კონფიგურაცია ლოკალური:

1) git config --local user.name "beqa eqseulidze";

2) git config --local user.email "beqa@examle.com;

კონფიგურაცია გლობალური:

1)git config --global user.name "user name"

2)git config --global user.email "usermail"

კონფიგურაციების ნახვა: git config --list;

---------------------------------------------------------------

ზოგადი ბრძანებები:

git - გამოიტანს გიტის ბრძანებების სიას (ძირითადებს)

git status გიტთან დაკავშირებით ფაილების სტატუსის ნახვა;

git log დაკომიტებული ფაილების ნახვა ;

git clean -f წაშლის ანთრექთ სივრცეში არსებულ ყველა ფაილს წაშლა;

git clean -n გვაჩვენებს რომელი ფაილები წაიშლება git clean -f ბრძანებით;

--------------------------------------------------------------

git init გიტის ინიცირემა კონცრეტულ ფოლდერში

----------------------------------------------------------------------------------------------------

.gitignore - შექმნა და გამოყენება;

1) შევქმნათ გიტ იგნორ ფაილი და მასში ჩავწეროთ გიტის მიერ დასაიგნორებელი ფალები,ან ექსთენშენები მაგ: \*.js დააიგნორებს ყველა .js ფაილს

2) git add . სთეიჯინგ არეაში გადაიტანს გიტ იგნორ ფაილს

3) git commit -m”message text”

უკვე დაკომიტებული ფაილის გიტიგნორში ჩასმა:

1) ვამატებთ ფაილის სახელს გიტ იგნორის სიაში;

2) git add .gitignore

3) git commit -m”commit message”

4) git rm - -cached main.js ეს ბრძანება საბოლლოდ ამოშლის main.js ფილს გიტის ისტორიიდან;

5) git add . main.js ის წაშლა ქეშიდან დავამატოთ სთეიჯინგ არეაში

6) git commit -m”comit message” დავაკომიტოთ

---------------------------------------------------------------------------------------------------

დამატება სტეიჯინგ არეაში

git add file1 file2 -ერთი ან რამდენიმე ფაილის დამატება;

git add . ან git add --all ანთრექთში არსებული ყველა ფაილის დამატება;

git add \*.js ყველა js გაფართოების ფაილის დამატება;

git add folder1/\*.js კონკრეტული საქაღალდიდან ყველა js ფაილის დამატება;

git add "\*.js" პროექტში არსებული ყველაა ფოლდერიდან დაამატებს ყველა js ფაილს;

---------------------------------------------------------------------------------

git commit -m "my commit text" სტეიჯინგ არეაში არსებული ფაილების დაცომიტება ;

git commit -am "my commit text" მოდიფაი (ანთრექთ ფაილებზე არ მუშაობს) არეადან ერთი ბრძანებით დაკომიტება

git commit --amend -m "new message" ბოლო კომიტის და კომიტ-მესიჟის ცვლილება(ერთად ან ცალცალკე);

------------------------------------------------------------------------------

git diff ადარებს მოდიფაიდ სივრცეში არსებულ ფაილებს ბოლო კომიტის დროს არსებულს მდგომარეობას;

git diff --staged ადარებს სეიჯინგ არეაში არსებულ ფაილებს ბოლო კომიტის დროს არსებულ მდგომარეობას;

git diff COOMIT\_iD ბოლო პარამეტრად გადაეცემა კომიტის ID და ამ კომიტს შეადარებს ბოლო კომიტს;

--------------------------------------------------------------------------------------

კომიტების გაუქმება :

git reset --soft HEAD~2 გააუქმებს ბოლო 2 კომიტს და გადაიტანს მათ სტეიჯინგ(მწვანე) არეაში;

gti reset --mixed HEAD~2 გააუქმებს ბოლო 2 კომიტს და გადაიტანს მათ ანთრექთ(წითელ) არეაში. ეს ბძანება შეიძლება გამოვიყენოთ --mixed პარამეტრის გარეშეც

git reset --hard HEAD~1 გააუქმებს და წაშლის ბოლო 1 კომიტს ანუ ცვლიებები წაიშლება როგორც კომიტებიდან ასევე მთლიანად პროექტიდან;

git reset თუ არსებობს დასაკომიტებელი ფაილები სტეიჯინგ(მწვანე) არეაში მათ დააბრუნებს ანთრექთ(წითელ) არეაში;

------------------------------------------------------------------------------------------------------------

გადაადგილება კომიტებს და ბრენჩებს შორის:

git checkout სხვადასხვა ვარიაცებით გამოიყენება კომითებს შორის და ბრენჩებს შორის გადასაადგილებლად, ამდროს თუ დავაკომიტებთ ეს კომიტი შენახული არ იქნება და წაიშლება გიტისმიერ;

gti checkout HEAD~1 დაგვაბრუნებს წინა კომიტის დროს არსებულ მდგომარეობაზე სტატუსი გვექნება detauched,რაც ნიშნავს რომ ჩახსნილები ვართ ბოლო კომიტს და ამჟამათ ვიმყოფებით სხვა

კომიტის მდგომარეობაზე ან სულაც სხვა ბრენჩზე(რაც ჩანს git branch ბრძანებით);

git checkout main(ან სხვა სახელი რომელსაც ვიყენებთ ძირითად შტოთ) გვაბრუნებს მაინ შტოს ბოლო კომიტზე;

git checkout HEAD~2 --test.txt შევა 2 კომიტის წინ არსებულ მდგომარეობაში აიღებს ფაილს (ამშემთხვევაში text.txt) და ჩასვავს მიმდინარე სიტუაციაში;

git checkout -- . გააუქმებს ყველა ცვლილებას პროექტში რომელიც არის მოდიფაი არეაში( ანთრექთ სთეიჯ არეას ვერ ეხება) და ყველა ფაილს დააბრუნებს ბოლო კომიტის მდგომარეობზე;

git checkout "feature" გადავყავართ მითითებულ ბრენჩზე(ამ შემთხვევაში "feature" ბრენჩზე)

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**ssh key -ს აწყობა და კავშირი გიტჰაბთნ**

გავიაროთ შემდეგი სტეპები:

ls -al ~/.ssh ვამწმებთ გვაქვ თუ არა SSH კავშირი გიტჰაბთან ;(თუ არის რამოდენიმე ფაილის სიას გვაჩვენებს თუ არა და მხოლოდ 2 დაფარული ფოლდერი იქნება)

ssh-keygen -t ed25519 -C "your\_email@example.com" აგენერირებს SSH კავშირს (key-ის) , მეილის ველში მიეთითება თქვენი გიტჰაბის ექაუნთზე მითითებული მეილი; (ბრძანების გაშვების შემდეგ 3 კითხვას დაგისმევთ და შესაძლებელია მხოლოდ ენთერზე დაჭერით გაიაროთ)

eval "$(ssh-agent -s)" წინა ბრძანების გაშვების შემდეგ ეს ბრძანება ქმნის სსჩ აგენტს რომელის საშუალებითაც ხდება კავშირის დამყარება ,ბრძანება დაგვიბრუნებს Agent pid 13601-ს(რაც ნიშნავს რომ აგენტი შეიქმნა)

ssh-add ~/.ssh/id\_ed25519 ეს ბრძანება საბოლოოდ შექმნის თქვენს დაცულ კავშირს გიტჰაბ ექაუნთთან ,თუმცა ჯერ საჭიროა დაგენერირებული პაბლიქ key შევინახოთ გიტ ჰაბზე რისთვისაც ვაგრძელებთ შემდეგ სტეპებს

cd ~/.ssh გადავდივართ ssh საქაღალდეში

ls შევიხედავთ საქაღალდეში და დავაკოპირებთ (მაუსის გამოყენებით)იმ ფაილის სახელს რომელის სახელიც მთავრდება .pub -ით

cat file-name (file-name -ს ნაცვლად იქნება წინა სტეპზე დაკოპირებული ფაილის სახელი) ეს ბრძანება გახსნის ფაილს ტერმინალშივე და დავაკოპირებთ შეგთავსს მთლიანად დაახლოებით ერთი ან ორი ხაზი იქნება;

შემდეგი წავალთ გიტჰაბის ექაუნთზე შევალთ settings => SSH and GPG keys => new SSH key ჩავაკოპირებთ დავასეივებთ და შეგვეძლება გიტჰაბის გამოყენება ssh კავშირით;

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

git branch გვიჩვენებს აქტიურ ბრენჩებს (მწვანე არის რომელზედაც ბიმყოფებით);

git branch name ახალი ბრენჩის შექმნა NAME-ის ნაცვლად ჩავწერთ ბრენჩის სახელს რომელსაც ვქმნით;

git branch -d feature წაშლის მითითებულ ბრენჩს(ამ შემთხვევაში 'feature'),არ უნდა ვიყოთ იმ ბრენჩზე რომელის წაშლაც გვინდა;

git merge feature დამერჯავს feature ბრენჩს მშობელ ბრენჩთან (უნდა ვიდგეთ მშობელ ბრენჩზე) თუ დამერჯვამდე მშობელ ბრენჩზე არაფერი შეცვლილა დამერჯვა მოხდება fast forvard რეჟიმში ხოლო თუ

რაიმე ცვლილება დახვდება მოითხოვს ახალ კომიტ მესიჯის გაკეთებას;

git checkout -b feature შექმნის ახალ ბრენჩს სახელან feature-ს და გადავა მასზე(ბრენჩის სახელი რა თქმა უნდა შეიძლება ნებისმიერი იყოს რაც გვინდა);

-------------------------------------------------------------------------------------------------------------------------------------------------------

git remote add origin "git-repository-ssh-linki-here" - ლოკალურ რეპოზიტორიზე რემოუთ რეპოზიტორის ლინკის მიმაგრება;

git remote -v გვაჩვენებს ლოკალურ რეპოზიტორიზე მიმაგრებული რემოუთ რეპოზიტორის ლინკს ე.წ. "origin" -ს

git clone "git-repository-ssh-linki-here" ჩამოტვირთავს მითითებულ რეპოზიტორის ლოკალურად;

git push origin main ლოკალური რეპოზიტორიის რემოუთზე ატვირთვა (main რემოუთ რეპოზიტორიზე ბრენჩის სახელია და ის შეიძლება საერთოდ სხვა იყოს, origin პირველ სტეპზე შევქმენით და ისიც რა თქმა

უნდა სხვა სეხელით შეიძლებოდა და ეხლაც შესაბამისად იმ სახელით გამოვიყენებდით )

git branch -r გვიჩვებნებს გიტჩაბთან საკომუნიკაციო ბრენჩი(თუ გიტჰაბთან კომუნიკაცია არ გვქონია ჯერ ისე არჩანს);

git fetch ჩამოწერს რემოუთ რეპოზიტორიიდან განახლებებს და შეინახავს გიტჰაბთან საკომუნიკაციო ბრენჩზე რომელიც წინა სტეპზე ვახსენეთ და დაელოდება შემდეგ pull ბრძანებას ან უნდა გავაუქმოთ.

git pull ჩამოწერს გიტჰაბზე არსებულ ახალ ვერსიებს და დამერჯავს ჩვენს ძირითად შტოში (თუ )

-------------------------------------------------------------------------

git revert HEAD - რეპოსიტორს დააბრუნებს ერთი კომიტის წინა მდგომარეობაზე ისე რომ ბოლო კომიტსაც დატოვებს.