Alati za razvoj softvera

Sistemi za kontrolu verzija, Git, GitFlow

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Uvod

- Eng. Version Control Systems **VCS**
- Programi koji se koriste kada je potrebno pratiti verzije projekta i/ili kada više učesnika radi na projektu
- Danas su nezaobilazni u procesu razvoja softvera
- Datoteke koje se prate zajedno sa svom istorijom promena se nalaze na udaljenom mestu — **repozitorijum**
- Svaka izmena koja se uradi, treba da se sačuva na repozitorijumu — **commit**
- Pored samih izmena čuva i informacije o tome ko je napravio izmenu, kad je izmena napravljena i šta je tačno izmenjeno

- Dostupan je istorijat izmena, pa je u svakom trenutku moguće pristupiti ranijim verzijama projekta i svim relevantnim podacima
- Rad sa repozitorijumima i VCS sistemima u velikoj meri olakšava kolaborativni razvoj softvera (u odnosu na ručno održavanje):
  - omogućavaju integraciju promena nastalu od strane različitih članova tima
  - omogučavaju dobijanje informacija ko je, kada, gde, promenio linije koda
- Upravljanje personalnim projektima je znatno olakašano jer je mogućnost greške ili gubitka podataka svedena na minimum
- Ne moraju biti ograničeni samo na programerske projekte!
- Omogučavaju formiranje više alternativnih tokova razvoja projekta, što pruža podršku eksperimentisanju i nezavisnom razvoju delova projekta

Tipovi

- Dva osnovna tipa sistema za kontrolu verzija:
  - **Centralizovani** (Subversion – SVN, CVS, ...)
  - **Distribuirani** (Git, Mercurial...)
- Ova podela je načinjena na osnovu načina na koji je implementiran centalni repozitorijum

Centralizovani VCS

- Glavni repozitorijum sa svom istorijom se nalazi na centralnom serveru, dok klijenti preuzimaju svako svoju verziju aktuelnih datoteka
- Jednostavnija implementacija
- Osnovni nedostaci ovakve arhitektre su:
  - nemogućnost offline rada
  - loša skalabilnost sistema
  - centralizovani server predstavlja **single point of failure** za ceo projekat

Distribuirani VCS

- Svaki klijent preuzima potpunu kopiju projekta sa servera
- Svaki čvor predstavlja ravnopravan repozitorijum sa kompletnom istorijom izmena (bilo koji može biti proglašen za glavni u slučaju otkaza servera)
- Dobra skalabilnost —- pogodan za velike projekte (Linux kernel, Kubernetes, Docker, ...)

Moderne primene

- U novije vreme ovi sistemi se ne koriste samo za programski kod
- Postoje implementacije i primene gde se celokupna specifikacija infrastrukture može 'v cuvatiu ovim sistemima
- Možemo pratiti promene kroz vreme
- Možemo videti razlike vrlo lako
- Jednostavna kolaboracija
- Ceo postupak omogućava da infrastrukturu podignemo na nivo softvera (Infrastrucure as Code)

Uvod

- DistribuiraniVCS
- Osnovna stvar koja razlikuje Git od ostalih VCS sistema (i centralizovanih i distribuiranih) je način na koji Git organizuje podatke u repozitorujumu:
  - ostali VCS-ovi čuvaju osnovne kopije datoteka i evidentiraju samo eventualne izmene uvede nad tim kopijama
  - Git za svaku uvedenu izmenu kerira novu sliku stanja celog sistema tako što kreira nove datoteke sa uvedenim izmenama
  - ovo omogućava timovima da ne budu vezani za samo jedan tok tazvoja
  - S obzirom da lokalni git repozitorijumi poseduju kompletnu bazu sa istorijom izmena, potpuni rad sa zajedničkim repozitorijumom je omogućen i bez mrežnog pristupa centralnom serveru
  - Sve načinjene izmene se primenjuju samo na lokalni repozitorijum do trenutka dok se ne pošalju na centralni repozitorijum

Instalacija

- Preuzeti instalaciju sa službene web stranice
- Prvobitna namena je korišćenje iz komandne linije
- Grafička okruženja za Git dostupna su na linku
- Provera uspešne instalacije:

```
git -version
```

- Golang i VSCode (verovatno) imaju integrisanu podršku

Podešavanja

- Nakon instalacije, dobra praksa je da se odmah podesi ime i email adresa koje će Git koristiti prilikom čuvanja izmena:

```
git config --global user.name "Ime Prezime"
git config --global user.email mailadresa@nesto.com
```

- *global* će izvršiti podešavanja za svaki repozitorijum na računaru
- Ukoliko se izostavi, podešavanja se vrše samo za onaj repozitorijum na koji smo trenutno pozicionirani iz komandne linije
- lista trenutno podešenih vrednosti se može dobiti komandom:

```
git config --list
```

Kreiranje lokalnog repozitorijuma

- Dva načina postoje:
  1. **Inicijalizacijom novog (praznog) repozitorijuma**
     - *git init* je komanda koja će prazan kreirati lokalni repozitorijum
     - Kreiraće se sakriveni folder **.git** unutar direktorijuma iz kojeg smo pokrenuli komandu (radnog direktorijuma)
  2. **Kloniranjem postojećeg repozitorijuma sa udeljanog servera**
     - *git clone [putanja_do_repozitorijuma] [lokalni_direktorijum_gde_se_klonira]* je komanda koja će kreirati lokalni repozitorijum sa svim fajlov ima I istorijom izmena sa udeljenog servera
     - U okviru radnog direktorijuma kreira skriveni folder **.git** koji zapravo označava da je folder git repozitorijum
  3. U svakom trenutku je moguće dobiti informacije o stanju repozitorijuma komandom *git status*

Praćenje datoteka

- Kreiranje datoteka u radnom direktorijumu neće navesti Git da prati izmene nad njima
- Pokretanje praćenja verzija nad nekom datotekom ili direktorijumom se vrši komandom *git add [naziv_datoteke]*:
- Komanda *git add* se mora izvršiti za sve datoteke koje su nove, ali i za sve postojeće datoteke nad kojima su izvršene neke izmene (nalaze se u stanju modified)
- Možemo automatski dodati sve izmene komandom *git add .*
- Naknadne izmene nad praćenim datotekama neće biti evidentirane automatski
- Prilikom svake izmene potrebno je ponoviti *git add* komandu

Postavljanje izmena na repozitorijum

- Vrši se u dve faze:
  1. postavljanje promena u deo repozitorijuma koji se zove **staging area (evidentiranje izmena)** — datoteke se u ovaj deo repozitorijuma dodaju komandom *git add* (za objekte koji pripadaju ovom delu se kaže da su u staged stanju)
  2. "pakovanje" evidentiranih izmena u **commit** — predstavlja prebacivanje izmene iz staging area na trenutnu granu repozitorijuma (format naredbe: *git commit -m "opis izmena"*)
- Saveti oko commit-a
  - Logički nezavisne izmene organizovati u odvojenim *commit-ima*
  - Jedan *commit* treba biti moguće opisati kroz rečenicu ili dve
  - Težiti ka davanju prefiksa opisu *commit-a* (add,fix,itd.) kako bi se ukazalo na tipi zmene

Git file status lifecycle

- Postavljanje jednog fajla u staging area:

```
git add <ime_fajla>
```

- Postavljanje svih fajlova u staging area

```
git add .
```

- Brisanje fajla iz fajl sistema i postavljanje te izmene u staging area

```
git rm <ime_fijla>
```

- Poništavanje izmena

```
git reset HEAD <ime_fijla>
```

Postavljanje izmena na repozitorijum

- Operacija *commit* izmene šalje na lokalni repozitorijum
- Operacija *push* šalje lokalne izmene na udaljeni repozitorijum (može više commit-a)
- Uobičajen ikoraci:

```
git add .
git commit -m "Tekst commit poruke"
git push origin master
```

Ignorisanje datoteka

- Pokretanje *git add* komande nad celim direktorijumima može dovesti do toga da u praćenim datotekama budu i neke koje ne želimo da imamo na repozitorijumu (npr. kompajlirani fajlovi nisu potrebni, dovoljno je da imamo source fajlove, logovi, biblioteke...)
- Git omogućava način da navedemo pravila za ignorisanje tih datoteka ili direktorijuma prilikom pokretanja komande *git add*
- Pravila se navode u datoteci sa imenom *.gitignore* koji se kreira u repozitorijumu

Uklanjanje datoteka

- Iz radnog direktorijuma i iz staging area: *git rm [naziv_datoteke]*
- Samo iz staging area: *git rm –cached [naziv_datoteke]*

Poništavanje izmena

- Dopuna prethodnog *commit-a* novimi zmenama: *git commit–amend*
- Vraćanje datoteke u unmodified stanje: *git checkout – [naziv_datoteke]*
- Prebacivanje u stanje nekog *commit-a*: *git checkout [hash]*
- Prebacivanje u stanje nekog *commit-a* i brisanje komplentne istorije nastale posle tog *commit-a*: *git reset –hard [hash]*

git stash komanda

- Koristimo je kako bi lokalno spremili trenutno stanje radnog direktorijuma (bez commit-ovanja) kako bi mogli da se prebacimo na rad na nečemu drugom
- Promene je moguće spremiti više puta, pri čemu se za svako spremanje kreira novi zapis u lokalnom repozitorijumu
- Nakon spremanja izmena, radni direktorijum se vraća u stanje poslednjeg commit-a
- Lista spremljenih izmena se može dobiti komandom *git stash list*
- Vraćanje spremljenih izmena se vrši komandom *git stash apply*
- Ovo će vratiti poslednje spremljeno stanje (sa vrha liste)

Stanje repozitorijuma

- Stanje repozitorijuma možemo dobiti komadom *git status*
- Prikazuje izmene koje:
  - treba da budu postavljene u staging area
  - su postavljene u staging area, ali nisu na repozitorijum
- Prikazuje i informacije kao što su koja je trenutno aktivna grana i broj *commit-a* koji nisu poslati na udaljeni repozitorijum

Istorija repozitorijuma

- Istoriju repozitorijuma možemo dobiti komadom *git log*
- Prikazuje sve *commit-ove*
- Za svaki commit je prikazan opisom, autor i datum
- Svaki commit ima svoj jedinstven identifikator (hash)

Preuzimanje ranije izmene

- Preuzimanje ranije izmene se vrši komadom *git checkout [hash]*
- Ovom komandom se lokalni repozitorijum može vratiti u stanje u kakvom je bio u nekom trenutku

Rad sa granama

- Omogućavaju izolovan rad na različitim komponentama sistema i pružaju dobru podršku testiranju ideja
- Novokreirani repozitorijumi imaju samo jednu granu koja se zove **master (main)**
- Razvijamo komponentu i kad smo zaokružili celinu spajamo **(merge)** kod komponente sa glavnom granom
- Preporuka je da **master** grana (glavna) ima samo merge čvorove — da se ništa ne radi na master grani dire
- **HEAD** pokazivač pokazuje na kojoj grani i čvoru smo trenutno pozicionirani

- Kreiranje grane: *git branch [ime_grane]*
- Pozicioniranje na granu: *git checkout [ime_grane]*
- Prikaz svih grana: *git branch*
- Brisanje grane: *git branch –d [ime_grane]*

Spajanje grana

- Prilikom spajanja, potrebno je vratiti se na roditeljsku granu i pozvati komandu *git merge [druga_grana]*
  - U granu na kojoj smo trenutno pozicionirani ubacuje se izmene napravljene na grani [druga_grana]
  - Izmene se "spajaju" tako da novi čvor sadrži izmene napravljene i na prvoj i na drugoj grani
- Drugi način spajanja je *git rebase*
  - Ne kreira novi commit
  - Uglavnom služi u svrhe dopunjavanja grana izmenama sa master grane

Udaljene grane

- Sve kreirane grane se nalaze na lokalnom repozitorijumu
- Ukoliko želimo da objavimo našu granu na udaljenom serveru koristimo naredbu: *git push origin [naziv_grane]*

Konflikti

- Automatsko spajanje izmena će biti uspešno izvršeno samo ako u granama koje se spajaju ne postoje izmene u istim linijama istih datotetka
- Ako ovo nije slučaj, datoteka čiji sadržaj nije mogao biti "spojen" je u konfliktu
- U datoteci koja je u konfliktu označene su konfliktne linije
- Primer konflikta

```
<<<<<<< HEAD:index.html
contact : email.support@github.com
=======
please contact us at support@github.com
>>>>>>> develop:index.html
```

- Potrebno je srediti sadržaj konfliktne datoteke tako da se obrišu oznake i neželjeni sadržaj, a zatim komadom *git add* dodati datoteku na staging area

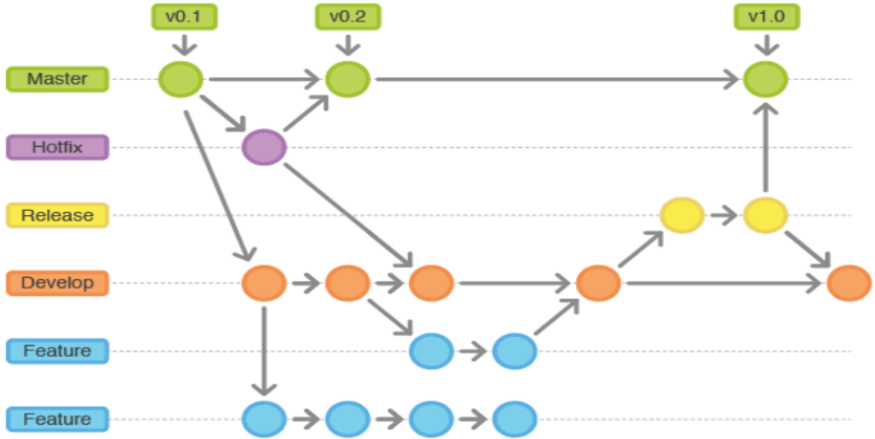
Rad sa udaljenim repozitorijumom

- Iako su svi repozitorijumi u Git-u ravnopravni, čest scenario za timski rad je da postoji jedan javno dostupan repozitorijum koji čuva izmene svih članova tima
- Ovom udaljenom repozitorijumu se obično dodeljuje ime **origin**
- Pri kloniranju repozitorijuma, za **origin** se postavlja repozitorijum čiji se sadržaj klonira
- *git remote add origin [adresa_repozitorijuma]*
- Lokalni git repozitorijum može da bude podešen tako da radi sa više udaljenih repozitorijuma
- Slanje sadržaja lokalnog repozitorijuma na udaljeni repozitorijum *git push [ime_ili_adresa_repozitorijuma] [ime_grane_koju_saljemo]*

Preuzimanje izmena sa udaljenog repozitorijuma

- Preuzimanje izmena sa udaljenog repozitorijuma: *git fetch [ime_ili_adresa_repozitorijuma]*
- Preuzete izmene je potrebno onda spojiti sa lokalnom granom: *git merge origin/master* (spajanje se radi sa master granom udaljenog repozitorijuma)
- Prethodne dve naredbe se mogu objediniti u jednu: *git pull [ime_ili_adresa_repozitorijuma]*

GitFlow model



(Image from https://anarsolutions.com/gitflow-branching-model/)

Dodatni materijali

- Gitflow
- Git docs
- Git krakern tutorial — Git UI tool
- Git tutorial

Kraj predavanja

Pitanja? :)