Git 101

Kako koristiti Git?

TUZLA,
Oktobar 2024.

Uvod

Jedna od prvih stvari koju moramo razmotriti kada kreiramo file-ove sa kodom jeste da ti file-ovi moraju biti dostupni nama i svim ostalim koji koriste te file-ove. Idealno, to bi bilo dostupno sa jedne centralne lokacije gdje će se uz file-ove nalaziti i kopije istih file-ova. Ovo omogućava da nadograđujemo i popravljamo kod po potrebi, te da imamo dostupan hronološki zapis tih izmjena za svaki pojedinačni file sa pomenute centralne lokacije. (**Dostupnost**)

Drugi aspekt jeste **kolaborativni proces** između članova tima, odnosno mogućnost da se zajednički radi na tim file-ovima, t.j. na projektu. Cilj je da bilo koji tekstualni file se može pratiti sa višestrukim unosom čije su promjene vidljive **svima** u timu ili nekoj skupini.

Treći i zadnji aspekt je **odgovornost**. Kada imamo sistem koji omogućava višestruk unos i promjene, potrebno je označiti te promjene odgovarajućim evidencijama, odnosno zapisima (eng. track record) koji se odražavaju na vlasnika promjena. Evidencija bi također trebala sadržavati kratak razlog za promjenu kako bi osoba koja pregledava historiju nekog projekta ili pojedinačnog file-a, mogla razumjeti zašto je promjena napravljena.

Ovi aspekti predstavljaju jedne od glavnih izazova **sistema kontrole verzije** (version-control system) poput **Git** -a (Global Information Tracker).

Prednosti Git -a kao sistema kontrole verzije:

- **Distribuirani razvoj** - paralelni, nezavisan i simultan proces razvoja u privatnim "sistemima" offline bez potrebe za konstantom sinhronizacijom sa centralnim "sistemom"
- **Performanse** - ušteda na vremenu i fizičkim resursima
- **Odgovornost i nepromjenjivost** - Git nameće dnevnik promjena za **svaku** izmjenu tako da uvijek postoji trag neke promjene i razlog.
- **Besplatan alat**

Git Terminologija

Prije nego što vidimo kako Git funkcioniše na primjerima potrebno je se upoznati sa relevantnom terminologijom. 1

- ☞ **Repozitorij** - Predstavlja spremište ili bazu podataka svih relevantnih informacija vezane za projekat poput file-ova, metapodataka i kompletne historije promjena.
- ☞ **Branch** - "Grana" je zapravo aktivna linija razvoja. Zadnji **commit** je **vrh** ili **HEAD** te grane.

  Repozitorij može imati više grana pomoću kojih možemo razvijati neki dodatak postojećem softveru ili popravljati neki bug, bez da utiče na glavni dio projekta ili **main** granu.
- ☞ **Working tree** - Predstavlja trenutačno "okruženje" u kojem radimo odnosno stablo. Naše stablo može biti asocirano samo sa **jednim** branch-om.
- ☞ **Checkout** - Komanda kojom ažuriramo čitavo ili dio stabla na neku "tačku". Često služi za promjenu branch-a.
- ☞ **Commit** - Predstavlja tačku u vremenu koju je Git zabilježio, odnosno predstavlja novi snapshot (snimak) u repozitoriju.
- ☞ **Fetch** - Ovo je akcija kojom preuzimamo sav sadržaj iz nekog udaljenog repozitorija u lokalni rep ozitorij.
- ☞ **Push** - Komanda kojom se izvršava upload lokalnog repozitorija na udaljeni repozitorij, uz file-ove prebacuju se i svi metapodaci kao i svaki commit napravljen do tog trenutka
- ☞ **Pull** - Fetching i merging repozitorija

[1] https://git-scm.com/docs/gitglossary

Instalacija i kreiranje prvog repozitorija

Git može se instalirati na Linux baziranim sistemima preko terminala iz respektivnih packet manager-a. Za Ubuntu 22.04 LTS verziju date su sljedeće komande za instalaciju.

```
$ sudo apt update
$ sudo apt install -y git
$ git --version
git version 2.34.1
```

Kada je Git instalisan potrebno je postaviti nekolikog konfiguracijskih parametara poput name i email, to možemo uraditi na sljedeći način:

```
$ git config --global user.name "imePrezime"
$ git config --global user.email "ime.prezime@fet.ba"
$ git config --list
user .name = imePrezime
user . email = ime. prezime@fet .ba
```

Alternativno može se urediti i .gitconfig file preko kojeg ujedno možemo i provjeriti našu konfiguraciju.

```
$ cat ~/.gitconfig
[ user ]
name = imePrezime
email = ime.prezime@fet.ba
...
$ sudo gedit ~/.gitconfig #uredivanje file-a
```

Nakon toga napravimo folder u kome će se nalaziti naš lokalni repozitorij i izvršimo incijalizaciju sa komandom git init. Pri inicijalizaciji git repozitorija po default-u se kreira branch pod nazivom "*master*", u ovom primjeru promjeniti ćemo da naziv default branch-a bude "*main*".

```
$ git config --global init.defaultBranch main # opcionalno
$ mkdir firstRepo
$ cd firstRepo
$ git init
Initialized empty Git repository in /home/student/firstRepo/.git/
```

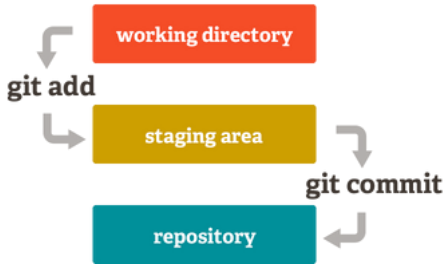
Ovo je jedan on načina da započnemo novi projekat pomoću Git-a. Nakon uspješne inicijalizacije kreira se .git direktorij koji sadrži sve relevantne podatke i metapodatke vezane za naš projekat poput svih referenca za svaki commit i branch. Ovaj folder se uvijek nalazi u *root* direktoriju projekta.

```
$ ls .git/
branches config description HEAD hooks info objects refs
```

Sljedeći korak je da dodamo kreiramo neki file u našem projektu kako bismo ga dodali u repozitorij.

```
$ echo "first file in repo" > firstFile.txt
```

Nakon kreiranja file-a potrebno je file dodati u **staging area**. Svaki file u git repozitoriju može biti **tracked** ili **untracked**. U suštini, **tracked** file-ovi su oni file-ovi koji su "poznati" git-u, odnosno za koje će git pratiti promjene. Dok **untracked** file-ovi su svi ostali file-ovi u radnom direktoriju, odnosno nisu u staging area.



Kako bismo dodali neki file u staging area potrebno je to uraditi sa komandom **git add –flags filename**. Opcionalno ukoliko želimo dodati sve file-ove za koje postoje promjene a nisu u već u staging area to možemo uraditi sa zastavicom 'A'. Na ovaj način ti file-ovi postaju tracked.

```
$ git add firstFile.txt
#ili
$ git add -A
```

Sa komandom **git status** možemo pogledati trenutačni status našeg direktorija. Tu možemo vidjeti na koji trenutačno branch git "gleda", te ukoliko imamo untracked file-ove i koji file-ovi su spremni za naredni commit. Pogledajmo status našeg repozitorija prije i poslije git add komande.

```
#prije komande "git add"
$ git status
On branch main

No commits yet


Untracked files:
  (use "git add <file >..." to include in what will be committed)
        firstFile.txt

nothing added to commit but untracked files present (use "git add" to
    track)

...

#nakon komande "git add"
$ git status
On branch main

No commits yet Changes to

be committed:
  (use "git rm --cached <file >..." to unstage)
        new file:       firstFile.txt
```

Vidimo da je sada naš file u staged statusu tako da potrebno je dodati file našem repozitoriju sa git commit komandom kako bi promjene bile uspješno dodate. Uz ovu komandu možemo dodati zastavicu 'm' poslije koje dodajemo deskriptivnu poruku kojom opisujemo napravljene promjene u paru dvostrukih navodnika. Poruke treba da u što manje riječi opišu promjene radi bolje čitljivosti commit poruka.

```
$ git commit -m "first commit"
[main (root-commit) 34fe160] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 firstFile.txt
```

Nakon ponovne provjere statusa repozitorija možemo primjetiti da nema više untracked file-ova i da su sve promjene uspješno upisane u repozitorij.

```
$ git status
On branch main
nothing to commit, working tree clean
```

Napravimo neke promjene na file-u i ponovimo proceduru za dodavanje promjena na repozitorij.

```
$ vim firstFile.txt $ cat
firstFile.txt making some
changes in file $ git status

On branch main
Changes not staged for commit:
  (use "git add <file >..." to update what will be committed) (use "git
  restore <file >..." to discard changes in working
    directory)
        modified:   firstFile.txt

no changes added to commit (use "git add" and/or "git commit -a")

$ git add -A
$ git commit -m "modified firstFile.txt"
[main 69a68cc] modified firstFile.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Primjetimo da nakon što je file promijenjen da je git prepoznao promjene.

Heksadecimalni broj "69a68cc..." je git commit broj koji predstavlja **SHA-1**[2] hash stanja repozitorija u trenutku pravljenja commit-a. Ova vrijednost predstavlja jednu od bitnijih karakteristika Git alata. Naime, da smo ponovoli istu proceduru na nekom drugom uređaju hash vrijednost bi bila ista. Na ovaj način Git može prepoznati da su dva repozitorija identična iako se na njima radilo paralelno.

[2] Secure Hash Algorithm - kriptografski algoritam koji za dati ulaz daje na izlazu 20 bajtnu hash vrijednost

Još jedna od mogućnosti jeste da možemo vratiti repozitorij u prethodno stanje pomoću ove vrijednosti ili klonirati udaljeni repozitorij od neke tačke razvoja (commit-a).
Kako bismo iskoristili ovu vrijednost prvu moramo kako pristupiti njenom cijelom zapisu za svaki pojedinačni commit. To možemo učiniti pomoću komande git log koja nam daje dnevnik promjena i ispis svakog commit-a u obrnutom hronološkom redoslijedu. Sadržaj se sastoji od username-a autora koji je napravio commit i njegova email adresa, potom datum, commit poruka kao i prethodno pomenuta SHA-1 hash vrijednost.

```
$ git log
commit 1e252eb9df426aceeb7791eb659feb3997991c9d (HEAD -> main)
Author: admirMustafic <admir.mustafic@fet.ba>
Date:       FriMar1520:11:502024+0100

    modified firstFile.txt
```

Sa komandom git revert možemo vratiti repozitorij u stanje odabranog commit-a tako što komandi proslijedimo commit broj. U našem slučaju to je "*1e252eb9df426aceeb7791eb659feb3997991c9d*".

```
$ git revert 1e252eb9df426aceeb7791eb659feb3997991c9d
$ cat firstFile.txt
first file in repo
```

Na ovaj način smo vratili repozitorij u željeno stanje, na sličan način to možemo i učiniti sa komandom git reset.

Branch i merge

Kao što je već pomenuto **branch** ili grana prestavlja aktivnu liniju razvoja. Pomoću komande git branch 'naziv' možemo kreirati novi branch. U većini slučajeva novi branch se stvara ukoliko se želi odvojeno razvijati kod od glavnog brancha (main ili master) u svrhu popravljanja neke greške, odvojenog feature-a, nove linije razvoja i tako dalje. U našem primjeru kreirajmo novi branch koji ćemo nazvati dev.

```
$ git branch dev
$ git branch
  dev
* main
```

Na osnovu komande i indikatora ***** možemo uvijek provjeriti u kojem smo branch-u. Promjenu branch-a uvijek možemo izvršiti sa komandom git checkout 'branch-naziv'.
Kreirajmo još jedan file kako bi radili odvojeno na njemu u odnosu na main branch.

```
$ git checkout dev
$ echo "new file in dev branch" > secondFile.txt
$ git add secondFile.txt
$ git commit -m "adding a second file to dev branch"
[dev 76716e4] adding a second file to dev branch
 1 file changed, 1 insertion(+)
 create mode 100644 secondFile.txt


$ ls
firstFile.txt    secondFile.txt
```

Vidimo da su vidljiva oba file-a u dev branch-u. Sada vratimo se u main branch kako bi potvrdili da su ove dvije linije razvoja odvojene.

```
$ git checkout main
Switched to branch 'main'
$    ls
 firstFile.txt
```

Kako bismo spojili dva branch-a, odnosno da sadržaj dev branch-a ude pristuan i u main branch-u potrebno je izvršiti komandu git merge.

```
$ git merge dev main
Merge made by the 'ort' strategy.
 secondFile.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 secondFile.txt

$ ls
firstFile.txt secondFile.txt
```

Sada možemo ponovno provjeriti sa komandom ls da je sadržaj iz dev branch-a dostupan i u main branchu, odnosno da je merge uspješan što vidimo da je doista to i slučaj.