2025

ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

ვერსიონირება და უწყვეტი ინტეგრაცია

თემა მეორე: Git განშტოებების შექმნა და შერწყმის სტრატეგიები

Git-ის მოშორებული დირექტორიებთან დაკავშირება

Git-ის მოშორებულ დირექტორიებთან დაკავშირება საშუალებას იძლევა, რომ ლოკალური რეპოზიტორიის ცვლილებები სინქრონიზებული იყოს remote დირექტორიასთან, მაგალითად GitHub-თან. Remote დირექტორიებტან მუშაობა პროექტს დეცენტრალიზებულს ხდის (commit-ების ისტორია ინახება პროექტის ყველა წევრის კომპიუტერზე) და ამარტივებს გუნდურ მუშაობას.

GitHub-თან დაკავშირების ორი ძირითადი მეთოდი არსებობს: HTTPS და SSH. ორივე მეთოდი საშუალებას იძლევა ლოკალური და remote რეპოზიტორიებსი სინქრონიზაციის. განსხვავება მათი მუშაობის პრინციპებშია და რა ტიპის ავტორიზაცია გამოიყენება.

- HTTPS (Hypertext Transfer Protocol Secure) არის ვებ-ბრაუზერისთვის უსაფრთხო პროტოკოლი, რომელიც უზრუნველყოფს მონაცემების გადაცემას ინტერნეტში. GitHub-თან დაკავშირების დროს, HTTPS მეთოდი მოითხოვს თქვენს GitHub-ის ანგარიშზე რეგულარულ დამოწმებას.

- SSH (Secure Shell) არის უსაფრთხო დაკავშირების მეთოდი, რომელიც უზრუნველყოფს ავტორიზაციას მფლობელის ღია და დახურული გასაღებების საშუალებით, ნაცვლად პაროლის. GitHub-თან SSH-ით დაკავშირებისას, თქვენ უნდა შექმნათ და დაამატოთ SSH გასაღები თქვენს GitHub ანგარიშზე. ეს მეთოდი უზრუნველყოფს ავტორიზაციას და უფრო უსაფრთხო კავშირს.

თითოეულ მეთოდს თავისი მინუსები აქვს. HTTPS-ით დაკავშირებისას საჭიროა ინიციალიზაცია, remote დირექტორიის დამატება ბმულის მეშვეობით და დადასტურება ამ დაკავშირების. ასევე პერიოდულად მომხმარებლის სახელისა და პაროლის შეყვანა. SSH მეთოდი უფრო მარტივია და ნაკლებ ბრძანებებს საჭიროებს დაკავშირებისთვის, თუმცა სხვადასხვა კომპიუტერებიდან ერთ რეპოზიტორიაზე მუშაობისას საჭირო იქნება გასაღებების დუბლირება.

HTTPS-ით GitHub-თან დაკავშირება:

1. შექმენით github პროფილი და დირექტორია
2. შეინახეთ რეპოზიტორიის ბმული.
3. ლოკალური რეპოზიტორიის დაკავშირების ბრძანება:

   git remote add origin “github-ის ბმული”

4. პირველი push-ის ბრძანება:

   git push origin main

5. პირველი push-ის შემთხვევაში github ითხოვს მომხმარებლის სახელს და პაროლს.

SSH-ით GitHub-თან დაკავშირება:

1. SSH გასაღებების შესამოწმების ბრძანება:

   ls -al ~/.ssh

2. თუ გასაღები არ გაქვთ ან ახლის შექმნა გინდათ, გამოიყენეთ ბრძანება:

   ssh-keygen -t rsa -b 4096 -C "github-ზე მითითებული ელფოსტა"

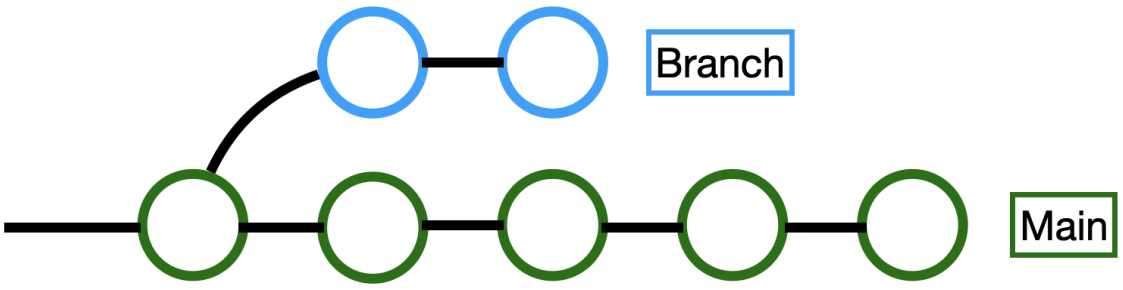
3. შეიქმნება 2 ფაილი id_rsa.pub-ღია გასაღები და id_rsa-დახურული გასაღები. ღია გასაღების შიგთავსი უნდა დაამატოთ github-ზე ssh-კონფიგურაციის გვერდზე.
4. ბრძანება დაკავშირებისთვის:

   git remote add origin "github-ის რეპოზიტორიის SSH ბმული"

github -თან დაკავშირება უფრო მარტივადაც შეიძლება **git clone** ბრძანების მეშვეობით. Git clone აერთიანებს : **git init** , **git remote add** და **git pull** ბრძანებებს.

ბრენჩინგი (Branching)

Git-ში ვერსიების შექმნა (branching) არის ერთი პროექტისგან რამდენიმე პარალელური განვითარების ხაზის, განშტოების (branch) შექმნა. თითოეული ბრენჩი წარმოადგენს დამოუკიდებელ სამუშაო გარემოს, სადაც შესაძლებელია სხვადასხვა ფუნქციონალის დანერგვა, რომელიც მოგვიანებით შეიძლება შეუერთდეს ძირითად ბრენჩს.



პრაქტიკაში main/master ბრენჩზე გუნდი არ მუშაობს. თითოეულ გუნდს აქვს გამოყოპილი ცალკეული დავალება რომელზე მუშაობაც ხდება ცალკეულ ბრენჩზე. ხშირია დიდი დავალების ბევრ პატარა დავალებად დაყოფა რის გამოც ბრენჩს შეიძლება ქვებრენჩები (sub branch) დაემატოს

ბრძანებები ბრენჩინგისთვის:

- ბრენჩის შექმნა - **git branch “ბრენჩის სახელი”**
- ბრენჩზე გადასვლა - **git checkout “ბრენჩის სახელი”**

  ზოგ ვესიაში გამოიყენება - **git switch “ბრენჩის სახელი”**

- ბრენჩის შექმნა და მასზე გადასვლა - **git checkout -b “ბრენჩის სახელი”**

  ზოგ ვერსიაში გამოიყენება **- git switch -c “ბრენჩის სახელი”**

- ბრენჩის წაშლა - **git branch -d “ბრენჩის სახელი”**

  ამ ბრძანებით იშლება ისეთი ბრენჩები რომლებიც უკვე დამატებულია (merged) ძირითად ბრენჩზე. დაუმერჯავი ბრენჩის წასაშლელად **-d** უნდა ჩანაცვლდეს **-D** ანუ:

  git branch -D “ბრენჩის სახელი”

- ლოკალური ბრენჩების სიის ნახვა - **git branch**
- ლოკალური და მოშორებული ბრენჩების სიიხ ნახვა - **git branch -a**

მერჯინგი (Merging)

მერჯინგი არის გიტის მექანიზმი რომელიც git-ის სხვადასხვა ბრენჩებს აერთიანებს. მერჯის იწერება იმ შტოში სადაც გვინდა ცვლილების გადმოტანა. Git-ს აქვს რამდენიმე merge-ის სტრატეგია:

ძირითადი სტრატეგიები:

- **Fast-forward Merge** - თუ feature-branch შეიცავს ცვლილებებს, ხოლო main-ზე (ან master-ზე) ამ პერიოდში ცვლილება არ განხორციელებულა, Git უბრალოდ გადაადგილებს main-ს `feature-branch`-ის commit-ებზე. ბრძანებები:

  git merge feature-branch ან **git merge feature-branch**

  Fast-forward merge ხდება მხოლოდ მაშინ, თუ main ტოტი არ შეიცავს ახალ commit-ებს მას შემდეგ, რაც feature-branch შეიქმნა.

- **Three-way Merge** - თუ mainc ტოტზე განხორციელდა ცვლილებები მას შემდეგ, რაც feature-branch შეიქმნა, მაშინ Git ვერ გამოიყენებს Fast-forward Merge-ს დაThree-way Merge-ს იყენებს. ბრძანებები:

  git merge feature-branch

  იგივე ბრძანებაა რაც fast-forward მერჯის დროს. Git-ი მერჯის ტიპს თვითონ განსაზღვრავს.

- **Octopus Merge** - გამოიყენება მაშინ, როდესაც ერთდროულად რამდენიმე ტოტის გაერთიანებაა საჭირო. ბრძანებები:

  git merge feature-branch1 feature-branch2 feature-branch3

  კონფლიქტების შემთხვევაში octopus merge ვერ შესრულდება.

- **Ours Merge** - გამოიყენება, როდესაც გვინდა, რომ მიმდინარე ტოტის ყველა ცვლილება შევინარჩუნოთ და სხვა ტოტის ცვლილებები დავაიგნოროთ. ბრძანებები:

  git merge -s ours feature-branch

  Feature branch-ში შეტანილი ცვლილებები არ შევა საბოლოო commit-ში.

- **Subtree Merge** - გამოიყენება, როდესაც საჭიროა სხვა რეპოზიტორიის (subproject) შერწყმა ძირითად პროექტთან ისე, რომ მისი ფაილები მოთავსდეს კონკრეტულ საქაღალდეში (subdirectory). ბრძანებები:

  git merge -s subtree feature-branch2