



Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Osnove rada s Git-om

3. listopada 2025.

Sastavio: Zvonimir Bujanović



Kratki uvod u Git

- Git je sustav za upravljanje izvornim kodom kojeg je razvio Linus Torvalds (engl. git = “neugodna osoba”).
- Osnovna namjena: razvoj kompleksnog softvera od strane više *developer*a, efikasno upravljanje verzijama koda.
- Dostupan za Windows, Linux, OS X.
- Brz, pouzdan i popularan; *open-source*.
- Skinuti ga možete sa službene stranice (www.git-scm.com).
- Instaliran je u praktikumima pod Linuxom.
- Korisna literatura:
 - “Git Tutorial”, <https://www.atlassian.com/git/tutorials/>
 - T. Krajina: “Uvod u Git”, <http://tkrajina.github.io/uvod-u-git/git.pdf>
- Javno dostupni popularni git-serveri:
 - GitHub – <https://github.com/>
 - Bitbucket – <https://bitbucket.org/>
 - GitLab – <https://gitlab.com/>
 - Koristit ćemo GitHub – iako svi gore navedeni nude besplatne privatne repozitorije.

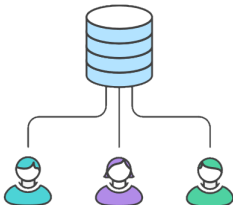
- Radimo samo najjednostavnije koncepte za operativni rad, bez ulaženja u detalje.
- **Repozitorij**
 - Skup datoteka koji su u sustavu verzioniranja i koji zajedno čini jedan (softverski) projekt.
 - **Globalni repozitorij** – nalazi se na git-serveru (Bitbucket).
 - **Lokalni repozitorij** – potpuna kopija globalnog koju svaki developer ima na svom računalu.
- Nakon instalacije git-a, potrebno je napraviti identifikaciju korisnika.
 - `git config -global user.name "Korisnik"`
 - `git config -global user.email "korisnik@adresa.hr"`
- Za svaku git-naredbu možemo dobiti pomoć sa
 - `git help ime_naredbe`

Stvaranje novog globalnog repozitorija

- Za svaki novi projekt, jedan od developera treba stvoriti novi globalni repozitorij.
- GitHub:
 - Privatni repozitorij – projekt je vidljiv samo developerima na projektu.
 - U besplatnoj varijanti možemo imati po volji mnogo developera na privatnom repozitoriju.
- Svima koji će sudjelovati na projektu možemo dati pristup (Read/Write/Admin).
 - Settings -> Collaborators -> Add people.
- Nakon toga, svaki developer na svoje računalo treba klonirati globalni repozitorij i tako dobiti svoj lokalni repozitorij.

Stvaranje novog lokalnog repozitorija – kloniranje globalnog

- 1 Pozicioniramo se u direktorij u kojem želimo napraviti novi lokalni repozitorij (recimo, `/home/user/repos`).
- 2 `git clone adresa_globalnog_repozitorija`
 - Stvara potpunu kopiju globalnog repozitorija na lokalnom računalu ~→ lokalni repozitorij.
 - GitHub: ako je instaliran `github-cli`, onda:
`gh repo clone ime_repozitorija`
Prvi puta na novom računalu treba izvršiti `gh auth login`.
 - GitHub: u protivnom, potrebno je generirati *personal access token* (Settings->Developer settings->Personal access tokens->Tokens (classic)).



Dodavanje datoteka u sustav verzioniranja

- 1 Stvorimo novu datoteku unutar tekućeg direktorija (recimo, `/home/user/repos/ime_datoteke`).
- 2 `git add ime_datoteke`
 - Uvodi datoteku *ime_datoteke* u sustav verzioniranja.
 - Nije nužno da sve datoteke u softverskom projektu budu u sustavu verzioniranja – tipično, dodajemo samo datoteke u kojima se nalazi kod ili neka konfiguracija projekta.
 - Obično ne dodajemo izvršne ili binarne datoteke (.exe i slične).
- 3 `git commit -a -m "Opis izmjene"`
 - Sprema nastale promjene **svih** datoteka u **lokalni** repozitorij.
 - Commit se ne može spremiti bez opisa.
 - DZ: Proučite čemu služi opcija `-a`.
- 4 `git push`
 - Prosljeđuje napravljene promjene iz lokalne kopije repozitorija u **globalni** repozitorij.
 - Prije ove naredbe je potrebno spremiti promjene u lokalni repozitorij (`git add`, `git commit`).

- `git status`
 - Daje popis datoteka u repozitoriju i info je li na njima u međuvremenu napravljena neka izmjena.
 - Kažemo da je datoteka promijenjena ako je tek nastala, izbrisana ili joj se promijenio sadržaj (nešto je dodano/izbrisano).
- `git diff ime_datoteke`
 - Daje popis nastalih izmjena u datoteci *ime_datoteke*.
- `git log`
 - Popis svih commitova s imenom korisnika koji je napravio commit, datumom i opisom.

Dohvaćanje promjena iz globalnog repozitorija

- `git pull -rebase origin master`
 - Dohvaća promjene iz globalnog repozitorija koje su u međuvremenu napravili drugi developeri.

Primjer tijeka rada s git-om

- ➊ Pero napravi globalni repozitorij, omogući pristup Ani.
 - ➋ Ana klonira globalni repozitorij.
 - ➌ Pero klonira globalni repozitorij.
 - ➍ Ana stvori novu datoteku `dat.txt`, napravi `add+commit+push`.
 - ➎ Pero napravi `pull`.
 - ➏ Pero modificira `dat.txt`, napravi `commit+push`.
 - ➐ Ana napravi `pull`.
 - ➑ Ana modificira `dat.txt`, stvori novu `nova.txt`, napravi `add+commit+push`
 - ➒ Pero napravi `pull`
 - ➓ ...
- Prije svakog `push`-a, treba dohvatiti izmjene u globalnom repozitoriju pomoću `pull`!

Zadatak 1

- 1 Napravite korisnički račun na GitHub-u.
- 2 Na GitHubu preko web-a stvorite novi privatni git repozitorij `"rwa-gittutorial-username"`.
- 3 Napravite klona ovog globalnog repozitorija na lokalnom računalu.
- 4 Dodajte rješenje Zadatka 1 iz Predavanja 02 (`forma.html`) u repozitorij.
- 5 Dodajte opis projekta u datoteci `readme.md` u repozitorij.
- 6 Napravite `commit + push` na GitHub.
- 7 DZ: Stvorite klona ovog repozitorija i na svom računalu doma.

Scenario I:

- ➊ Ana izmijeni `dat.txt`, napravi `commit+push`.
- ➋ Pero ne napravi prvo `pull`, nego izmijeni `dat.txt` i pokuša napraviti `commit+push`.
- ➌ Git traži da se prvo napravi `pull` prije `commit+push`.

Scenario I:

- 1 Ana izmijeni `dat.txt`, napravi `commit+push`.
- 2 Pero ne napravi prvo `pull`, nego izmjeni `dat.txt` i pokuša napraviti `commit+push`.
- 3 Git traži da se prvo napravi `pull` prije `commit+push`.

Scenario II:

- 1 Ana izmijeni `dat.txt`, napravi `commit+push`.
- 2 Pero ne napravi prvo `pull`, nego izmjeni `dat.txt`.
- 3 Pero napravi `pull`.
- 4 Konflikt u Perinom radnom direktoriju.

- Ako je došlo do konflikta, ponekad ga git može sam automatski razriješiti.
 - Npr. Ana je promijenila 3. liniju u `dat.txt`, a Pero je dodao novu liniju na kraj $\rightsquigarrow$ git uvaži i jednu i drugu modifikaciju.
- Konflikte nastale zbog većih modifikacija iste datoteke od strane oba developera treba riješiti manualno:
 - 1 Došlo je do konflikta u Perinom lokalnom direktoriju, na datoteci `dat.txt`.
 - 2 Prilikom `git pull`, git je spojio i Perinu i Aninu datoteku `dat.txt` u jednu.
 - 3 Pero analizira i editira `dat.txt` i stvara konačnu verziju.
 - 4 Pero: `git add dat.txt` (ponovno!)
 - 5 Pero: `git rebase -continue`
 - 6 Pero: `git push`
 - 7 Konflikt je riješen, Perina konačna verzija ide u glob. repozitorij.

Opisali smo tzv. *centralized workflow*.

<https://www.atlassian.com/git/tutorials/comparing-workflows/centralized-workflow>

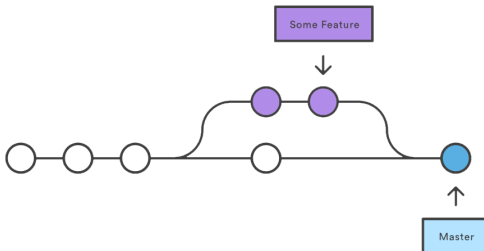
Taj pristup ima sljedeće probleme:

- ➊ Ako developeri *push-aju* svaku i najmanju promjenu, onda je čitavi projekt spremljen u **globalni repozitorij gotovo uvijek u neoperativnom stanju** (npr. ne kompajlira se).
- ➋ Ako developeri *push-aju* promjene tek kad su u potpunosti gotovi i testirali svoj dio posla, onda:
 - Mogu nastati velike razlike između globalnog i lokalnog repozitorija
↪ puno kompliciranih konflikata.
 - Developer ne može raditi *backup* svog koda u globalni repozitorij.
 - Developer ne može raditi na više računala istovremeno.

- 1 Dozvolite pristup svom globalnom repozitoriju iz **Zadatka 1** kolegi sa susjednog računala.
- 2 Neka kolega klonira vaš repozitorij na lokalno računalo.
- 3 Potrebno je pomoću CSS-a dobiti prikaz forme točno kao na slici iz Predavanja 02.
 - Dogovorite se o id-ovima i klasama elemenata na slici.
 - Jedan od vas neka uredi datoteku `forma.html`, a drugi `stil.css`.
- 4 Slijedite *centralized workflow* u nekoliko iteracija.
- 5 Napravite i razriješite barem jedan konflikt.

Feature branch workflow

- Nadogradnja: *feature branch workflow*
 - Projekt sadrži **glavnu granu** – tzv. **master** ili **main**.
 - **Svaki put** kada developer želi implementirati neku funkcionalnost (*feature*), stvorit će **novu granu** projekta.
 - Nova grana sadrži potpunu kopiju master-a/main-a, ali je potpuno nezavisna od njega.
 - Developer dodaje, mijenja, *commit*-a i *push*-a kod u svoju granu.
 - I drugi developeri se mogu pridružiti razvoju nove grane.
 - Kad je funkcionalnost u potpunosti gotova, developer obavijesti druge (tzv. **pull request**).
 - Nakon revizije koda, nova grana se integrira u master i nestaje.



Rad s granama (branch) u git-u

- `git checkout -b anin-dio-posla main`
 - Ako ne postoji, stvara novu granu s imenom "anin-dio-posla".
 - Nova grana sadrži potpunu kopiju svih datoteka iz trenutne verzije main grane.
 - Ako grana već postoji, samo se prebacuje u tu granu $\rightsquigarrow$ u radnom direktoriju postavlja datoteke iz grane "anin-dio-posla".
- U novu granu posve jednako kao u master dodajemo nove datoteke (add+commit+push; pull).
- Više developera može raditi na novoj grani $\rightsquigarrow$ konflikti.
- `git push -u origin anin-dio-posla`
 - Iznimno, prvi push nove grane treba izgledati ovako.
 - Kasnije je dovoljno samo `git push`.
- Lagano prebacivanje između grana:
 - `git checkout main`
 - `git checkout anin-dio-posla`

Pull request

- Developer je implementirao i testirao kod u svojoj grani
↪ želi ju spojiti s `main` granom.
- Pristojno je obavijestiti druge developere da veći komad koda dolazi u `main`.
- **Pull request** – zamolba drugim developerima da analiziraju i komentiraju kod, nakon čega će doći do spajanja.
- Na web-stranici GitHub-a:
 - Pull requests -> New pull request
 - Treba odabrati granu ("compare") koja će se spojiti u drugu granu ("base" - tipično `main`).
 - Treba navesti osobe koje će pregledati kod koji se dodaje (Reviewers).
 - Možemo vidjeti popis `commit`-ova i razlika (`diff`) datoteka iz grane i datoteka iz `main`-a.
 - Osobe uključene u pull request mogu komentirati i mijenjati kod.
 - Na kraju ↪ Merge pull request.
 - Tada možemo i obrisati granu koju smo spojili u `main` granu.

Spajanje grane u main iz komandne linije

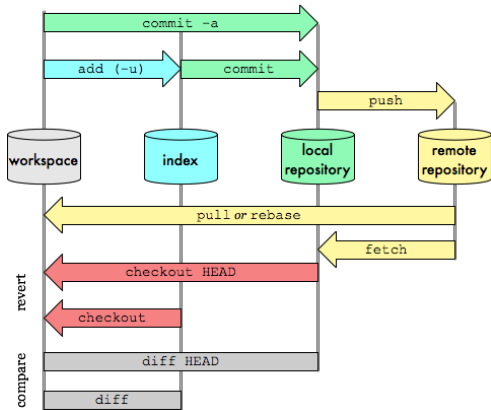
- 1 Spremiti sve promjene u grani: `commit, push`.
 - 2 Promijeniti granu u master: `git checkout main`.
 - 3 Dohvatiti zadnju verziju main-a: `git pull -rebase origin`.
 - 4 Napraviti spajanje grane u main:
`git pull origin anin-dio-posla`.
 - 5 Dojaviti spajanje u globalni repozitorij: `git push`.
 - 6 Obrisati granu koja je bila spojena:
`git branch -d anin-dio-posla`.
- Spajanje može napraviti bilo autor grane bilo reviewer.
 - Konflikti kod spajanja se rješavaju kao i ranije:
korekcija datoteke -> `add` -> `commit` -> `push`.
 - `git branch -list` – popis svih postojećih grana.
 - Cilj: **main uvijek sadrži korektan kod** – može se kompajlirati i radi nešto smisleno.

Zadatak 3

- 1 Dozvolite pristup svom globalnom repozitoriju iz **Zadatka 1** kolegi sa susjednog računala.
- 2 Neka kolega klonira vaš repozitorij na lokalno računalo.
- 3 Potrebno je napraviti još dvije HTML stranice s formama.
 - Jedna stranica neka sadrži upitnik o omiljenim knjigama.
 - Druga stranica neka sadrži upitnik o omiljenim jelima.
 - U `forma.html` dodajte linkove na obje stranice.
- 4 Slijedite *feature branch workflow*, svatko ima svoju granu.
- 5 Napravite uzajamnu reviziju koda preko "Pull requestova".
- 6 Spojite obje grane u main granu.

Git Data Transport Commands

<http://osteele.com>



GUI za git:

- SourceTree – <https://www.sourcetreeapp.com/>
- SmartGit – <https://www.syntevo.com/smartgit/>