

ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

ვერსიონირება და უნწყვეტი ინტეგრაცია

თემა მეოთხე: **Git Merge** კონფლიქტები,
მიზეზები და მათი ეფექტური გადაჭრა

Git Merge კონფლიქტები

Git Merge კონფლიქტი ხდება მაშინ, როდესაც ორი სხვადასხვა ცვლილება ერთსა და იმავე ფაილში, იმავე ხაზებში ხდება და Git ვერ წყვეტს, რომელი ვერსიაა სწორი. კონფლიქტი შეიძლება წარმოიქმნას თუ:

- ორი დეველოპერი მუშაობს ერთსა და იმავე ფაილზე და სხვადასხვა ვერსიას ამატებს.
- ერთი დეველოპერი ფაილს შლის, ხოლო მეორე რედაქტირებას ახდენს.
- ერთი ბრენჩი ცვლის ფაილის სახელს, ხოლო მეორე ბრენჩი იმავე ფაილში ცვლილებებს აკეთებს.

თუ კონფლიქტი წარმოიქმნა, Git ამას გატყობინებთ და გამოაჩენს ფაილებს, სადაც კონფლიქტია, ისევე როგორც სურათში.

```
zean_7@BlueBytes:~/testing_merge_conflict$ git checkout master
Switched to branch 'master'
zean_7@BlueBytes:~/testing_merge_conflict$ echo "Append this text to initial commit" >> test_file.txt
zean_7@BlueBytes:~/testing_merge_conflict$ git add .
zean_7@BlueBytes:~/testing_merge_conflict$ git commit -m "appended some text to initial commit"
[master 5f30568] appended some text to initial commit
 1 file changed, 1 insertion(+)
zean_7@BlueBytes:~/testing_merge_conflict$ git merge new_branch_for_merge_conflict
Auto-merging test_file.txt
CONFLICT (content): Merge conflict in test_file.txt
Automatic merge failed; fix conflicts and then commit the result.
zean_7@BlueBytes:~/testing_merge_conflict$
```

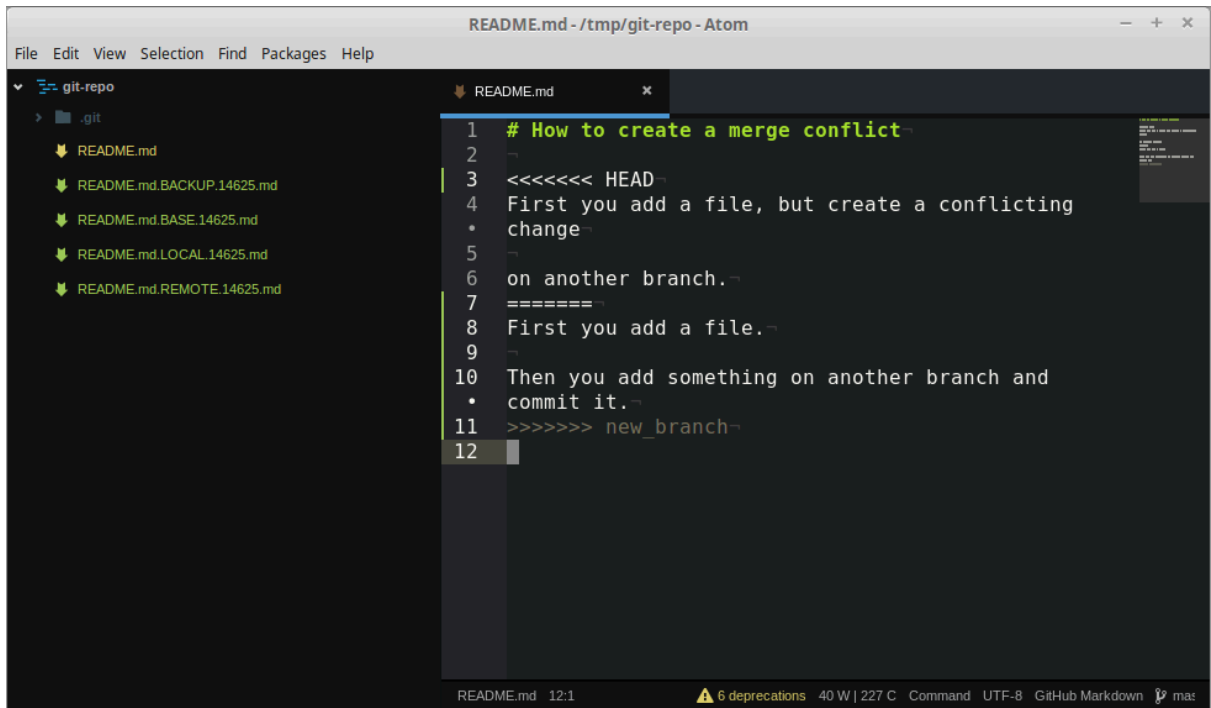
კონფლიქტის წარმოქმნისას ფაილი ხელით უნდა ჩასწორდეს.

Git status

შემოწმებისას ფაილს ექნება სტატუსი **both modified** მანამ სანამ ჩვენ არ ჩავასწორებთ ფაილს ან არ გავაქუმებთ ქომითს. ქომითის გაუქმება ხდება ბრძანებით:

Git merge --abort

კონფლიქტური ფაილის კონფლიქტის მიზეზი, გადაკვეთილი კოდის ნაწილი გამოყოფილი იქნება მეტობისა და ნაკლებობის ნიშნებით. მიმდინარე ვერსიის შესაბამისი კოდის ნაწილი HEAD-ით იქნება აღნიშნული. მეორე ვერსიისგან გამოყოფილი იქნება ტოლობებით, კოდის მეორე ვერსია კი ტოლობოს შემდეგ იქნება მოცემული რომლის შემდეგაც იქნება მითითებული ბრენჩის სახელი.



```
1 # How to create a merge conflict
2
3 <<<<<<< HEAD
4 First you add a file, but create a conflicting
5 • change
6 on another branch.
7 =====
8 First you add a file.
9
10 Then you add something on another branch and
11 • commit it.
12 >>>>>>> new_branch
```

ფაილის შეცვლის შემდეგ საჭიროა მისი დამატება და დაქომითება.

არსებობს რამოდენიმე აპრობირებული მეთოდი რომლითაც შეიძლება მერჯ კონფლიქტის თავიდან აცილება ან კონფლიქტების რაოდენობის მინიმუმამდე დაყვანა.

- პერიოდული Pull-ები
- კონვენციების დაცვა - კოდის წერის საერთო სტანდარტით ხელმძღვანელობა
- ფაილების დასუფთავება / ზედმეტი ფაილების წაშლა
- კონფლიქტების ლოგიკური გადაჭრა

Stash და ბრენჩინგი

Git stash-ი მჭიდროდაა ბრენჩინგთან დაკავშირებული. როგორც ვიცით პოექტის

მსვლელობისას ძირითადი მუშაობა და ცვლილებები სწორედ ბრენჩებზე ხდება. რიგ სისტემებში შეიძლება შეზღუდული იყოს ბრენჩების შეცვლა თუ ბრენჩში გვაქვს დაუქომითებული ცვლილებები. ცვლილებების დაქომითების შემდეგ ბრენჩების შეცვლა ისევ შესაძლებელი იქნება თუმცა არსებობს გზა ბრენჩის შეცვლის ზედმეტი ქომითის დამატების გარეშეც. ეს არის **stash**-ი რომელიც ინახავს **Tracked** და **Staged** ფაილებს, ხოლო **Untracked** და **Ignored** ფაილებს ნაგულისხმევი მნიშვნელოვით ვერ ინახავს. **stash** -ში ფაილების გადასატანად გამოიყენება ბრძანება:

- **Git stash** - ნაგულისხმევი მნიშვნელობით მოქმედებს **Tracked** და **Staged** ფაილებზე
- **git stash -u** - მოქმედებს **Untracked** ფაილებზე
- **git stash -a** - მოქმედებს **Ignored** ფაილებზე

დაუქომითებული ცვლილებები გაქრება და შეინახება **Git**-ის ლოკალურ რეპოზიტორიაში. **Stash** ბრძანების ძირითადი ნაირსახეობებია:

- **git stash save "კომენტარი"** - ცვლილებების კომენტართან შენახვა
- **git stash list** - **stash**-ის სია
- **git stash show** - შემოკლებულად გვიჩვენებს რა ცვლილებები გვაქვს შენახული
- **git stash show -p** - სრულად გვიჩვენებს რა ცვლილებები გვაქვს შენახული
- **Git stash apply** - აღადგენს შენახულ ცვლილებას თუმცა **stash**-ი მაინც დარჩება
- **Git stash pop** - აღადგენს ცვლილებებს და ნაშლის **stash**-იდან
- **git stash drop "stash-ის სახელი"** - შლის კონკრეტულ **stash**-ს
- **git stash clear** - შლის ყველა **stash**-ს