
echo 'todo.md' >> .gitignore
git rm --cached notes.txt
git commit -m 'remove notes.txt from repo'
git add hexlet.txt
git commit -m 'update hexlet.txt'
git add .gitignore
git commit -m 'add .gitignore'
```

39. В репозитории создано 2 дополнительные ветки, помимо main.
Переключитесь на ветку refactoring и удалите два последних коммита с помощью git reset --hard.
В ветке working-on-html сделайте коммит, отредактировав файл index.html следующим образом:

Замените содержимое тега h1 на «Ветки в Git достойны отдельного курса»
Замените содержимое тега p на «Ветки являются ссылками на определённый коммит.»

```
git switch refactoring
git reset --hard HEAD~2
git switch working-on-html
vim index.html
git add index.html
git commit -m 'fix index.html'
```

40. В рабочей директории уже есть изменения, сохраните их с помощью git stash.

- Создайте файл todo.md с любым содержимым
- Добавьте в файл hexlet.txt строку: «stash не трогает новые файлы, которые ещё не добавлены в индекс.»
- Сделайте коммит
- Восстановите сохранённые ранее изменения
- Сделайте еще один коммит, включающий и новые файлы

```
git add .
git stash
echo 'Hexlet is awesome!' > todo.md
echo 'stash не трогает новые файлы, которые ещё не добавлены в индекс.' >> hexlet.txt
git add .
git commit -m 'refactoring'
git stash pop
git commit -am 'changes from stash'
```

Зачем нужен индекс (stage)?

- (•) Для формирования набора файлов, которые попадут в один коммит
- ( ) Всё, что попадает в индекс, автоматически оказывается внутри репозитория
- ( ) Файлы, добавленные в индекс, не попадают в коммит

Правильно! Следующий вопрос

Что такое коммит?

- (•) Фиксация изменений, добавленных в индекс внутри репозитория
- ( ) Команда добавления файлов в индекс перед фиксацией
- ( ) Удаление файлов из репозитория

Правильно! Следующий вопрос

Можно ли закоммитить неотслеживаемые файлы?

- ( ) Конечно, любой новый файл после коммита сразу попадает в репозиторий
- ( ) Зависит от типа файла
- (•) Нет, сначала их нужно добавить в индекс

Правильно! Следующий вопрос

Как соотносятся репозиторий на Гитхабе и локальный репозиторий после выполнения связывания?

- ( ) Локальная копия — это неполноценная копия репозитория, находящегося на Github
- (•) Это два независимых репозитория. Локальный репозиторий может заливать изменения и забирать их из репозитория на Github.
- ( ) Это один и тот же репозиторий

Правильно! Следующий вопрос

Как отправить изменения в репозиторий на Github?

- ( ) git commit
- (•) git push
- ( ) git github
- ( ) git pull

Правильно! Следующий вопрос

Что делать в том случае, если локальный репозиторий удалён, но осталась копия на Github?

- ( ) git pull
- (•) git clone
- ( ) Восстановление данных невозможно. Создавать репозиторий с нуля

Правильно! Следующий вопрос

Что такое рабочая директория?

- (•) Файлы и директории находящиеся в директории проекта (там где находится каталог .git)
- ( ) Место куда попадают файлы после выполнения команды `git add`
- ( ) Место на диске, в котором нужно держать все разрабатываемые проекты

Правильно! Следующий вопрос

Можно ли восстановить файлы из репозитория после удаления их в рабочей директории?

- ( ) Нет, они автоматически пропадают из репозитория
- ( ) Только в том случае, если коммиты были добавлены на github
- (•) Можно, за исключением новых файлов, которых еще не было в репозитории

Правильно! Следующий вопрос

Какой командой восстанавливаются файлы?

- (•) git restore
- ( ) git add
- ( ) git init

Правильно! Следующий вопрос

Что означает `+` в выводе `git diff`?

- ( ) Удаленные строки
- (•) Добавленные строки
- ( ) Не измененные строки
- ( ) Техническая информация, которая нам не нужна

Правильно! Следующий вопрос

Какие файлы показывает `git diff` по умолчанию?

- ( ) Не отслеживаемые
- (•) Измененные в рабочей директории, но еще не добавленные в индекс
- ( ) Все изменения, которые были сделаны не зависимо от их состояния
- ( ) Добавленные в индекс

Правильно! Следующий вопрос

Показывает ли команда `git diff` неотслеживаемые (untracked) файлы по умолчанию?

- ( ) Да
- (•) Нет

Правильно! Следующий вопрос

Какая команда показывает историю коммитов?

- ( ) git show
- (•) git log
- ( ) git grep

Правильно! Следующий вопрос

Для чего нужна команда git blame?

- ( ) С ее помощью можно изучить изменения сделанные в конкретном коммите
- (•) Она показывает информацию о том, кто и в каком коммите менял строчки в указанном файле
- ( ) Она указывает на того, кто в команде пишет кривой код

Правильно! Следующий вопрос

Что вернет команда `git grep hexlet`

- ( ) Список всех коммитов, в описании которых есть подстрока *hexlet*
- (•) Список строк во всех файлах репозитория, в которых есть подстрока *hexlet*
- ( ) Список файлов, в которых встречается слово *hexlet*

Правильно! Следующий вопрос

Какая команда перемещает изменения, сделанные в файле, из индекса в рабочую директорию?

- ( ) git restore
- (•) git restore --staged
- ( ) git clean

Правильно! Следующий вопрос

Почему команда git restore не работает для неотслеживаемых файлов?

- ( ) Если долго вызывать, то сработает!
- ( ) Потому что для таких файлов нужно вызывать команду `git restore --staged`
- (•) Неотслеживаемые файлы невозможно восстановить, у них нет старого состояния

Правильно! Следующий вопрос

В каком состоянии должен быть файл, чтобы сработала команда `git restore <file>`

- ( ) Новый неотслеживаемый файл
- (•) Файл, измененный в рабочей директории
- ( ) Измененный и добавленный в индекс файл

Правильно! Следующий вопрос

Выберите верные утверждения

(нужно выбрать все корректные ответы)

- [x] Самый простой и безопасный способ делать изменения коммитов в гит, это создание новых коммитов
- [ ] Неправильные коммиты нужно удалять, чтобы не засорять историю
- [x] Если изменения были уже отправлены во внешний репозиторий, то изменять их следует с помощью новых коммитов, а не правки старых.

Правильно! Следующий вопрос

Что делает команда revert?

- ( ) Удаляет последний коммит в репозитории
- (•) Создает новый коммит, в котором выполнены изменения обратные указанному в команде коммиту
- ( ) Очищает рабочую директорию от изменений

Правильно! Следующий вопрос

С помощью какой команды можно физически удалить последний коммит?

- (•) git reset --hard HEAD~1
- ( ) git revert --hard HEAD~1
- ( ) git reset --hard

Правильно! Следующий вопрос

Выберите верное утверждение

- ○ amend не меняет последний коммит, вместо этого создается новый
- ● amend меняет последний коммит (откатывает его и создает новый)
- ○ amend меняет любой указанный коммит

Правильно! Следующий вопрос

В каком случае использовать `git commit --amend` безопасно?

- ○ В любом
- ○ Пока в локальном репозитории не более одного нового коммита (по сравнению с удаленным репозиторием на github)
- ● Пока изменения не запушены на github

Правильно! Следующий вопрос

Сколько раз можно добавлять изменения с помощью `--amend` в последний коммит?

- ○ Два раза
- ○ Один раз
- ● Любое количество раз

Правильно! Следующий вопрос

Что нужно делать перед коммитом?

- ● Изучить сделанные изменения с помощью команды `git diff --staged`
- ○ Просматривать последние изменения с помощью команды `git log -p`
- ○ Ставить свечку

Правильно! Следующий вопрос

По какому принципу нужно формировать коммиты

- ○ Один коммит – один файл
- ● Один коммит, одно логически связанное изменение (возможно одна задача)
- ○ В коммит должны попадать все сделанные изменения в рабочей копии

Правильно! Следующий вопрос

Какой флаг используется для интерактивного добавления изменений в индекс?

- ○ -a
- ○ -m
- ● -i

Правильно! Следующий вопрос

Что делает команда `git checkout`?

- ● Загружает в рабочую директорию состояние репозитория на момент нужного коммита
- ○ Выводит на экран изменения, сделанные в конкретном коммите
- ○ Откатывает изменения, сделанные в указанном коммите

Правильно! Следующий вопрос

Как быстрее всего вернуться обратно после чекаута?

- ● git switch -
- ○ rm && git clone
- ○ git checkout

Правильно! Следующий вопрос

Какой командой можно узнать текущий загруженный коммит?

- ○ git add
- ● git branch
- ○ git checkout

Правильно! Следующий вопрос

Как обозначается последний коммит в ветке, загруженной в рабочую директорию?

- ● HEAD
- ○ LAST
- ○ REST

Правильно! Следующий вопрос

Выберите верное утверждение

- ○ Каждый коммит имеет ссылку на дочерний коммит
- ● Каждый коммит имеет ссылку на родительский коммит
- ○ Коммиты сами по себе, они ни на кого не ссылаются

Правильно! Следующий вопрос

Выберите верное утверждение

- ○ Git невозможно использовать в командной разработке без создания веток
- ● Параллельную разработку в Git можно осуществлять с помощью веток, что позволяет не пересекать их код между собой
- ○ Git не позволяет работать параллелльно

Правильно! Следующий вопрос

Выберите верные утверждения

(нужно выбрать все корректные ответы)

- [ ] Файл *.gitignore* не нужно добавлять в репозиторий
- [x] Файл *.gitignore* нужно создать самостоятельно
- [ ] Команда `git init` создает файл *.gitignore* автоматически
- [x] Файл с описанием правил игнорирования должен называться *.gitignore*

Правильно! Следующий вопрос

Что произойдет, если добавить в *.gitignore* файл, который уже был в репозитории?

- (•) Ничего
- ( ) Файл удалится из репозитория
- ( ) Файл удалится в рабочей директории

Правильно! Следующий вопрос

Какие файлы обычно игнорируют?

(нужно выбрать все корректные ответы)

- [x] Файлы, создаваемые редакторами
- [x] Временные файлы
- [ ] Файлы с кодом
- [x] Зависимости

Правильно! Следующий вопрос

Что будет проигнорировано, если содержимое *.gitignore* выглядит так?

```
node_modules/
```

- (•) Все директории *node_modules* внутри репозитория на любом уровне
- ( ) Директория *node_modules*, находящаяся в корне проектов
- ( ) Первая попавшаяся в репозитории директория *node_modules*

Правильно! Следующий вопрос

Допишите *gitignore*, чтобы игнорировалась только директория *node_modules/* которая находится в корне проекта

```
access.log
/node_modules/
/coverage/
```

Правильно! Следующий вопрос

В какой ситуации может понадобиться `git stash`

- ( ) Когда изменения не жалко потерять
- (•) Когда нужно сделать быстрый коммит с исправлением, но не добавляя туда код, над которым идёт работа прямо сейчас
- ( ) Когда надо разбить изменения на несколько коммитов

Правильно! Следующий вопрос

Сколько раз можно сохранять изменения в stash?

- ( ) Только два раза
- ( ) Только один раз
- (•) Сколько угодно раз

Правильно! Следующий вопрос

Как правильно извлекаются последние изменения, добавленные в Stash?

- ( ) git stash
- (•) git stash pop
- ( ) git stash push

Правильно! Следующий вопрос