**Commands Short List**

**Навигация в BASH:**

pwd – показать текущее местоположение

ls – показать файлы в папке, кроме скрытых

ls -f – показать файлы в папке, включая скрытые

cd c:/ – перейти в данный каталог

cd- – вернуться назад

cd .. – выйти на 1 уровень вверх

cd ../.. – выйти на 2 уровня вверх

mkdir – создать папку

cd!$ – перейти во вновь созданную папку

mkdir -p {folder1,folder2} – создать сразу несколько папок folder1 и 2

mkdir -p folder/{subfolder 1,subfolder 2} – создать folder и сразу несколько подпапок subfolder1 и 2

touch file.html – создать файл

touch folder/{file.html,file.txt} – создать папку folder и несколько файлов

rm folder – удалить папку, только если она пустая

rm -r folder – удалить папку и файлы в ней

echo “abcdefg” > file.txt – перезаписывает файл новым содержанием “abcdefg”

echo “abcdefg” >> file.txt – добавляет в файл новое содержание “abcdefg”

cat file.txt – выводит текст из файла

q – закрытие встроенных текстовых программ

**Команды в GIT**

РЕПОЗИТОРИЙ

git.init – создание репозитория

git clone (URL) – копирование существующего репозитория, с текущим именем

git clone (URL)NEWNAME – копирование существующего репозитория, с новым именем NEWNAME

cat .git/config – инфо о содержимом

git config user.name – показывает логин

git config user.email – показывает имейл

git help config – помощь по config

/abcde – найти инфу abcde

n – переход вниз по найденному

<shift>n – переход вверх по найденному

ИЗМЕНЕНИЕ

git status – состояние каталога и раздела проиндексированных файлов

git add file.txt – добавляет файл в индекс

git add -all = git add **.** – добавляет всё в индекс, отслеживает и новые файлы и изменения файлов **без удаления**

git add -u – отслеживает изменения файлов и удаление, **без создания новых файлов**

git commit -m “text about commit” – («-m» message ОБЯЗАТЕЛЬНО!) берёт данные из индекса, сохраняет слепок в локальном репозитории и сдвигает маркер на этот слепок

git commit -a -m “text about commit” = git commit -am “text about commit” – («-a» add, «-m» message) берёт все изменения, МИНУЯ ИНДЕКС, сохраняет слепок в локальном репозитории и сдвигает маркер на этот слепок

git fetch – (приносит) показывает изменения В удалённом репозитории по сравнению с локальным

git pull – заливает изменения ИЗ удалённого репозитория в локальный (если в удалённом РЕПО к.л. или мы сделали изменения, и он уже не совпадает с последним снимком нашего локального, то при попытке запушить из локального в удалённый будет ошибка; сл-но перед push – надо обновить снимок pull)

git push = git push origin main – заливает изменения ИЗ локального репозитория в удалённый

git log – посмотреть историю коммитов

git log --author – посмотреть лог изменений определённого автора

git show adJuNKL34e33f(hash) – изменения, которые были в коммите hash № adJuNKL34e33f

git show – изменения, которые были в последнем или указанном коммите

git blame file.txt | grep “Abcde” – инфо об изменениях «Abcde», которые вносились в файл file.txt

git blame file.txt | grep UserName – инфо об изменениях, которые вносились в файл file.txt автором UserName

git diff – показывает изменения, которые были произведены в файле относительно локального репозитория

git reset – отменить git add

git reset HEAD~1 – откатывает изменения на 1 шаг назад

git reset HEAD Folder – откатывает изменения в папке Folder

git reset hard HEAD~1 – безвозвратно удаляет

git rm Folder – удаляет Folder из WORK DIRECTORY и ИНДЕКСА, и сохраняет изменения в индексе

(rm Folder = git rm Folder + git add)

git rm --cashed File.txt – удаляет файл из отслеживаемых в Индексе

git checkout – удаляет последние изменения в файле

git checkout **.** – удаляет последние изменения в файлах

git checkout path/to/file – привести измененный файл в начальное состояние (до изменения)

git stash – сохраняет последние изменения в последнем файле во временное хранилище

git stash pop – достаёт обратно последние изменения в последнем файле из временного хранилища + показывает инфу о временном хранилище

git stash clear – очищает временное хранилище

git apply path/to/file – применить патч в Git

git clean -f – удалить все не отслеживаемые файлы

git clean -fd – удалить все не отслеживаемые файлы и папки

СОЗДАНИЕ ВЕТОК

git branch NewBranch – создаёт новую ветку

git branch – инфа о локальных ветках и местонахождении среди них

git checkout NewBranch – переключается с ветки Main на NewBranch

git push -u origin NewBranch – («-u» - upstream) сохранит новую ветку в удалённом РЕПО (иначе не получится запушить изменения/файлы, т.к. в удалённый РЕПО не видит ветки NewBranch)

git fetch origin NewBranch – скачать ветку, если её нет в локальном репозитории, но есть в удалённом

СЛИЯНИЕ ВЕТОК

git checkout - – перейти с ветки NewBranch на ветку main

git merge NewBranch – ветка NewBranch сольётся с веткой Main в локальном РЕПО

git merge --abort – отменить слияние веток, если произошёл конфликт

git push – ветка NewBranch сольётся с веткой Main в удалённом РЕПО

git push -u origin NewBranch – исправить ошибку "fatal: The current branch my\_branch has no upstream branch", возникающую при вводе git push

Конфликт: в удалённом и локальном репозитории изменили там и там разные файлы

1. git branch 2nd
2. git checkout 2nd
3. touch File-2.txt
4. git add File-2.txt
5. git commit -am “2nd”
6. git push -u origin 2nd
7. git checkout main
8. git merge 2nd

**Error**

1. git pull
2. git push

Если конфликт: **You have not concluded your merge (MERGE\_HEAD exists)**, выполнить:

1. git add .
2. git commit -m “message”
3. git push
4. или
5. git merge --abort
6. или
7. Зайти в папку, открыть файл, стереть лишнее (будут помечены >>>>> и <<<<<), сохранить
8. git add File.txt
9. git commit -am “message”
10. git push

ПЕРЕИМЕНОВАНИЕ ВЕТОК

git branch -m first\_branch second\_branch – first\_branch переименуется в second\_branch

Конфликт: в ветке содержится файл и она была запушена, переименовать second\_branch в first\_branch

1. git branch first\_name
2. git branch -m first\_branch second\_branch – first\_branch переименуется в second\_branch
3. git checkout second\_branch – переходим на ветку
4. touch File.txt – создаём файл
5. git commit -am “message” – коммитим
6. git push -u origin second\_branch – отправляем в удалённый РЕПО
7. git branch -m second \_branch first \_branch – **ОК НО!** в удалённом РЕПО ветка НЕ переименуется
8. git push origin :second\_name – удалит ветку в удалённом РЕПО
9. git push -u origin first\_branch – отправляем в удалённый РЕПО ветку с новым названием

УДАЛЕНИЕ ВЕТОК

git branch -d NewBranch – удаляет ветку на локальном репозитории

git fetch origin NewBranch – возвращает ветку в локальный репозиторий, НО! чтобы она появилась (в списке git branch) на неё надо перейти: git checkout NewBranch