Git-Github

Git i git Hub

Uvod u Git

Ako već koristite GNU/Linux, a niste još imali prilike da koristite Git, u ovom tekstu ćemo vam objasniti šta je Git i na koji način možete da ga koristite.

Idejni tvorac Git-a je Linus Torvalds, inače tvorac Linux-a, koji ga je razvio dok je radio na razvoju Linux kernela. Od tog vremena Git je postao prihvaćen u mnogim softverskim projektima kao standard u praćenju razvoja softvera u svim njegovim fazama.

Šta je Git

Git je specifična implementacija alata za kontrolu verzije, poznatiji kao distribuirana revizija kontrole verzije, sa mogučnošću praćenja izmena nad setom fajlova. On omogućava praćenje istorije promena, kako lokalno tako i u zajedničkom radnom okruženju. Prednost ovog praćenja je u tome što znamo šta je menjano, kada je menjano, ko ga je menjao i zašto ga je menjao. Dobra strana svega je što tako praćene izmene na kraju mogu biti spojene u jednu istoriju.

S obzirom da je u pitanju distribuirani sistem za kontrolu verzije, svaki računar poseduje ceo repozitorijum koji se čuva lokalno, dok server služi za razmenu i saradnju između korisnika. Svaki računar ima kopiju podataka koja se čuva lokalno i koja se sinhronizuje sa serverom.

Čemu Git služi

Važno je da odmah napomenemo da Git može u potpunosti biti korišćen od strane pojedinca, bez potrebe da se ikada konektuje na server i sarađuje sa drugima, ali da on to može jednostavno učiniti kada god mu to bude potrebno.

Git je pre svega usmeren ka procesiranju lokalnog sadržaja. Ako ste početnik, dovoljno je da se prvo fokusirate na opcije koje vam Git nudi za praćenje vaših projekata na lokalnom računaru.

Instalacija Git-a

Instalacija Git-a na GNU/Linux-u je jednostavna. Radite je uz pomoć package manager-a iz komandne linije. Evo nekoliko primera kako to možete da uradite u nekim popularnim distribucijama.

Na Debian sistemima, ili onima koji su zasnovani na Debian-u (kao Ubuntu):

$ sudo apt-get install git

Na Redhat Enterprise Linux i sistemima zasnovanim na RedHat-u, kao što je Fedora koristite yum:

$ sudo yum install git

napomena: na Fedori 22 ili kasnije zamenite yum sa dnf:

$ sudo dnf install git

Konfigurisanje Git

Sada kada ste instalirali Git potrebno je da ga konfigurišete da biste ga koristili. Prvu stvar koju treba da uradite je da konfigurišete svoj email i koriničko ime u Git-u. Imajte u vidu da oni neće biti korišćeni za logovanje na neki sistem, već za dokumentovanje izmena koje pravite.

Da biste konfigurisali vaš email i korisničko ime unesite sledeće komande u terminal:

```
$ git config --global user.email "email@domen.com"
```

```
$ git config --global user.name "korisnicko ime"
```

(Naravno, ne zaboravite da umesto "email@domen.com" i "korisnicko ime" uneste vaš email i vaše korisničko ime.)

Vaš prvi Git projekat

Vaš prvi Git projekat

Da biste setovali vaš prvi projekat, on mora biti inicijalizovan pomoću ove komande:

$ git init projectname

Na taj način ćete kreirati direktorijum u vašem radnom direktorijumu sa imenom vašeg projekta. On će sadržati projektne fajlove/foldere zajedno sa kontrolnim fajlovima koji se koriste za praćenje. Git čuva ove kontrolne fajlove u .git skrivenom direktorijumu.

Kada radite sa Git-om, trebalo bi da započnete sa novim projektnim folderom:

$ cd projectname

Korišćenjem touch komande kreirate prazan fajl koji ćete koristiti da kreirate jednostavn hello world program:

$ touch helloworld.c

Da bismo fajlove u direktorijumu pripremili da budu dodati sistemu za kontrolu verzija, koristićemo komandu **git add.** Ovo je proces poznat pod nazivom stagging. Za dodavavanje svih fajlova u direktorijumu kosristićemo tačku (.).

Ukoliko želite da dodate samo odabrane fajlove, onda je potrebno da tačku (.) zamenite sa imenom nekog određenog fajla.

Git commit

Commit ćete koristiti za kreiranje trajnog istorijskog zapisa o tome kako izgledaju projektni fajlovi u datom trenutku. U našem slučaju to je samo jedan prazan fajl. Commit koristimo dodavanjem flag-a **–m**.

$ git commit -m "Prvi commit projekat je samo prazan fajl"

Sada možemo da kreiramo naš prvi kod u tom praznom fajlu. Otvorite bilo koji tekst editor i unesite sledeći kod u naš helloworld.c fajl. Ne zaboravite da na kraju sačuvate fajl.

```
#include

int main(void)

{

printf("Hello, World!\n");

return 0;

}
```

Sada kada smo ažurirali naš projekat, možemo ponovo da pokrenemo git add i git commit:

```
$ git add helloworld.c
$ git commit -m "dodat izvorni kod u helloworld.c"
```

$ git commit -m "dodat izvorni kod u helloworld.c"

Čitanje logova

Sada kada imamo dva commit-a u našem projektu, možemo da pogledamo kako izgledaju logovi sa istorijskim podacima. Da bismo videli logove kucamo:

$ git log

Kao što ćete primetiti svaki commit je organizovan po svom jedinstvenom SHA-1 hash ID-u. Poslednji sačuvani commit je označen kao head i predstavlja našu trenutnu poziciju u projektu.

Da biste videli koje su promene sačuvane u određenom commit-u, jednostavno pokrenite **git show** komandu, sa njegovim hash id-em kao argumentom:

$ git show 6a9eb6c2d75b78febd03322a9435ac75c3bc278e

Rad sa udaljenim repozitorijumom

Sada kada smo prošli kroz osnove koristeći lokalni računar, vreme je da vidimo kako možete raditi sa Git-om kada su u pitanju udaljeni repozitorijumi i projekti koji se hostuju na njima. Projekat može biti hostovan na privatnom git serveru koji je u vlasništvu neke organizacije, ili može biti hostovan na nekom third-party online hosting servisu kao što je Github.

Za potrebe ovog teksta pretpostavimo da imate neki projekat koji hostujete na GitHub-u. Pre svega je potrebno da klonirate repozitorijum lokalno koristeći git clone komandu i URL projekta i ujedno praveći direktorijum kloiniranog projekta kao naš trenutni radni direktorijum.

$ git clone project.url/projectname.git

```
$ cd projectname
```

Zatim treba da editujemo lokalne fajlove, implementirajući promene koje želimo. Nakon toga treba da ih dodamo u stagging sekciju i pokrenemo commit kao u našem prethodnom primeru:

```
$ git add .
```

```
$ git commit -m "dodate promene"
```

Nakon izmene na lokalnom serveru, sada treba da pošaljemo te izmene na git server korišćenjem sledeće komande:

```
$ git push
```

Pre nego je pokrenete važno je da znate da ova komanda zahteva da se autentifikujete na udaljenom serveru (u ovom slučaju to bi bili korisničko ime i lozinka za GitHub), pre nego što je pokrenete. Time ste ujedno i zatvorili krug od instalacije, preko konfigurisanja i na kraju čuvanja i upload-ovanja sačuvanih izmena na udaljeni server. Ujedno ste i savladali prve korake u korišćenju Git-a, što je savim dovoljno da počnete da ga samostalno koristite za sopstvene potrebe.

Git komande za početnike

U prošlom delu smo pisali šta je Git i kako se on koristi. U ovom delu ćemo predstaviti koje su po nama najkorisnije Git komande za početnike i objasniti kako da ih primenite u okviru standardnog Git workflow-a.

Pre toga da se podsetimo kako da primenite osnovni Git workflow kada radite u timu:

- kreirajte vaš repozitorijum na centralnom serveru (to će biti folder koji će biti praćem od strane Git softvera)
- preuzmite repozitoijum na vaš lokalni računar (klonirajte ga)
- dodajte fajlove na taj repozitorijum i commit-ujte ih. Git radi snapshot vašeg projekta (fajlova u repozitorijumu) kada god commit-ujete nešto i čuva istoriju snapshot-ova za kasniju upotrebu.
- upload-ujte izmenjene fajlove na server
- preuzmite sve fajlove koji su modifikovani od strane ostalih članova tima.

Ukratko ovo je osnovni workflow svakog Git projekta. Sada da vidimo kako to možemo da realizujemo pomoću Git komandi.

Kreirajte repozitorijum

Do sada već znate da Git po default-u ne prati svaki pojedinačni fajl na vašem računaru. Morate eksplicitno da mu date informaciju koji fajl želite da bude praćen. Da biste to uradili otvorite terminal i izvršite sledeću komandu

$ git init

Ova komanda praktično pretvara postojeći direktorijum u repozitorijum. U osnovi on govori Git-u da započne praćenje svih fajlova i foldera unutar repozitorijuma.

Obratite pažnju da sve Git komande počinju sa Git.

Razvojno okruženje Git-a se sastoji od tri sekcije. Važno je da ih ovde prođemo kako biste bolje razumeli sam koncept Git-a jer sve dalje aktivnosti zavise od njega

1. Radni direktorijum se odnosi na trenutno stanje fajlova i foldera unutar vašeg sistema fajlova. U ovom delu Git još uvek ne prati vaše fajlove.

2. Staging area je mesto u koje smeštate fajlove pre nego ih sačuvate u repozitorijumu. To je kao neka privremena lokacija za vape foldere i fajlove pre nego pokrenete commit. Fajlove unutar staging area možete jednostavno dodavati i uklanjati. Ako ste modifikovali neke fajlove i želite da ih dodate u staging area jednostavno izvršite sledeću komandu:

$ git add .

Kao što smo već pomenuli u prethodnom tekstu, tačka (.) na kraju znači da su modifikovani svi fajlovi.

3. Repozitorijum sadrži sve vaše commit-ovane fajlove. Sve ove informacije Git čuva u skrivenom folderu pod nazivom .git. Kada commit-ujete nešto, šta god da je u vašoj staging area se trajno čuva u repozitorijumu. **Možda je lakše da commit shvatite kao checkpoint. Kada commit-ujete on poredi vaš prethodni checkpoint sa sadašnjim i tako čuva samo modifikovane fajlove**. Dobra je praksa da prilikom commit-ovanja pokrenete komandu sa kratkom porukom šta je izmenjeno:

$ git commit –m “ovde unesite kratak opis izmene”

Commit poruke su veoma korisne (nešto slično komentarisanim redovima prilikom pisanja koda), jer ostalim članovima tima daju više informacija o tome šta ste menjali).

Provera statusa

Git vam omogućava da jednostavno proverite koji se fajlovi prate unutar staging area. Da biste to uradili koristite sledeću komandu:

$ git status

Povezivanje lokalnog repozitorijuma sa serverom

U realnom scenariju, možda će vam biti lakše da prvo kreirate repozitorijum na nekom Git hosting provajderu (npr. GitHub), a zatim ga klonirate na vaš lokalni računar. Razlog tome je što se u tom slučaju automatski setuje i remote repozitorijum, što znači da su vaš lokalni i udaljeni repozitorijum tada povezani po default-u.

Ipak, ako ste kreirali repozitorijum prvo na vašem lokalnom računaru i nakon toga želte da ga povežete sa serverom, onda je potrebno da pokrenete sledeću komandu:

$ git remote add [remote_name] [remote_url]

Umesto [remote_name] možete staviti šta god želite, ali neko opšteprihvaćeno pravilo je da se stavlja origin. Sa druge strane remote_url je putanja do vašeg Git repozitorijuma na serveru. Ova URL će vam biti dodeljena od strane vašeg Git hosting provajdera.

Evo primera kako bi gore napisana komanda moga da izgleda na konkretnom primeru:

$ git remote add origin https://github.com/nmh/bootstrap.git

Ukoliko želite da proverite koliko udaljenih repozitorijuma je povezano sa vašim lokalnim repozitorijumom, koristite sledeću komandu:

$ git remote –v

Upload-ujte izmene na server

Sada ste povezali vaš lokalni repozitorijum sa udaljenim. Vreme je da upload-ujete izmene na server. Evo sintakse koju koristite u ovom slučaju:

$ git push [remote_name] [branch_name]

Naravno remote_name i branch_name ćete zameniti pravim nazivima. Podrazumevano imamo samo jedan branch koji je poznatiji kao „master“. Branch-eve ćemo pomenuti malo kasnije.

Dakle, primer te komande bi mogao da izgleda ovako:

$ git push origin master

Preuzmite izmene sa servera

Već smo pomenuli da osim upload-a vaših izmena na server, isto tako možete preuzeti izmene urađene od strane članova vašeg tima. Njih možete preuzeti pomoću sledeće komande:

$ git pull [remote_name] [branch_name]

Opet, na jednom primeru to može da izgleda ovako:

$ git pull origin master

Branches

Git branching model je sličan standardnom modelu granjanja (branch = grana). On je posebno koristan kada želite da dodate novu funkcionalnost ili rešite neki bag u vašem projektu.

U suštini branch kreirate onda kada je potrebno da nešto testirate odvojeno. Da biste kreirali novi branch koristite ovu komandu:

$ git branch <branch_name>

Da biste izbrisali branch koristite sledeću komandu:

$ git branch – D <branch_name>

Spajanje (merge)

Pretpostavimo da ste napravili neke izmene u novom branch-u i želite da ih spojite sa vašim master branch-om. To možete uraditi tako što ćete izvesti operaciju spajanja (merge) koristeći sledeću komandu:

$ git merge <branch_name>

ova komanda se koristi za spajanje dva branch-a. Ukoliko je neki fajl izmenjen u oba branch-a, onda će doći do konflikta prilikom spajanja. U tom slučaju morate da ručno pregledate fajl i odaberete izmene koje želite da sačuvate, a ostale da uklonite.

Kako obrisati Git keš

Dok radite sa Git-om, imaćete mnogo situacija gde se nakon dodavanja novih linija koda u vaš .gitignore fajl, ignorisani fajlovi i dalje pojavljuju u ‘’git commit’’ staging area. Kada vam se ovo dogodi, najbolji način da rešite problem je da obrišete i očistite vaš Git keš. U ovom tekstu ćemo upravo objasniti kako obrisati Git keš.

Pre nego što nastavimo da pre svega objasnimo šta je to Git keš, šta on radi I koje prednosti donosi njegovo brisanje.

Šta je Git keš?

Git keš sadrži radni direktorijum koji se sastoji od tri grane: repository, commit i branches. Sve promene u njima se beleže pomoću komande git commit, koja radi snapshot promena u određenom trenutku. Keš pomaže da odaberete određene promene u radnom direktorijumu pre nego ih commit-ujete ili preuzmete najskorije commit-e.

Bez Git keša, Git commit bi vratio promene u radni direktorijum pre čuvanja novih promena.

Šta Git keš radi?

Suština staging area je da poboljša preformanse rešavanjem konfliktnih spajanja commit-a i smanjivanjem stalne potrebe za preuzimanjem biblioteka i sličnih sadržaja kad god se za to pokaže potreba.

Gde da pronađete Git keš

Git keš se nalazi unutar fajla pod imenom index koji se nalazi u .git direktorijumu. Da biste pronašli ili došli do Git keš fajla, pokrenite sledeću komandu da biste ušli u .git direktorijum.

git rev-parse --git-dir

Jednom kada ste pronašli lokaciju .git direktorijuma, uđite u njega koristeći cd komandu:

cd .git

Sada kada ste unutra, možete locirati index fajl koristeći ls komandu:

ls –la

Da biste videli fajl, koristite sledeću komandu:

file index

Komanda će vam dati izlaz sa tipom fajla, verzijom I brojem ulaza unutar Git keš fajla:

index: Git index, version 2, 5 entries

```
[redacted]~/mte$ git rev-parse --git-dir
.git
[redacted]:~/mte$ cd .git
[redacted]:~/mte/.git$ ls -la
total 44
drwxr-xr-x 7 debian debian 4096 Jul  9 04:00 .
drwxr-xr-x 3 debian debian 4096 Jul  9 03:59 ..
drwxr-xr-x 2 debian debian 4096 Jul  9 03:45 branches
-rw-r--r-- 1 debian debian   92 Jul  9 03:45 config
-rw-r--r-- 1 debian debian   73 Jul  9 03:45 description
-rw-r--r-- 1 debian debian   23 Jul  9 03:45 HEAD
drwxr-xr-x 2 debian debian 4096 Jul  9 03:45 hooks
-rw-r--r-- 1 debian debian  392 Jul  9 04:00 index
drwxr-xr-x 2 debian debian 4096 Jul  9 03:45 info
drwxr-xr-x 5 debian debian 4096 Jul  9 04:00 objects
drwxr-xr-x 4 debian debian 4096 Jul  9 03:45 refs
[redacted]~/mte/.git$ file index
index: Git index, version 2, 5 entries
```

Brisanje Git keš fajla

Da biste obrisali određeni fajl iz Git keša, koristite git rm komandu sa nazivom tog fajla.
Da biste rekurzivno uklonili fajlove iz keša, koristite git rm komandu sa –r flegom.

Osnovna sintaksa za ovu komandu je:

git rm --cached filename

Naravno, ovde treba da zamenite filename nazivom određenog fajla koji želite da uklonite iz Git keša.

Na primer, da biste uklonili fajl mte-info.c koristićete komandu:

git rm --cached mte-info.c

Zatim, proverite da li je fajl uspešno uklonjen koristeći komandu:

file .git/index

Brisanje Git keš fajla

Napomena: izvršavanje komande git rm --cached filename ne briše fajl iz radnog direktorijuma već samo iz staging area.

Sa druge strane, da biste obrisali ceo keš i staging area, koristite komandu git rm sa rekurzivnom opcijom -r.

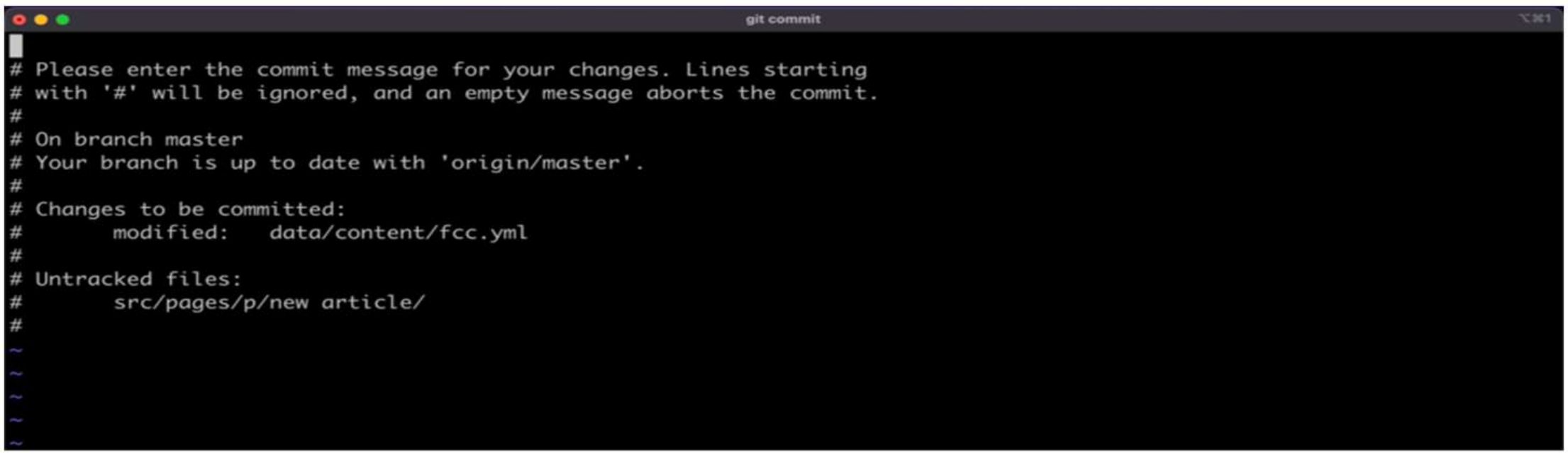
git rm -r –cached

To bi ujedno bilo sve na temu kako obrisati Git keš.

Saveti za konfiguraciju Gita

Izaberite podrazumevani editor za Git

Kada pokušate da napravite commit u Gitu, on će podrazumevano otvoriti vi editor koji izgleda ovako:



Ovaj editor može biti težak za korišćenje, pa ćete možda poželeti da koristite svoj omiljeni editor za pisanje commit-a. U vaš ~/.gitconfig fajl dodajte sledeće

editor = code --wait

ili koristite ovu shell komandu

git config --global core.editor "code --wait"

Ova konfiguracija govori Gitu da za operacije poput commit-a i tagova da želte da koriste na primer VSCode editor.

Za druge tipove editora potražite odgovarajuću komandu na slici ispod

Editor	config command
Atom	`~ git config --global core.editor "atom --wait"~`
emacs	`~ git config --global core.editor "emacs"~`
nano	`~ git config --global core.editor "nano -w"~`
vim	`~ git config --global core.editor "vim"~`
Sublime Text (Mac)	`~ git config --global core.editor "subl -n -w"~`
Sublime Text (Win, 32-bit install)	`~ git config --global core.editor "'c:/program files (x86)/sublime text 3/sublimetext.exe' -w"~`
Sublime Text (Win, 64-bit install)	`~ git config --global core.editor "'c:/program files/sublime text 3/sublimetext.exe' -w"~`
Textmate	`~ git config --global core.editor "mate -w"~`

2. Pruning komanda tokom fetch-ovanja

Pruning komanda tokom fatch-ovanja **je metoda čišcenja** koja briše zastarele udaljene reference u vašem .git direktorijumu kada izvršite **git fetch --prune**

Ovo možete automatizovati bez da uvek dodajete opciju --prune. Da biste to uradili, dodajte sledeće u **~/.gitconfig**:

[fetch] prune = true

ili koristite sledeću komandu:

git config --global fetch.prune true

Sa ovim na mestu, pruning će se desiti kad god izvršite git fetch

3. Git aliasi

U Git konfiguracionom fajlu možete da dodate pseudonime za one dugačke komande koje ukucate s vremena na vreme. Na primer, commit, stashing i tako dalje.

Recimo da želite da dodate alias za dodavanje praznog commit-a. U tom slučaju možete dodati sledeće u konfiguracioni fajl:

[alias] empty = "git commit --allow-empty"

ili u terminalu:

git config --global alias.empty "git commit --allow-empty"

a možete koristiti naredbu i ovako:

git empty "Empty commit"

Takođe možete dodati druge shell komande izvan Gita kao aliase. Na primer, alias koji briše lokalne grane koje su spojene:

[alias] delete-local-merged = "!git fetch && git branch --merged | egrep -v 'master' | xargs git branch -d"

Znak uzvika "!" kaže Gitu da ga pokrene kao shell komandu, a ne kao git * komandu.

Za alias, radimo **git fetch**. Zatim dobijamo spojene grane, prenosimo to kao ulaz u egrep komandu, filtriramo „master“ granu i brišemo grane.

4. Setovanje podrazumevane grane

Prilikom inicijalizacije repozitorijuma (git init), podrazumevana grana je master.

Ne morate da kreirate novu granu koja bi se zvala na primer main, pa da onda brišete glavnu granu i onda da koristite glavnu kao podrazumevanu. To je dug proces. U Git konfiguracionom fajlu možete postaviti podrazumevanu granu nakon Git inicijalizacije. Evo kako:

[init] defaultBranch = main (ili bilo koje drugo ime)

Na ovaj način, git init bi kreirao „main“ granu kao podrazumevanu.