2025

ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

ვერსიონირება და უწყვეტი ინტეგრაცია

თემა პირველი: Git-ის მიმოხილვა, ფუნქციონალი და ინსტალაცია

ვერსიონირების სისტემების ტიპები

ვერსიონირების სისტემა (Version Control System - VCS) არის პროგრამული უზრუნველყოფა, რომელიც უზრუნველყოფს კოდის ცვლილებების მართვას, პროექტებთან მუშაობისას თანამშრომლობის გამარტივებას და პროექტის ისტორიის შენახვას. არსებობს ვერსიონირების სისტემის ორი ძირითადი ტიპი:

1. ცენტრალიზებული ვერსიონირების სისტემები (CVCS) – მონაცემთა ერთიანი საცავი, სადაც ყველა მომხმარებელი უკავშირდება ცენტრალურ სერვერს.

2. დისტრიბუციული ვერსიონირების სისტემები (DVCS) – მონაცემთა საცავის სრული ასლი შენახულია ყველა მონაწილის მოწყობილობაზე, რაც გვაძლევს ოფლაინ (offline) მუშაობის საშუალებას და დეცენტრალიზირებული სტრუქტურის გამო პროექტის უსაფრთხოებას უზრუნველყოფს.

GIT

Git შეიქმნა 2005 წელს ლინუს ტორვალდსის მიერ, Linux-ის ბირთვის (განვითარების პროცესის გასამარტივებლად. ის არის დისტრიბუციული ვერსიონირების სისტემა (DVCS), რომელიც სწრაფი, ეფექტური და მოქნილია.

Linux-ის ბირთვის განვითარებისთვის, დეველოპერები იყენებდნენ **BitKeeper**-ს კომერციული ვერსიონირების სისტემას. თუმცა, 2005 წელს BitKeeper-ის მფლობელმა თავისი პროდუქტი ფასიანი გახადეს, რამაც Linux-ის დეველოპერებს პრობლემები შეუქმნა. ამ სიტუაციის საპასუხოდ, ლინუს ტორვალდსმა გადაწყვიტა შეექმნა “თავისუფალი და ღია წყაროს (open source) ვერსიონირების სისტემა”.

GIT-ს დღესაც აქტუალური ტექნოლოგია რომელიც გამოიყენება მსოფლიო მასშტაბით, რომელსაც მილიონობით მომხმარებელი ჰყავს. დრეს ის უკვე კარგად ნაცნობი ტექნოლოგიაა თუმცა 2000-იანი წლების დასაწყისში მან პოპულარულობა მოიპოვა ისეთი უპირატესობების გამო როგორიცაა:

- **Snapshot სისტემა** – ცვლილებისას ინახება პროექტის სრული სტრუქტურა და არა ცალკეული ცვლილება.
- **დისტრიბუციული არქიტექტურა** – კოდის სრული ასლი ინახება თითოეულ მომხმარებელთან.
- **Branching და Merging** – განსხვავებული განშტოების შექმა და გაერთიანება მარტივია.
- **დატა ინტეგრაციის მექანიზმები** – უზრუნველყოფს ეფექტურ პარალელურ მუშაობას და კონფლიქტების მოგვარებას.
- **კომპაქტური და სწრაფი** – მონაცემთა არქივაციის მექანიზმების წყალობით, Git სწრაფად მუშაობს.

დაყენების ინსტრუქცია

1. გადმოწერეთ GIT - https://git-scm.com/downloads
2. დააინსტალირეთ GIT. (ინსტალაციისას კონფიგურაციაში ცვლილებების შეტანა არ არის საჭირო)
3. ჩართეთ GIT Bash.

ინსტალაციის შემდეგ, პირველ რიგში უნდა დააყენოთ თქვენი მომხმარებლის მონაცემები (ეს მონაცემები იქნება გამოყენებული ყველა შემდგომ პროექტში ავტომატურად ცვლილების შესრულებისას).
ამისთვის GIT Bash-ში ჩაწერეთ შემდეგი ბრძანებები:

git config --global user.name "თქვენი სახელი"

git config --global user.email "თქვენი ელ. ფოსტა"

კონფიგურაციის შესამოწმებლად გამოიყენეთ შემდეგი ბრძანება:

git config --list

თუ კონკრეტული პროექტისთვის დაგჭირდებათ მომხმარებლის შეცვლა ისე რომ გლობალური მომხმარებელი არ შეცვალოთ, ჩაწერეთ შემდეგი ბრძანებები:

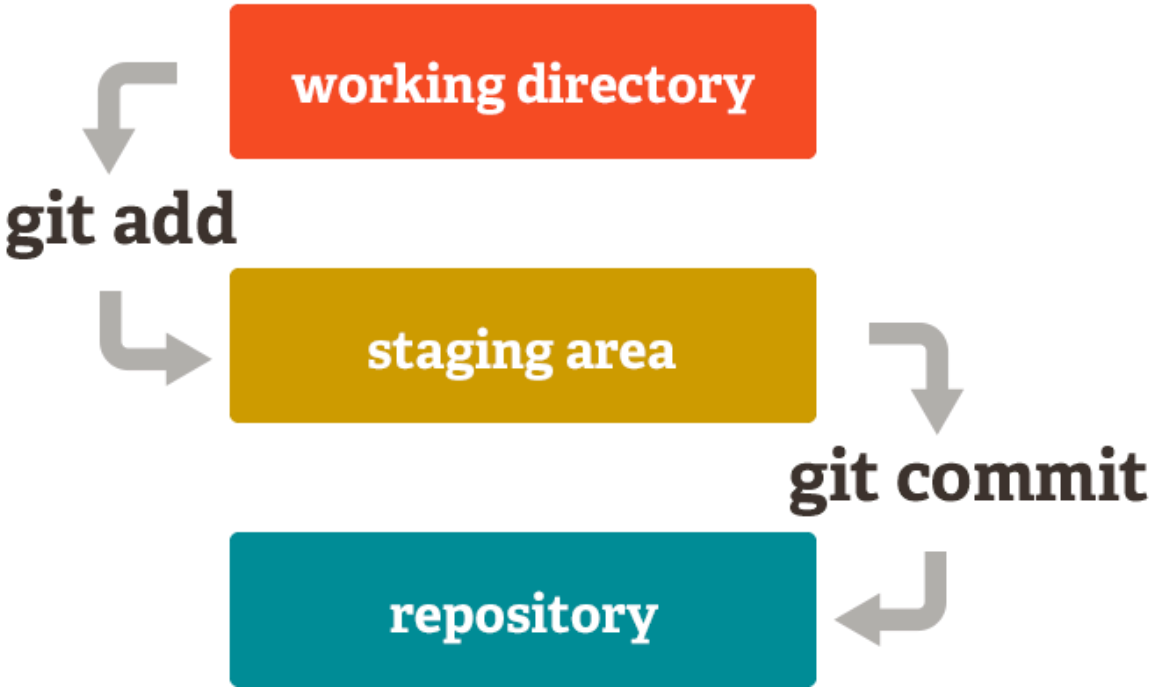
git config --local user.name "თქვენი სახელი"

git config --local user.email "თქვენი ელ. ფოსტა"კო

კონფიგურაციის შემდეგ ჩვენ გვჭირდება საქაღალდის ინიციალიზაცია იმისთვის რომ GIT-მა მისი კონტროლი დაიწყოს, ამისთვის გვჭირდება შემდეგი ბრძანება:

git init

GIT-ის ინიციალიზაციის შემდეგ საქარალდე სრულიად კონტროლირებადია GIT-ის მიერ, მასში ყველა ახლად დამატებული ან შექმნილი ფაილი შემდეგ სტრუქტურას დაექვემდებარება



GIT add ბრძანების სინტაქტი მორგებულია სხვადასხვა სამუშაო სიტუაციებს, განვიხილოთ რამოდენიმე ბრძანების ტიპი:

- კონკრეტული ფაილის დამატება - **git add file.txt**
- რამდენიმე ფაილის დამატება - **git add file1.txt file2.py file3.cpp**
- სამუშაო დირექტორიის ყველა ფაილის დამატება - **git add**
- ფაილების დამატება კონკრეტული დირექტორიიდან - **git add folder_name/**
- კონკრეტული გაფართოების მქონე ფაილების დამატება - **git add *.txt**

ფაილების სტატუსების სანახავად დაგვეხმარება ბრძანება:

git status

დამატებული ფაილების და საქაღალდეების ასლის (Snapshot)-ისას აუცილებლად უნდა დავუღტოთ კომენტარი, აუცილებლად უნდა მოიცავდეს მიმდინარე სტრუქტურის მოკლე აღწერას ან რიგ შემტხვევებში ცვლილების მოკლე აღწერას, სასურველია იყოს დაწერილი აწმყო დროში.

git commit -m "კომენტარი"

შეცვლილი ფაილების შემოწმება ძალიან მნიშვნელოვანია, რაც ნებისმიერ ეტაპზე

შეიძლება გამოგვადგეს git diff ბრძანება გამოიყენება, განვიხილოთ მისი ძირითადი ფუნქციონალი:

- სამუშაო დირექტორიისა და ბოლო commit-ის განსხვავება - **git diff**
- Staging შრისა და სამუშაო დირექტორიის განსხვავება - **git diff** --staged
- ორი კონკრეტული commit-ის განსხვავება - **git diff commit_id1 commit_id2**
- კონკრეტული ფაილის ცვლილებების შედარება ორ commit-ს შორის - **git diff commit_id1 commit_id2 -- file1.txt**

GIT-თან მუშაობისას, ხშირია არასწორი სინტაქსი და ბრძანებების არასწორი თანმიმდევრობა. ნებისმიერი ბრძანების სინტაქსის და აზრის ნახვა თვითონ GIT-ში შეიძლება:

- ზოგადი დახმარება - git help
- კონკრეტული ბრძანების ნახვა - git help “ბრძანება”