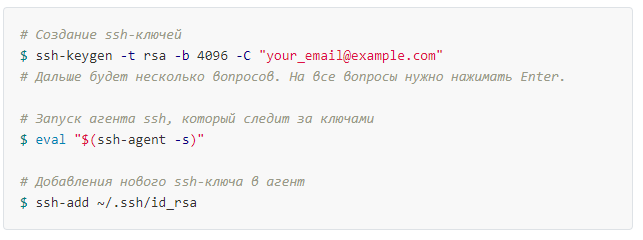
**Алгоритм создания GIT репы + команды для работы:**

1. Установить GIT и прописать 2 строчки для конфига (чтобы видели кто изменял файлы):
   1. **git config --global user.name “Demyankov Vladislav”**
   2. **git config --global user.email “**[**demyankov@devopser.by**](mailto:demyankov@devopser.by)**”**
   3. **cat .gitconfig** – для проверки (/c/Users/пользователь/.gitconfig)
2. Создаем базу данных git для созданной локальной папки репы:
   1. **git init MYproject**
3. Добавляем файлы и делаем снимок (commit):
   1. **git add file.txt** (**.** or **\*** – чтобы добавить все файлы)
   2. **git commit -m “V1”**
   3. **git log** – для просмотра истории снимков
   4. **git checkout** **“имя”** - для возращения файла в исходное состояние до переноса его на этап stage
4. Заходим на github создаем репу,синхронизируем ее с локальной и закидываем изменения локальной копии:
   1. **git remote add origin url**
   2. **git push -u origin master**
5. Клонируем репу с github на локальный компьютер:
   1. **git clone url-ссылка репы**
6. Клонируем репу с github на комп, создаем ветку fix, изменяем в ней файлы и добавляем в ветку master, удаляем ветку fix:
   1. **git clone url**
   2. **git checkout -b fix** – создали и перешли в ветку fix
   3. **git checkout имя-ветки** – переход между ветками
   4. **git branch** – показывает на какой ветки находимся
   5. **git merge fix** (объединяем ветки, должны находиться на ветки master)
   6. **git branch -d fix** (удаление ветки fix, должны находиться на ветки master)
   7. **git branch -D имя-ветки** (если изменения в ветки уже ненужны, но вы сделали коммит)
7. Как вернуться на прошлый снимок:
   1. **git checkout хэш-снимка** (берем с git log)
   2. **git checkout master** –переходим на самый последний снимок
8. Изменение файлов и имя снимка без создания нового:
   1. Изменяем файлы
   2. **git commit --amend** (нажимаем insert, изменяем, ESC, нажимаем : wq!
9. Для полного возвращения к прошлому снимку:
   1. **git reset --hard HEAD~число-насколько снимку вернуться назад**
10. Для удаления снимков но с сохранением текущих файлов:
    1. **git reset --soft HEAD~число-сколько снимков удалить**
11. Добавления своей ветки с изменениями для проверки и последующей синхронизацией другим человеком:
    1. **git push --set-upstream origin имя-ветки**
    2. Заходим на github нажимаем compare and pull request
    3. Заходим в pull request, оставляем коммент и нажимаем creat
    4. После того как другой человек нажал merge, удаляем локальную и удаленную ветку (если он не удалил)
    5. **git branch -d имя-ветки** (удаление с ветки master) –локальную репу удаляем
    6. **git push origin --delete имя-ветки** –удаленную ветку удаляем

**Полезное от курса ru.hexlet.io**

**Создание SSH ключей**



**git pull --rebase -** каждый раз при начале работы

**git config --global pull.rebase "true"** – для прописи команды выше (по умолчанию false).

**Для работы по ssh нужно на компе добавить**

В **sudo nano ~/.ssh/config** следующие строки:

*Host github.com*

*Hostname ssh.github.com*

*Port 443*

Проверяем соединение: **ssh -T** [**git@github.com**](mailto:git@github.com)

**git restore 'имя файла' –** восстанавливает файл из последнего коммита в исходном состоянии, **git restore --staged ‘имя файла’ –** позволяет откатиться до шаг stage.

**git diff** – показывает разницу между неотслеживаемыми файлами (если нужно и в stage, то с ключом **--staged**).

**git log -p** - показывает все коммиты, а также ключ -p показывает изменения (diff). **--oneline** – ключ выводит все комииты строчкой хэш и название коммита.

**git show ‘хэш’** – показывает изменение по определенному коммиту, хэш берем из git log и хватает только первые 8 символов.

**git blame ‘имя файла’ –** показывает изменения файла, кто изменял и в каком коммите.

**git grep -i ‘что ищем’** - показывает все файлы в который совпадает искомое. (ключ -i поиск без учета регистра).

**git clean -fd** - очищает все незакомиченные данные и возвращает на состояния последнего коммита.

**git revert ‘хэш’** - отменяет последний коммит.

**git reset --hard HEAD~** - отменяет последний коммит(делать только если этого коммита нет в github) **--hard** означает полное удаление (Без него git reset отменит коммит, но не удалит его, а поместит все изменения этого коммита в рабочую директорию), **HEAD~** означает "один коммит от последнего коммита".

**git commit --amend --no-edit** – если забыли добавить что-то в коммит (--amend), --no-edit – для того чтобы не заходить в редактор.

**git add -i** – переход в интерактивный режим.

**git checkout ‘хэш’ -** перейти в один из коммитов. (или вернуть файл к состоянию коммита: **git checkout ‘хэш'-- ‘имя файла'**)

**git switch -** -перейти быстро на главную ветку.

**git branch** – показывает в какой мы ветки и коммите находимся. (**git switch branch** – переключение между ветками).

**git stash** - сохраняет измененные файлы в .git (чтобы их вызвать **git stash pop**).

**GIT UDEMY (полезное)**

**git add “\*.java” –** добавить все файлы в проекте с расширением .java

**git reset** имеет 3 режима (не может убрать untracted файлы):

* **soft** (удаляет коммиты, но изменения в них переводит в отслеживаемую зону)
* **mixed** (удаляет коммиты, но изменения в них переводит в неотслеживаемую зону
* **hard** (удаляет коммиты и изменения)

**git branch ‘имя-ветки’ (git branch -r** –показывает удаленные ветки) **–** создание ветки без перехода в нее.

**git fetch** – обновляет удаленную ветку на локальном компьютере, но не обновляет локальную.



**git rebase** - делает слияние веток как и merge, но без merge коммитов (ключи **--continue** и **--abort** позволяют соответственно продолжить или отменить слияние веток; **--skip –**пропускает коммит который дает конфликт.

**git cherry-pick ‘хэш’** – используется когда нужно взять один или несколько коммитов из другой ветки в нашу ветку. (ключ **--edit** –позволяет изменить имя коммита; **--no-commit** – позволяет не добавляя коммит вытянуть файлы в отслеживаемые).