# **------------------ GIT -----------------**

# **Basics Git**

**git version git --version** проверка версии git

**git init** создание локального репозитория

**git help** выводит все команды git

# **Git Config**

**git config** настройка самого git

**git config --system** настройка git для всех пользователей текущей системы

**git config --global** настройка git для конкреного пользователся системы

**git config** или **git config --local** настройка git для текущего репозитория

# **Main**

**Git Status -> Git Stage -> Git Commit**

**git status** показывает изменения в дереве, отслеживаемые и неотслеживаемые файлы git , которые будут или не будут добавленные в git

**git add .** добавление всех файлов в stage(черновая область) для последующего их коммита

**git add my\_file** добавление только указаного файла в область stage (черновая область)

**git commit -m “my\_text”** создание коммита с описанием. Команда git commit -am “my\_text” также автоматически делает git add ы.

# **Git Information**

**git log -> Q** показывает созданные комиты

**git log -p** выводит все коммиты с более подробным выводом

**git show f3e1j61x01sfо5bz1** показывает детальную информацию про указанный коммит

**git reflog** выводит все изменения в проекте с момента создания ( движения, коммиты, ветки)

**git diff** выводит изменения в файлах, которые были после последнего коммита , было +++ добавление кода в файле , было --- удаление кода в файле . Изменения которые еще не находяться в stage

**git diff --staged** выводит изменения в файлах которые, уже добавленнные в stage

**git branch** показывает все ветки

# **Git Come Back**

**git restore .** возвращате все файлы до состояния предидущего коммита (еще не были добавлены в stage через git add .)

**git restore my\_file** возвращает указаный файл до состояния предидущего коммита коммита (еще не был добавлен в stage через git add my\_file)

**git restore --staged .** выбрасывает все измененные файлы из stage ( потом нужно писать git restore .)

**git restore --staged my\_file.txt** выбрасываеть указаный файлы из stage ( потом нужно писать git restore my\_file.txt)

**git checkout my\_file.txt** возвращает только указаный файл до состояния последнего коммита в текущей ветке

# **Actions**

**git mv my\_file.txt new\_name.txt** переименование файла

**git rm my\_file.txt** удаление файла из stage + и из проекта

**git rm --cached my\_file.txt** удаление файла из stage , а в проекте остаеться. Нужно прописывать, если файл уже помещен в stage , но у же и есть в .gitignore чтобы выкинуть из етого stage

# **Moving - Creating - Deleting**

**git checkout f3e1j61x01sfо5bz1** перемещение на указаный коммит - смещение указателя текущей ветки

**git checkout name\_branch** переключение на указаную ветку на последний коммит

**git commit -m “my\_text”** создание коммит с указаным описанием

**git branch name\_branch** создание ветки с указаным названием

**git checkout -b name\_branch** создание ветки из текущего коммита и сразу перемещение на нее ( тоже самое git branch name\_branch --> git checkout name\_branch)

**git checkout -b name\_branch f3e1j61x01sfо5bz1** создание ветки из указаного коммита и сразу перемещение на нее ( тоже самое git checkout f3e1j61x01sfо5bz1 --> git branch name\_branch)

**git reset --soft f3e1j61x01sfо5bz1** удаление всех коммитов в ветке до указаного, изменения и stage коммита от которого отходили остаються

**git reset --hard** **f3e1j61x01sfо5bz1** удаление всех коммитов в ветке до указаного, изменения и stage коммита от которого отходили удаляються

**git reset --soft HEAD~1** или **git reset --hard HEAD~2** удаление количества указанных коммитов в ветке

**git branch -d name\_branch** удаление указаной ветки (нужно проводить слияние перед удалением)

**git branch -D name\_branch** удаление указаной ветки ( не нужно проводить слияние перед удалением)

# **Merge - Rebase - Chery Pick**

**git merge name\_branch** берем изменения из указаной ветки и перемещаем в текущую

**git merge --abort** отмена последнего слияния, используеться когда возникли кофликты при попытке слияния

**git rebase master(**или **name\_branch)** перебазируем коммиты из указаной ветки в текущую не производя merge, перестановка производится про хронологии веток. Указатель начинает указывать на последний зделанный коммит. Хеши коммитов изменяються (не рекомендуеться использовать)

# **Git Tag**

**git tag name\_tag** создание легковесного тега на текущем коммите

**git tag -a name\_tag -m “my\_text”** создание анотированого тега с текстом в текущем коммите

**git tag** выводит все теги

**git show name\_tag** показывает информацию про тег и коммит на котором он стоит

**git checkout name\_tag** переключение на указаный коммит по тегу

**git tag -d name\_tag** удаление указаного тега

**git push origin name\_tag** отправление указаного тега в удаленный репозиторий (нужно делать самому)

**git push origin --tags** отправление всех тегов удаленный репозиторий

# **Git Ignore**

**.gitignore** внутри файла хранться файлы, которые не будут добавляться в stage

**my\_file.txt** не будет записываться в stage указаный файл

**\*.txt \*.log** не будут записываться в stage все файлы с указаным розширением

**\*.txt !my\_file.txt** не будут записывать все файлы с указанм окончанием, кроме второго

**my\_folder/** в stage не будет записываться целая папка с внутри лежащими файлами

**my\_folder / my\_file.txt** не будут записываться в stage только файл с указаным названием, который только лежит в указаной папке

**#** коментарии в файле .gitignore

# **GitHub**

**origin** название удальоного репозитория

**HEAD ->** **origin / master** или **HEAD -> origin/name\_branch** показывает где остановились указатели удаленого репозитория

**git remote add origin(**или **name\_repository) url\_adress** устанавливает связь между локальным репозиторием и удаленным. Даем уникальное названия для удаленого репозитория name\_repository (по умолчанию origin)

**git remote -v** показывает все подключения к удаленным репозитория

**git remote remove origin** отключить доступное подключение к удаленому репозиторию

**git push origin name\_remote\_branch** берет текущую ветку и закидывает в указаную уделенную ветку в удаленый репозиторий

**git push origin master -f**  полностью изменяет историю удаленной ветки на текущую локальную ветку

**git push -u origin name\_branch** пушим текущие коммиты не в ветку по умолчанию , а в указаную ветку name\_branch

**git pull** **origin name\_remote\_branch** витягиваем указаную удаленую ветку из удаленого репозитория в текущую локальную ветку

**git pull = git fetch + git merge**

**git fetch** загружает недостающие удаленные коммиты в текущую локальную ветку , не перемещает указатель текущей локальной ветки (иcпользуеться при конфликтах при git pull) (после нужно делать git merge name\_branch --> git push origin master)

**git push --set-upstream origin master (** или **name\_branch)** соединяем едаленую и локульную ветку с одними и темиж названиями, чтоб можна было прописывать git push(pull) вместо git push(pull) origin master (или name\_branch). Нужно связывать каждую локальную ветку с каждой удаленной

**git clone url\_adress** копирует репозиторий и как папку помещает в текущий путь (удаленное соденение уже уствновленно)

**git clone url\_adress .** клонирование репозитория без внешней папки

**git clone url\_adress my\_name** копирует репозиторий и изменяет название главной папки

# **---------------- Doker --------------**

# **Info**

**docker images** или **docker image ls** показываеть все сохраненыне образ(docker images) на етом пк

**docker ps** или **docker ps -a /** **docker container ls** или **docker container ls -а** показывает все запущение контейнеры в текущий момент

**docker info** выдает всю информацию про docker и images и containers в нем

**docker my\_command --help** выводить всю информацию об указаной команде

# **Actions**

**docker build .** создает doker image из нашого dokerfile , . - означает что doker file находится в текущей директории.

**docker build ./name\_folder** создание докер образа, когда Dockerfile лежит в другой директории

**docker build .** **-t my\_image** или **docker build .** **--tag my\_image** тоже создаем docker image из Dockerfile и еще даем ему свое название

**docker build . --file Dockerfile** создание докер образа из докер файла при явном указаныи названия докер файла

**docker run my\_image** или **docker run --name my\_name my\_image** создание нового контейнера из указаного docker image и сразу запуск (не нужно приписывать docker start) . Можно дополнительноу указывать имя для контейнера

**docker run -d** **my\_image** запус контейнера в фоновом режиме

**docker run -p 80 : 80 my\_image** или **docker run -p 127.0.01:80:80** -p связиивание портов, первоче число - адрес\_порт машини которая будет подключаться к docker контенеру , второе число - порт взятой програмы внутри контейнера docker. Также можна указывать ip локальной машины

**Первый порт** для обращение от пользователей к контейнеру

**Второй порт** внутри контейнера , от которого идет ответ для пользователя от контейнера. Также второй порт служит для связывания текущего контейнера с другими контейнерами через docker-compose

**docker run -it my\_image** запус контейнера из образа в интерктивном режиме

**docker tag old\_name new\_name** переименование docker образа

**docker logs my\_container\_id** возвращает данные про запущенный контейнер(на каком ip , на каком порту)

**docker exec -it my\_container\_id**  **/bin/sh** заходум внутрь запущеного контейнера

# **Already Exist Container**

**docker start my\_container** запуск уже существующего контейнера

**docker stop my\_container** остановка выполнения контейнера (удаляеться кеш данных)

**docker restart my\_container**  перезапуск контейнера

**docker pause my\_container** ставим контейнер на паузу (не удаляеться кеш данных)

**docker unpause my\_container** выходим из паузы контейнера

# **Deleting**

**docker rmi my\_image** или **docker image rm 0x31233** удаляет указаный образ по image id

удаление doker image проекта

**docker rmi -f my\_image** или **docker rmi -f** **0x31233** удалить image, если даже у него есть запущенные контейнеры

**docker kill my\_container** выход с контейнера с ошибкой (если контейнер не отвечает)

**docker container rm my\_container** или **docker rm 90a26aa89259** удаление контейнера

**docker system prune -a** удаление все образ на етом пк

**Остановка Контейнера -> Удаление всех контейнеров даного Image (образа) -> Удаление самого Image (образа)**

# **Docker-Compose**

**docker-compose build** создаем docker образы для наших сервисов

**docker-compose up** и собрать и запустить все docker контейнеры

**docker-compose up -d** запуск контейнеров из уже созданого docker-compose image в фоновом режиме

**docker-compose down** остановка контейнера и следовательно его удаление

**docker-compose ps** показывает запущеные контейнеры

# **.dockerignore**

# **DockerHub**

**docker login** авторизация пользователя в терминале через акаунт docker hub

**docker logout** выход из учетной записи пользователся в терминале

**docker push my\_image** отправляем свой docker image в репозиторий. При етом локальное название docker image и название репозитория должно иметь одно название

**docker push my\_image:latest** можна явно указывать версию образа, который отправляем

**docker pull my\_image** скачивание сторонего Docker Образа, который будет основой нашего приложения

**docker pull my\_image:latest** можна явно указывать версию образа, который скачиваем

# **Dokerfile**

**FROM python:3.12-slim** берем другой docker image(образ) на основе которого будет запускаться наш проект. Ето образ нужно перед ети скачать через docker pull

**WORKDIR usr/src/app** установка рабочей директории внутри, которой будет выполняться такие команды как COPY , RUN , CMD

**.**  означает текущая директория

**COPY . app/** или **ADD . app/** копирование файлов из нашего проекта в docker image(образ)

**RUN** команда выполняеться один раз при создании docker image

**EXPOSE 8000** указыват порт внутри нашего контейнера

**CMD** **python test.py**  команды могут выполняеться много раз внутри запущеного контейнера