2025

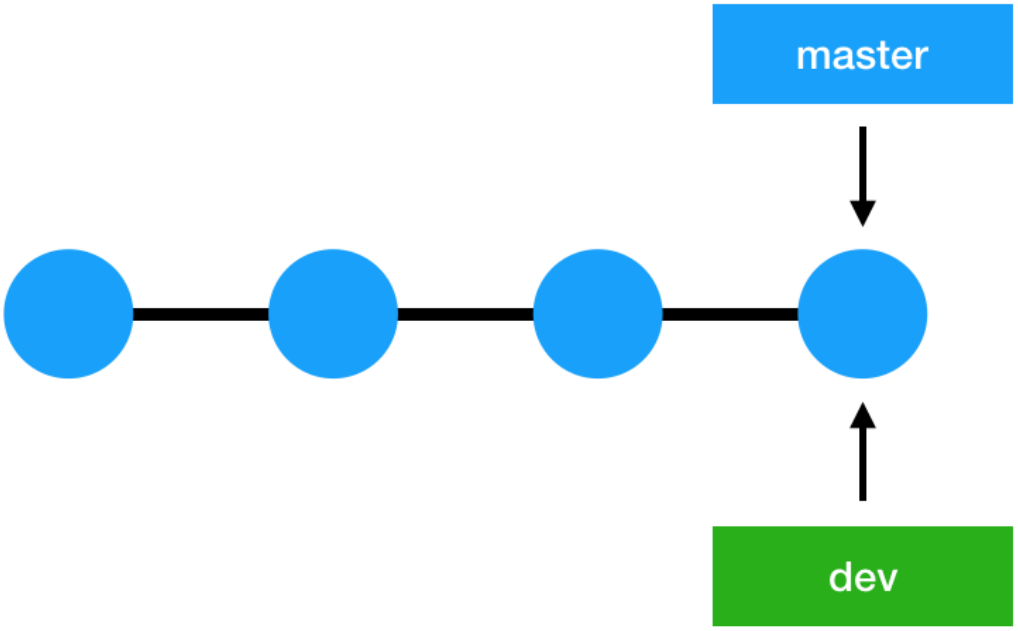
ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

ვერსიონირება და უწყვეტი ინტეგრაცია

თემა მეორე: Git რეპოზიტორიებთან მუშაობა, ტაგირება და ალიასები.

Main იგივე Master Branch-ი

GIT-თან მუშაობისას ყველი Commit-ი ინახავს ფაილების სტრუქტურის გარკვეულ მდგომარეობას/ვერსიას, მსგავსი ვერსია შეიძლება მრავლად გვქონდეს.



სურათში ნაჩვენებია პროექტის განვითარების მთავარი Main ან Master შტო. აღნიშნულ პროექტს გააჩნია მხოლოდ ერთი შტო 4 Commit-ით.

შტოს შეიძლება სხვა სახელიც ერქვას თუ მას თავიდანვე გავნსაზღვრავთ, შტოს სახელის შესამოწმებლად გამოიყენეთ ბრძანება:

Git branch

მიმდინარე, აქტუალური ვერსია git bash-ში HEAD-ით აღინიშნება. მის შესამოწმებლად გამოიყენეთ ბრძანება:

Git log

```
E:\Dev\Project>git log --pretty=short
commit 8363498a3c26cc50a0bc8d63e7ee3c0fb5a0548d (HEAD -> master)
Author: John Doe <john@doe.com>

    Added animation to the placeholder content

commit f03e53d3db9f5d382717629d965b04ff9370b9ad
Author: John Doe <john@doe.com>

    Added metadata for Open Graph
```

პროგრესის გაქუმება

ჩვენ უკვე ვიცით თუ როგორ უნდა დავადაროთ ორი ვერსია ერთმანეთს და ვნახოთ მათ შორის სხვაობა, თუმცა ეგ ფუნქციონალი სამუშაო გარემოში არ არის საკმარისი, ამის გარდა შეიძლება დაგვჭირდეს ძველ ვერსიაზე დაბრუნება.

ძველ ვერსიაზე დასაბრუნებლად დაგვჭირდება ბრძანება Checkout რომლის HEAD პარამეტრი განსაზღვრავს თუ რომელ ვერსიაზე გვინდა დაბრუნება. განსაზღვრა ანუ ვერსიების ათვლა ხდება მიმდინარე ვერსიიდან. ვერსიებს შორის ბიჯის განმსაზღვრელია “^” სიმბოლო, რამდენი სიმბოლოც თან მოსდევს HEAD-ს მაგდენი ბიჯით წინა ვერსიას ვუბრუნდებით. იმ შემთხვევაში თუ პროექტს ძალიან ბევრი ვერსია აქვს და გვჭირდება მაგალითად 27 ბიჯით წინა ვერსია დაბრუნება, 27 ჯერ “^” სიმბოლოს გამეორების ნაცვლად შეგვიძლია HEAD-ს მოვაყოლოთ “~27” რაც იმავე პრინციპით იმუშავებს. გვაქვს ასევე მე-3 ვარიანტიც, რომ მივუთითოთ commit-ის id.

1 ბიჯით	HEAD^	HEAD~1	ბოლოდან მე-2 commit-ის id
2 ბიჯით	HEAD^^	HEAD~2	ბოლოდან მე-3 commit-ის id
5 ბიჯით	HEAD^^^^^	HEAD~5	ბოლოდან მე-6 commit-ის id
10 ბიჯით	HEAD^^^^^^^^^^	HEAD~10	ბოლოდან მე-11 commit-ის id

ბრძანების მაგალითები:

- **git checkout HEAD^**
- **git checkout HEAD~4**
- **git checkout a9f8b3d6c2e93b0c045f9d7e812fc7a16c2354b7**

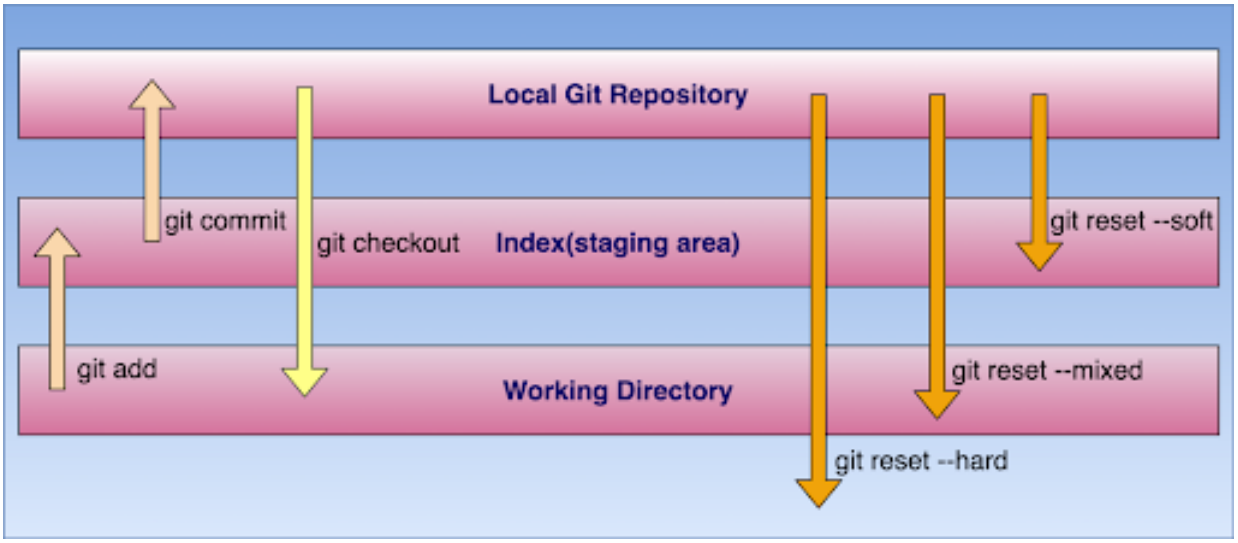
ზემოთ მოყვანილი ბრძანებების შემდეგ შემდეგ საქაღალდის სტრუქტურა შეიცვლება ვერსიის შესაბამისად, თუმცა checkout ბრძანება commit-ების ისტორიას არ შლის, მხოლოდ აბრუნებს წინა ვერსიაზე. იმისათვის რომ დაბრუნდეს ბოლო ვერსიაზე პარამეტრად უნდა გადავცეთ branch-ის სახელი.

Git checkout master

იმ შემთხვევაში თუ წინა ვერსიაში გამოგვეპარა შეცდომა ან რაიმე სხვა მიზეზით გვინდა წინა ვერსიას მივუბრუნდეთ ისე რომ გავაუქმოთ მთელი ის პროგრესი და ყველა ის commit-ი რაც მის შემდეგ მოხდა უნდა გამოვიყენოთ reset ბრძანება.

Reset ბრძანება რამოდენიმე ტიპის არის:

- Reset –soft
- Reset –mixed
- Reset –hard



Git reset –soft ბრძანების მეშვეობით ყველა ცვლილებას ვერსიებს შორის ვათავსებთ staging სივრცეში

Git reset –mixed ბრძანების მეშვეობით ყველა ცვლილებას ვერსიებს შორის ვათავსებთ Working directory სივრცეში. Reset ბრძანება ნაგულისხმევი მნიშვნელობით (პარამეტრს თუ არ გადავცემთ) იგივენაირად მუშაობს

Git reset –hard ბრძანების მეშვეობით ყველა ცვლილებას ვერსიებს შორის ვშლით. –hard პარამეტრის გამოყენება დიდ სიგრთხილეს საჭიროებს ლოკალურად ხოლო მოშორებულ (remote) გარემოზე რომელშიც სხვა პირებიც მუშაობენ მისი გამოყენება არ არის რეკომენდირებული, პოტენციური კონფლიქტების და ინფორმაციის დაკარგვის თავიდან ასაცილებლად

ტაგირება Tags

ტეგები Git-ში გამოიყენება კონკრეტული კომიტების აღსანიშნავად, განსაკუთრებით მაშინ, როდესაც პროექტი მიაღწევს მნიშვნელოვან ეტაპს, მაგალითად თუ მორჩა პროექტის რომელიმე ვერსიაზე მუშაობა ამის აღსანიშნად შეგვიძლია გამოვიყენოთ ტაგირება.

მიმოვიხილოთ ტაგებთან სამუშაოდ ძირითადი ბრძანებები:

- ტაგის შექმნა - **git tag “ტაგის სახელი”**
- ტაგების სიის ნახვა - **git tag**
- ტაგის წაშლა - **git tag -d “ტაგის სახელი”**
- ანოტირებული ტაგის შექმნა (ტაგის რომელიც მეტ ინფორმაციას შეიცავს commit-ის შესახებ) - **git tag -a “ტაგის სახელი” -m "ტეგის აღწერა"**

ხშირია შემთხვევები როდესაც ტაგს არასწორ commit-ს ვამაგრებთ, ამ შემტხვევაში გამოიყენება ბრძანება

- **git tag -f “ტაგის სახელი” ახალი commit-ის ჰეში**

ნაგულისხმევი მნიშვნელობით ტაგები მიმდინარე commit-ს ედება, თუ გვინდა რომელიმე ძველ commit-ს დავადოთ ტაგი მაშინ ბოლოში უნდა მივუწეროთ commit-ის ჰეში

- **git tag v1.0 “commit-ის ჰეში”**

ნაადრევად შევეხოთ მოშორებულ (remote) დირექტორიებს, ტაგების ცვლილებები remote დირექტორიის შემთხვევაში სხვანაირ სინტაქსს მოითხოვს. მაგალითად ტაგის წაშლა:

- **Local - git tag -d “ტაგის სახელი”**
- **Remote - git push --delete origin “ტაგის სახელი”**

ამგვარად ჩანს რომ remote გარემოში ტაგებთან მუშაობა მეტ ყურადრებას მოითხოვს.

ალიასები Alias

ალიასები გამოიყენება ბრძანებების შესამოკლებლად და git-ის სინტაქსის ჩვენსთავზე მორგებას. ალიასების მეშვეობით შეგვიძლია შევცვალოთ მაგალითად ბრძანება status ბრძანება st-ზე და ყოველ ჯერზე როცა სტატუსის დაწერა დაგვჭირდება საშუალება გვექნება რომ დავწეროთ ან status ან st.

ალიასის შესაქმნელად უნდა დავწეროთ ბრძანება- **git config --global alias.”ალიასი რომელიც გვინდა რომ გამოვიყენოთ” “ბრძანება რომლის ნაცვლადაც გვინდა რომ გამოვიყენოთ ალიასი"**

მაგალითი - **git config --global alias.st status**

ალიასების გამოყენება თქვენი არჩევანია, თუმცა არსებობს რამოდენიმე მოსახერხებელი ალიასი:

- ბოლო ქომითის გაუქმების ალიასი
  git config --global alias.undo "reset --soft HEAD~1"

- ბოლო კომიტის სრულად გაუქმება
  git config --global alias.hardundo "reset --hard HEAD~1"

- ბოლო კომიტის დეტალები
  git config --global alias.last "log -1 HEAD"