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

# **Basics Git**

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

**git init** создание локального репозитория

**git status** выводит информацию о моей рабочей директории

**git help** выводит все команды git

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

**.gitignore -> text.txt** в созданом файле записывает названия файлов, которые не будет отслеживать git и не будет добавлять в коммит. Работает только после создания файла (потом не будет)

**.gitignore -> \*.txt \*.log** будут записаны все файлы с указаными окончаниями

**.gitignore -> \*.txt !new.txt** будут записаны все файлы с окончанием, но кроме указаного файла new.txt

**.gitignore -> my\_folder/\*** будет записаня вся папка для игнора с лежащими внутри файлами

# **Staging Area**

**git add .** добавление всех файлов в stage для последующего их коммита

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

**git ls-files** выводи файлы которые лежать в stage

**git restore my\_name** востановление удальоного файла из stage, когда уже удалили из проекта. Также откат изменений указаного файла до предидущего коммита

**git add my\_name** или **git rm my\_name**  полное удаление файла из stage, когда уже удалили из проекта

**git restore --staged my\_name** убирает файл из stage, если сами добавили ошибочно

**git clean -dn** показывает какие новосозданные неотслеживаемые (untracked) файлы будут удалены, не были добавленные в stage

**git clean -df**  удаляет новосозданые неотслеживаемые (untracked) файлы которые, не были добавленные в stage

# **Commit**

**git commit -m “my\_text”** создание коммита

**git log** **-> Q** возвращает все коммиты текущей ветки в терминал (снизу в вверх)

**git checkout id\_commit** переходит в указанный коммит в любой ветке по его индексу

**git checkout .** или **git restore .** возвращает все файлы к состоянию пердидущему коммиту, если файлы не были добалены в stage

**git checkout my\_name** или  **git restore my\_name** возвращает указанный файл к состоянию предидущего коммита, если файл не был добавлен в stage

**git reset -> git checkout .** или **git checkout my\_name** возвращает файлы (или указанный) к состоянию предидущего коммита даже если он уже был добавленный в stage

**git reset --soft HEAD~1** или **git reset HEAD~1** мягкое удаление одного (можна указывать) коммита и stage файлов. Возможно отменить удаление коммитов. Файлы в рабочей директории не изменяються

**git reset --soft HEAD~1** или **git reset HEAD~1 -> git commit -m “text”** отменяет удаленние коммитов

**git reset --hard HEAD~1** тяжолое удаление одоного(можно указывать) коммита и фалов из stage. Невозможно отменить удаление коммитов. Файлы из рабочей директории удаляютсья

**git reset --hard f88430a** возвращает удаленный коммит, номер из git reflog

# **Branch**

**git branch** показывает все ветки в текущем репозиторее

**git branch my\_branch** создание новой ветки, в названии нельзя использовать пробелы

**git checkout my\_branch** или **git switch my\_branch** переходит в указаную ветку

**git checkout -b my\_name** или **git switch -c my\_branch** создание новой ветки и переключение на нее

**git branch -d my\_name** удаляет ветку, перед удаление нужно **делать** слияние

**git branch -D my\_name** удаляет ветку, перед удаление можна **не делать** слияние

**git checkout id\_commit -> git add . -> git commit -m “text” -> git branch my\_name->**

**-> git branch my\_name**  создание новой ветки из изменьоного выбраного коммита на котором стоял указатель

**git reflog -> get checkout f88430a (id\_commit) -> git branch my\_name -> git checkout my\_name ->** востановление удаленной ветки через коммит етой ветки, id\_commit из git reflog

# **Merge**

**git merge second\_branch** forward-слияние с перемещение указателя

**git merge --squash second\_branch** forward-слияние без перемещения указателя

**git merge --no-ff second\_branch** рекурсивное слияние независимо от того, если ли паралельные коммиты которые отличаються

**git merge second\_branch** рекурсивное слияние git автоматически выполнит, если у обеих веток есть паралельные коммиты, которые отличаютсья

**git rebase** **second\_branch** переносит копии комитов из указаной ветки в текущую

**git cherry-pick id\_commit\_in\_second\_branch** тип слияния - создлается копия выбраного коммита из второй ветки и сливается с текущей веткой. В текущей ветке становится последним коммитом, на которого указывает указатель

**git merge --abort** отмена слияния веток для быстрого слияния

**git log --merge** показывает все слияния

**git diff** показывает какие файлы проходили последнее слияние

# **Stash**

**git stash** заберет в себя изменения, которые ще не были закомиченные - создание стеша без описания

**git stash push -m “text”** создание стеша с описанием

**git stash list** показывает все сохраненные стеши в проекте (снизу вверх, [ 0 ] - самый последний)

**git stash apply** вернет из себя последний стеш изменений назад в проект

**git stash apply 1** возвращает из себя указаный стеш по номеру списка.

**git stash pop 1** используем стеш по номеру списка и удаляем его из етого списка

**git stash drop 1** удаление стеша по номеру списка

**git stash clear** удаление все записей стеша

# **Tag**

**git tag** выводит все созданные в проекте теги

**git tag name\_tag id\_commit** создание тега

**git show name\_tag** выводить коммит в терминал по указаному тегу

**git tag -a name\_tag -m “text”** создание анотированого тега по соледнем сзделаном коммита + можна добавить описание

**git checkout name\_tag** перемещение указателя по тегам

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

# **GitHub**

**origin** название удальоного репозитория

**origin / master origin/name\_branch** названия удаленьоных веток в git

**git remote add origin url\_adress** устанавливает связь между локальным репозиторием и удаленным

**git --set-upstream origin master** создаем первую-главную master ветку в удаленом репозиторее (делается один раз вначале)

**git remote -v** показывает к каком удаленым репозитория я поключен

**git push origin name\_branch** пушим указаную ветку в удаленный репозиторий

**git pull** **origin name\_branch** перенос данных (коммитов - файлов - истории) из удаленого репозитория указаной ветки в локальный в такую же ветку

**git fetch** загрузка содержимого из удаленого репозитория (не производит слияние)

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

**git clone url\_adress** копирует репозиторий и как папку помещает в текущий путь

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

# **Подключение К Уже Существующей Ветке**

1. **git init**

2. **git remote add origin url\_adress**

3. **git pull origin master(как будет называться ветка)**

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