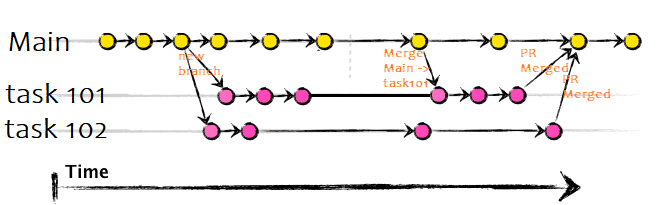


U dev-firmama se obično kreiraju feature grane iz **develop**, te se na kraju taska radi merge u **develop** (kao na gornjoj slici).

# RS1 – branchovi

U ovom predmetu nećemo koristiti **develop** granu, već samo master granu. Stoga ćemo imati:

1. **master** grana – stabilna verzija projekta koja sadrži samo završene i testirane promjene.
2. **grana za task/feature** – kreira se za svaki novi zadatak (task) iz **master** brancha, sadrži promjene specifične za taj zadatak.

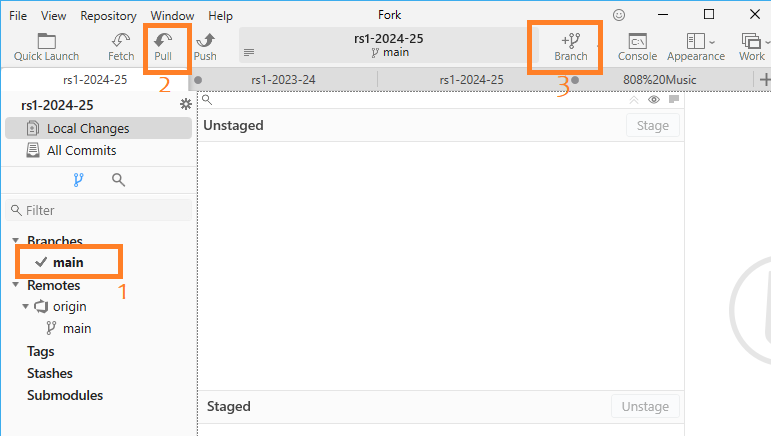


## Kako kreirati novi branch za rad na tasku?

Prilikom rada na projektu, obično se koristi **grana** (branch) za implementaciju specifičnih zadataka (taskova) kako bi se osiguralo da glavna grana (main branch) ostane stabilna. Sljedeći koraci objašnjavaju kako pravilno kreirati novi branch za task.

A screenshot of a computer

Description automatically generated

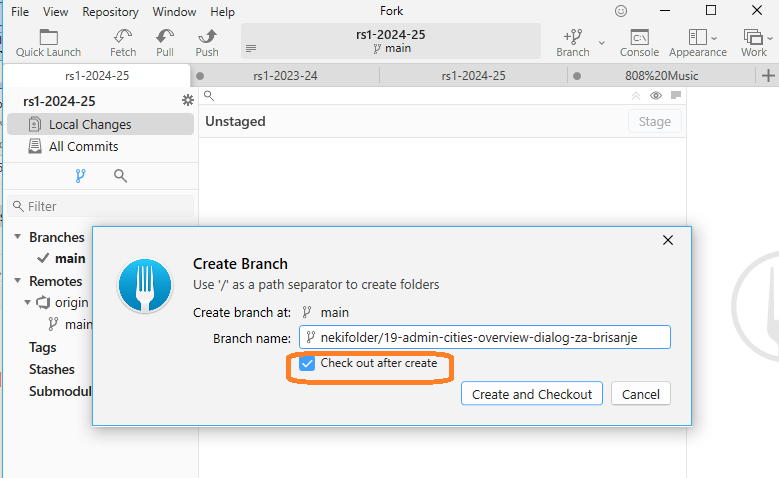


**Koraci:**

1. **Preći na main branch:**
   * Prvo, potrebno je prebaciti se na **main** branch (glavnu granu) jer želimo osigurati da kreiramo novi branch iz ažurirane verzije glavne grane.
   * U alatu (kao što je Fork), provjerite je li selektovan **main** branch.
2. **Pull zadnje verzije (ažuriranje):**
   * Izvršite **pull** (preuzimanje zadnjih promjena sa udaljenog repozitorija) kako biste osigurali da radite sa najnovijom verzijom koda.
   * Ovaj korak je važan kako biste izbjegli konflikte kada kasnije šaljete (push) promjene na udaljeni repozitorij.
3. **Kreirati novi branch:**
   * Kliknite na opciju **Branch** ili koristite komandu za kreiranje nove grane.
   * Preporuka je da naziv grane uključuje:
     + **Broj taska** – kako bi grana bila povezana sa zadatkom.
     + **Opis taska** – kratak, jasan opis (npr. 19-admin-cities-overview za Task 19).
   * Ovaj format olakšava identifikaciju grana unutar tima.

**Obrazloženje:**

* **Zašto koristimo grane?**
  + Grane omogućavaju paralelan rad na različitim zadacima bez ometanja glavne grane.
  + Svaka grana može se testirati i integrirati u glavnu granu tek nakon što zadatak bude završen i kod validiran.
* **Zašto ažurirati main prije kreiranja grane?**
  + Time osiguravamo da počinjemo rad sa najnovijom verzijom projekta, što minimizira potencijalne konflikte.



A screenshot of a computer

Description automatically generated

Nakon lokalnih commitova možete uraditi **push** na server i nastaviti s kreiranjem **pull requesta (PR)** na **Azure DevOps** platformi.

Ako postoji konflikt prilikom merge-a vašeg feature brancha, greška će biti prikazana na PR-u u Azure DevOps (web sučelju). U tom slučaju, konflikte morate riješiti lokalno tako što ćete najnovije izmjene s master grane **mergati** u vaš lokalni feature branch te riješiti konflikte fajl po fajl.

## Kada raditi merge mastera u task/feature branch:

1. **Reaktivno (nakon što se konflikt pojavi na PR-u):**
   * Merge kreirate tek kada se jave konflikti prilikom otvaranja ili provjere PR-a.
2. **Proaktivno (tokom razvoja):**
   * Merge s mastera u feature branch radite u bilo kojem trenutku dok radite na feature branši, kako biste osigurali da je vaš kod ažuriran s promjenama iz master grane.

Preporučuje se merganje mastera **prije otvaranja PR-a,** ako bi riješili konflikte ako ih ima.

## Koraci za lokalno merganje najnovijih izmjena s mastera:A screenshot of a computer Description automatically generated

1. **Kliknite na Fetch:**
   * Osvježite najnovije promjene s udaljenog repozitorija (origin).
2. **Merge bez prelaska na master:**
   * Ako želite ostati na vašem feature branchu, možete direktno mergati promjene iz remote/origin/main u trenutni branch.
   * Koristite **desni klik** na origin/main i odaberite opciju:
3. **Rješavanje konflikata (ako ima)**
   * Ako se konflikti pojave, Git će prikazati listu fajlova s konfliktima.
   * Otvorite svaki fajl s konfliktom i ručno odaberite koje promjene zadržati.

A screenshot of a computer

Description automatically generated

Detalji su opisani ispod

1. **Commit nakon rješavanja konflikata:**
   * Nakon što riješite sve konflikte, commitajte rješenja:
2. **Push promjena:**
   * Kada su svi konflikti riješeni i promjene testirane, pošaljite ih na udaljeni repozitorij:

---

## Rješavanje konflikta u svakom fajlu pojedinačno

Kada se započne merge sa konliktima, onda morate završiti proces do kraja ili prekinuti sa Abort.

Za svaki fajl treba odabrati jednu od 3 opcije

1. Koristiti dolazeću verziju fajla (odbaci **moje** izmjene)

A screenshot of a computer

Description automatically generated

1. Koristi moju verziju fajla (odbaci **tuđe** izmjene)

A screenshot of a computer

Description automatically generated

1. Merge u novom prozoru / editoru

A screenshot of a computer

Description automatically generated

Ako se koristi Git za merge i dođe do konflikta, Git automatski dodaje specifične oznake u fajlove kako bi označio gdje su konflikti. Evo kako Git to radi i kako je prikazano u UI editorima

**Kako Git označava konflikte:**

Kada Git ne može automatski spojiti promjene, u konfliktne fajlove ubacuje ove oznake koje su vidljive u običnom tekst editoru:

|  |
| --- |
| **<<<<<<< HEAD**  vaš code (trenutne promjene na lokalnoj branši)  **=======**  njihov code (promjene iz druge branše, npr. `origin/main`)  **>>>>>>> origin/main** |

**Kako to izgleda u UI editorima:**

Većina modernih tekstualnih editora i IDE-a nudi vizualni prikaz za rješavanje konflikata, koji uključuje tri glavna prozora:

1. **Vaš kod**:
   * Promjene koje ste napravili u trenutnoj branši.
2. **Njihov kod**:
   * Promjene iz druge branše (origin/main).
3. **Spojeni kod**:
   * Prostor gdje trebate odlučiti šta želite zadržati, odbaciti ili kombinovati.

Git omogućava fleksibilnost kod rješavanja konflikata, čak i kada započnete proces merge-a u jednom alatu, a nastavite rješavanje u drugom. Ovo je korisno u projektima gdje različite datoteke zahtijevaju specifične alate za pregled i uređivanje.

1. **Fleksibilnost Git-a:**
   * Kada započnete merge u jednom Git alatu (npr. Fork), konflikti se jednostavno mogu riješiti u bilo kojem drugom alatu ili IDE-u (npr. WebStorm za frontend datoteke ili Visual Studio za C# kod).
2. **Nezavisno rješavanje fajlova:**
   * Git označava konflikte za svaki fajl pojedinačno. Možete riješiti konflikte u različitim alatima dok se svi fajlovi ne riješe.

**Evo kako isti fajl izgleda u Forku**A screenshot of a computer

Description automatically generated

**Evo kako isti fajl izgleda u VS-u**A screenshot of a computer

Description automatically generated

**Evo kako isti fajl izgleda u Webstormu-u**A screenshot of a computer

Description automatically generated

## Kreiranje Pull Request (PR)

1. Nakon završetka rada na feature grani, otvorite opciju za kreiranje PR-a.

Obično nakon pusha coda na server, na azure prozoru za prikaz branchova i PR-ova se pojavi poruka za kreiranje novog PR-a za tek commitovan code.

1. Popunite polja:
   * **Title:** Naziv PR-a – automatski se popunjava iz naziva brancha.
   * **Description:** Dodatne napomene
2. Povežite PR s odgovarajućim **Work Item-om** (Taskovima).
3. Dodajte **reviewere** (osobe koje trebaju pregledati kod).
4. Kliknite **Create** da kreirate PR.

A screenshot of a computer

Description automatically generated

**Ako ima konflikata**A screenshot of a computer

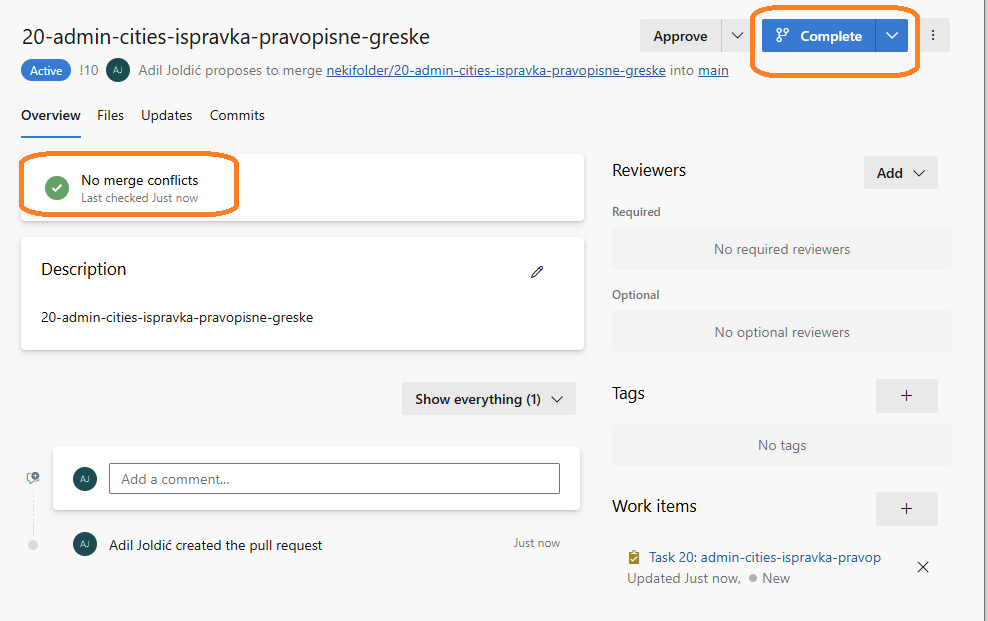
Description automatically generated

Ako ima konflikt, treba proći korake za lokalni merge (opisane gore).

Nakon ispravke i pusha, automatski će se prikazati novo stanje u ovom PR prozoru (azure devops web sučelje).

Znači, dok je otvoren PR moguće raditi commit i push novog code-a.

**Ako nema konflikata**



Naravno prije klika na „Complete“, code treba biti pregled. Detaljni opis je ispod.

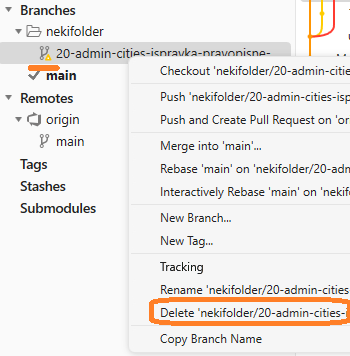
A screenshot of a computer

Description automatically generated

Brisanje feature grane na serveru nakon što je merge završen, kako biste održali čistoću repozitorija.

Nakon brisanja grane na serveru, nakon što Fork prepozna izmjene (klikom na fetch ili pull) prikazuje se žuta warning ikonica.

Moguće je lokalno obrisati branch.



## Code Review

Reviewer započinje pregled code-a te dodaje eventualne komentare.

**Proces pregleda koda od strane reviewera:**

1. **Označavanje pregledanih fajlova:**
   * Reviewer tokom pregleda može označiti pregledane fajlove pomoću checkbox opcije pored svakog fajla.
   * Ova oznaka pokazuje da je reviewer pregledao određeni fajl i ne mora ga ponovo provjeravati ako nije došlo do novih izmjena.

A screenshot of a computer

Description automatically generated

1. **Dodavanje komentara:**
   * Tokom pregleda, reviewer može dodati komentare za svaki fajl:
     + Komentari mogu sadržavati prijedloge za izmjene, pitanja ili objašnjenja.
     + Komentari se automatski vežu za određene linije koda u fajlu.

A screenshot of a computer

Description automatically generated

1. **Reakcija na nove izmjene:**
   * Ako autor Pull Requesta napravi **novi commit i push**, Git automatski uklanja checkbox oznake sa fajlova koji su izmijenjeni.
   * Ovo signalizira revieweru da treba ponovo pregledati te fajlove kako bi potvrdio izmjene.
2. **Komunikacija:**
   * Komentari omogućavaju dvosmjernu komunikaciju između autora PR-a i reviewera. Autor može odgovoriti na komentare ili izvršiti potrebne izmjene.

**Korisne napomene za reviewere:**

* **Koncentrisani pregled:** Fokusirajte se na promjene koje su u sklopu zadatka (PR-a).
* **Pratite historiju:** Pregledajte prethodne komentare kako biste razumjeli kontekst trenutnih izmjena.
* **Jasni komentari:** Jasno obrazložite svoje sugestije kako bi autor lako razumio zahtjeve.

**Automatizacija procesa:**

Ako koristite integracije (npr. sa CI/CD alatima), možete dodati pravila koja onemogućavaju završavanje PR-a dok svi fajlovi nisu pregledani i označeni kao prihvaćeni od strane reviewera.