3.1 Grananje u Gitu - Grananje ukratko

Skoro svaki VCS ima neku vrstu podrške za grananje. Grananje znači da se divergira od glavne linije razvitka programa i da se nastavi rad bez uticaja na tu glavnu liniju. U mnogim VCS alatima, ovo je nešto skuplji proces, jer se često zahteva pravljenje nove kopije direktorijuma sa izvornim kodom, što zna da potraje za veće projekte.

Neki ljudi Gitov model grananja zovu prvoklasnom osobinom, i zaista, mogućnost koje nudi odvaja Git od ostalih VCS-ova. Ali po čemu je poseban? Način na koji se Git grana je neverovatno jednostavan za obradu, što čini da se operacije grananja izvrše skoro u istom trenutku kada se zadaju, a skakanje s jedne na drugu granu je podjednako brzo. Za razliku od mnogih drugih VCS-ova, Git ohrabruje korišćenje tokova rada koji se često granaju i spajaju, čak i nekoliko puta tokom jednog dana. Razumevanje i savladavanje ove tehnike predstavljaće snažno i unikatno oruđe koje bi u potpunosti moglo da promeni način na koji razvijate svoj proizvod.

**Grananje ukratko**

Da bismo stvarno razumeli kako Git barata grananjem, moramo da se odmaknemo korak unazad i da istražimo kako Git čuva podatke.

Kao što se možda sećate iz [Početak](https://git-scm.com/book/sr/v2/ch00/_getting_started), Git ne čuva podatke kao seriju skupa promena ili razlika, već kao seriju snimaka.

Kada napravite komit, Git čuva komit-objekat koji sadrži pokazivač na snimak objekta koji je stejdžovan. Ovaj objekat sadrži i autorovo ime i mejl, poruku koja je ukucana, i pokazivače na komit ili komitove koji su direktno došli pre ovog komita (roditelja ili roditelje): nula roditelja na inicijalni komit, jednog roditelja za normalni komit, i više roditelja za komit koji je rezultat spajanja dveju ili više grana.

Da bismo vizualizovali ovo, pretpostavimo da imate direktorijum koji sadrži tri fajla, i sve ih stejdžujete a zatim komitujete. Stejdžovanje fajlova računa kontrolnu sumu nad svakim od njih (SHA-1 heš kako smo pomenuli u [Početak](https://git-scm.com/book/sr/v2/ch00/_getting_started)), čuva tu verziju fajla u Git repozitorijumu (Git to naziva blobovima) i dodaje tu kontrolnu sumu na stejdž:

$ git add README test.rb LICENSE

$ git commit -m 'The initial commit of my project'

Kada napravite komit sa git commit, Git pravi kontrolnu sumu svakog poddirektorijuma (u ovom slučaju, samo koreni direktorijum projekta) i čuva ta stabla u Git repozitorijumu. Git onda kreira komit-objekat koji ima metapodatke i pokazivač na koren stabla projekta tako da može da ponovo kreira taj snimak kada nam bude bio potreban.

Vaš Git repozitorijum sada sadrži pet objekata: jedan blob za sadržaj svakog og tri fajla, jedno stablo koje sadrži listu sadržaja direktorijuma i specificira koja imena fajlova su sačuvana u kom blobu, i jedan komit sa pokazivačem na taj koren stabla i sve komit metapodatke.



Figure 9. Komit i njegovo stablo

Ako napravite neke izmene i komitujete ponovo, sledeći komit čuva pokazivač na komit koji je došao odmah pre njega.



Figure 10. Komitovi i njihovi roditelji

Grana u Gitu je jednostavno mali pokretni pokazivač na jedan od ovih komitova. Podrazumevano ime grane u Gitu je master. Kada počnete da komitujete, daje vam se master grana koja pokazuje na poslednji komit koji ste napravili. Svaki put kada komitujete, kreće se napred automatski.

|  |  |
| --- | --- |
| Note | master grana u Gitu nije posebna grana. Ona je ista kao i svaka druga grana. Jedini razlog zbog kojeg skoro svaki repozitorijum ima takvu granu je to što je komanda git init napravi po podrazumevanim podešavanjima i većina ljudi nema potrebu da je menja. |



Figure 11. Grana i njena istorija komitova

**Pravljenje nove grane**

Šta se dešava kada napravite novu granu? Pa, kada to uradite, stvara vam se novi pokazivač kojim se krećete unaokolo. Recimo da napravite novu granu koju ćete nazvati testing. Ovo se radi komandom git branch:

$ git branch testing

Ovo stvara novi pokazivač na isti komit na kom se trenutno nalazite.



Figure 12. Dve grane pokazuju na istu seriju komitova

Kako Git zna na kojoj se grani trenutno nalazite? Čuva poseban pokazivač koji se zove HEAD. Obratite pažnju na to da je ovo mnogo drugačije od koncepta HEAD-a u drugim VCS-ovima na koje ste možda navikli, kao što su *Subversion* ili *CVS*. U Gitu, ovo je pokazivač na lokalnu granu na kojoj se trenutno nalazite. U ovom slučaju, još uvek ste na master grani. Komandom git branch ste samo *napravili* novu granu — niste prešli na nju.



Figure 13. HEAD pokazuje na granu.

Ovo lako možete da vidite tako što ćete pokrenuti običnu komandu git log koja vam pokazuje gde pokazuju pokazivači na grane. Ova opcija se zove --decorate.

$ git log --oneline --decorate

f30ab (HEAD -> master, testing) add feature #32 - ability to add new formats to the central interface

34ac2 Fixed bug #1328 - stack overflow under certain conditions

98ca9 The initial commit of my project

Vidite da su master i testing grane odmah pored komita f30ab.

**Menjanje grana**

Da biste se prebacili na postojeću granu, treba da pokrenete komandu git checkout. Hajde da se prebacimo na novu granu testing:

$ git checkout testing

Ovo pomera pokazivač HEAD na granu testing.



Figure 14. HEAD pokazuje na trenutnu granu.

Čemu ovo? Pa, hajde da uradimo još jedan komit:

$ vim test.rb

$ git commit -a -m 'made a change'



Figure 15. HEAD grana se pomera napred kada se napravi komit.

Ovo je zanimljivo, jer se sada grana testing pomerila napred, ali grana master još uvek pokazuje na komit na kome ste bili kada ste pokrenuli git checkout da promenite grane. Hajde da se vratimo nazad na granu master:

$ git checkout master



Figure 16. HEAD se pomera kada uradite čekaut.

Ova komanda je uradila dve stvari. Pomerila je pokazivač HEAD nazad na mesto u grani master, i vratila je fajlove u radnom direktorijumu na snimak na koji pokazuje master. Ovo takođe znači da će promene koje pravite odsad pa nadalje divergirati od stare verzije projekta. Bukvalno se vraćate nazad poništavajući ono što ste uradili u testing grani da biste mogli da krenete drugim putem.

|  |  |
| --- | --- |
| Note | Menjanje grana menja fajlove u radnom direktorijumu  Važno je primetiti da će se fajlovi u radnom direktorijumu promeniti kada skačete s grane na granu. Ako se prebacite na stariju granu, radni direktorijum će se vratiti na izgled koji je imao u vreme kada ste komitovali na toj grani. Ako Git nije u stanju da to uradi bez problema, neće vam dati da se prebacite uopšte. |

Hajde da napravimo neke promene i ponovo komitujemo:

$ vim test.rb

$ git commit -a -m 'made other changes'

Sada je istorija vašeg projekta divergirala (pogledate [Divergentna istorija](https://git-scm.com/book/sr/v2/ch00/divergent_history)). Napravili ste granu, prebacili ste se na nju, uradili nešto na njoj, i onda se vratili nazad na glavnu granu i uradili još malo posla. Obe ove promene su izolovane u posebnim granama: možete da skačete s jedne na drugu napred-nazad i da ih spojite kada budete bili spremni. I sve ste to uradili prostim komandama branch, checkout i commit.



Figure 17. Divergentna istorija

Ovo lako možete pogledati i sa git log komandom. Ako pokrenete git log --oneline --decorate --graph --all, odštampaće vam se istorija komitova, pokazujući vam gde se sada nalaze pokazivači na grane i kako je istorija divergirala.

$ git log --oneline --decorate --graph --all

\* c2b9e (HEAD, master) made other changes

| \* 87ab2 (testing) made a change

|/

\* f30ab add feature #32 - ability to add new formats to the

\* 34ac2 fixed bug #1328 - stack overflow under certain conditions

\* 98ca9 initial commit of my project

Pošto je grana u Gitu zapravo jednostavan fajl koji sadrži 40 karaktera SHA-1 čeksume na koju pokazuje komit, pravljenje i uništavanje grana je jeftino. Napraviti novu granu je brzo i jednostavno koliko i upisivanje 41 bajta u fajl (40 slova i nova linija).

Ovo je u oštrom kontrastu sa načinom na koji radi većina starijih alata za grananje u VCS-ovima, u kojima je potrebno kopirati sve fajlove projekta u drugi direktorijum. Ovo može potrajati nekoliko sekundi ili čak minuta, u zavisnosti od veličine projekta, dok je u Gitu ovakav proces uvek trenutan. Takođe, pošto beležimo roditelje kada komitujemo, pronalaženje odgovarajuće baze za spajanje se automatski odradi umesto da se mi bavimo time i u opštem slučaju je taj proces veoma jednostavan. Ove osobine podstiču programere da često prave i koriste grane.

Hajde da pogledamo zašto to treba da radite.

# 3.2 Grananje u Gitu - Osnove grananja i spajanja

## Osnove grananja i spajanja

Hajde da prođemo kroz jednostavan primer grananja i spajanja sa tokom rada kakav se često javlja u realnom svetu. Pratićete ove instrukcije:

1. Radite na veb-sajtu.
2. Napravite granu za novu priču na kojoj radite.
3. Radite nešto na toj grani.

U ovom trenutku, dobijate poziv da postoji kritičan problem koji morate da rešite istog trenutka. Uradićete sledeće:

1. Prebacite se na granu za produkciju.
2. Napravite novu granu na kojoj ćete dodati kôd koji rešava problem.
3. Kada ga testirate, spojite granu sa rešenjem problema, i gurnete na server.
4. Vratite se nazad na priču na kojoj ste radili.

### Osnove grananja

Prvo, pretpostavimo da radite na projektu na kome već imate nekoliko komitova.



Figure 18. Jednostavna istorija komitova.

Odlučili ste da ćete raditi na problemu #53 koji se nalazi na kom god sistemu za praćenje problema koristi vaša kompanija. Da biste napravili granu i skočili na nju istovremeno, možete da pokrenete komandu git checkout sa opcijom -b:

$ git checkout -b iss53

Switched to a new branch "iss53"

Ovo je skraćenica za:

$ git branch iss53

$ git checkout iss53



Figure 19. Kreiranje novog pokazivača na granu.

Radite neke stvari na svom veb-sajtu i obavite neke komitove. Dok radite to, grana iss53 se kreće napred, jer je ona čekautovana (odnosno, HEAD pokazuje na nju).

$ vim index.html

$ git commit -a -m 'added a new footer [issue 53]'



Figure 20. Grana iss53 se pomerila napred u skladu s poslom koji ste obavili.

Sada dobijate poziv da postoji problem sa vebsajtom, i morate odmah da ga popravite. Sa Gitom, ne morate da rešavate problem zajedno sa iss53 promenama koje ste napravili, i ne morate da ulažete mnogo truda u to da povratite te promene pre nego što možete da krenete da radite na rešavanju novonastalog problema. Sve što treba da uradite jeste da se prebacite na master granu.

Ipak, pre nego što to uradite, obratite pažnju na to da vam, ako vaš radni direktorijum ima nekomitovane promene koje su u konfliktu sa granom koju čekautujete, Git neće dozvoliti da promenite granu. Najbolje da radno stanje bude čisto kada pravite skok između grana. Postoje načini da se ovo zaobiđe (naime, skrivanje i komitovanje ispravki) koje ćemo obraditi kasnije, u [[git\_stashing]](https://git-scm.com/book/sr/v2/ch00/git_stashing). Zasad, pretpostavimo da ste komitovali sve promene, tako da možete da se vratite na master granu:

$ git checkout master

Switched to branch 'master'

U ovom trenutku, radni direktorijum vašeg projketa izgleda isto onako kako je izgledao pre nego što se počeli da radite na problemu #53, i možete da se koncentrišete na hitni slučaj. Ovo je bitna stvar koju treba zapamtiti: kada menjate grane, Git resetuje radni direktorijum da izgleda onako kako je izgledao kada ste poslednji put komitovali na toj grani. Dodaje, briše i modifikuje fajlove automatski da bi se postarao da vaša radna kopija izgleda tačno onako kako je izgledala na grani kada ste uradili poslednji komit.

Sledeće, treba da rešite hitan problem. Napravićemo hotfix granu na kojoj ćemo raditi dok to ne bude sređeno.

$ git checkout -b hotfix

Switched to a new branch 'hotfix'

$ vim index.html

$ git commit -a -m 'fixed the broken email address'

[hotfix 1fb7853] fixed the broken email address

1 file changed, 2 insertions(+)



Figure 21. hotfix grana bazirana na master-u.

Možete da testirate ono što ste uradili, da budete sigurni da je problem rešen, i da spojite to nazad sa granom master. Ovo možete da uradite pomoću komande git merge.

$ git checkout master

$ git merge hotfix

Updating f42c576..3a0874c

Fast-forward

index.html | 2 ++

1 file changed, 2 insertions(+)

Primetićete izraz Fast-forward (motanje unapred) koji se pojavio na izlazu prilikom spajanja. Pošto se komit C4 na koji pokazuje grana hotfix u koju ste se spojili nalazio direktno ispred od komita C2 na kom ste bili, Git jednostavno pomera pokazivač napred. Da parafraziramo, kada probate da spojite jedan komit sa komitom do kog se može stići prateći istoriju prvog komita, Git pojednostavljuje stvari tako što samo pomeri pokazivač napred jer nema divergencije sa kojom bi se spajao — zato se ovo zove motanje unapred.

Promena se sada nalazi u snimku komita na koji pokazuje master grana.



Figure 22. master se premotao unapred do hotfix

Pošto je super-važan problem rešen, vreme je da se vratite na ono što ste radili pre nego što ste bili prekinuti. Ipak, prvo ćete obrisati hotfix granu, jer vam više nije potrebna — grana master pokazuje na isto mesto. Možete je obrisati pomoću opcije -d uz komandu git branch:

$ git branch -d hotfix

Deleted branch hotfix (3a0874c).

Sada se vraćate na tiket #53 i nastavljate sa radom

$ git checkout iss53

Switched to branch "iss53"

$ vim index.html

$ git commit -a -m 'finished the new footer [issue 53]'

[iss53 ad82d7a] finished the new footer [issue 53]

1 file changed, 1 insertion(+)



Figure 23. Rad se nastavlja na iss53

Ovde treba napomenuti da posao koji ste odradili u grani hotfix nije sadržan u fajlovima na grani iss53. Ako vam je on potreban, možete da spojite granu master u granu iss53 pokretanjem komande git merge master, ili možete da sačekate sa integrisanjem tih promena dok kasnije ne odlučite da povučete granu iss53 nazad u master.

### Osnove spajanja

Pretpostavimo da ste odlučili da je rad na problemu #53 gotov i da je kod spreman da se spoji sa mastergranom. Da biste uradili to, treba da pripojite granu iss53 grani master, kao što ste ranije spojili hotfix. Sve što treba da uradite jeste da čekautujete granu u koju želite da se spojite i da onda pokrenete git merge komandu:

$ git checkout master

Switched to branch 'master'

$ git merge iss53

Merge made by the 'recursive' strategy.

index.html | 1 +

1 file changed, 1 insertion(+)

Ovo izgleda malo drugačije od spajanja hotfix od ranije. U ovom slučaju, istorija je divergirala na nekoj ranijoj tački. Pošto komit na grani na kojoj se nalazite nije direktan predak grane u koju se spajate, Git mora uradi neki posao. U ovom slučaju, Git radi jednostavan trostruki spoj, koristeći dva snimka na koje pokazuju vrhovi grana i njihovog zajedničkog pretka.



Figure 24. Tri snimka koji se koriste u tipičnom spoju

Umesto da samo pomeri pokazivač na granu napred, Git pravi novi snimak koji je rezultat ovog trostrukog spoja i automatski pravi novi komit koji pokazuje na njega. Ovo se naziva spojni komit (merge commit), i poseban je jedino u tom smislu što ima više od jednog roditelja.



Figure 25. Spojni komit

Vredi napomenuti da je Git taj koji odlučuje koji zajednički predak je najbolje iskoristiti kao bazu za spoj; ovo je drugačije od starijih alata kao što su CVS i Subversion (pre verzije 1.5), gde programer koji radi spoj mora sam da odgonetne koji čvor je najbolja baza. Ovo čini spajanje mnogo jednostavnijim u odnosu na ostale sisteme.

Sada kada je ceo rad spojen, više nema potrebe za iss53 granom. Možete da zatvorite tiket u vašem sistemu za praćenje i da obrišete granu:

$ git branch -d iss53

### Osnovni konflikti pri spoju

S vremena na vreme, ovaj proces neće teći ovako glatko. Ako ste promenili isti deo istog fajla u dvema različitim granama koje spajate u jednu, Git neće moći da ih spoji kako valja. Ako je fiks za tiket #53 modifikovao isti deo kao i hotfix, dobićete konflikt pri spoju koji izgleda ovako nekako:

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

Git nije automatski napravio novi spojni komit. Pauzirao je proces dok vi ne rešite konflikt. Ako želite da vidite koji fajlovi nisu spojeni bilo kada nakon konflikta pri spoju, treba da pokrenete git status:

$ git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: index.html

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

Sve što ima konflikt pri spoju a nije rešeno izlistano je pod nespojeno (unmerged). Git dodaje standardne markere za rešavanje konflikta u fajlove koji imaju konflikt, da biste mogli da ih otvorite i ručno rešite konflikte. Fajl sadrži sekciju koja izgleda nekako ovako:

<<<<<<< HEAD:index.html

<div id="footer">contact : email.support@github.com</div>

=======

<div id="footer">

please contact us at support@github.com

</div>

>>>>>>> iss53:index.html

Ovo znači da je verzija u HEAD (vaša master grana, jer to ste čekautovali kada ste pokrenuli komandu merge) na vrhu tog bloka (sve iznad =======), dok je verzija iz grane iss53 prikazana u donjem delu. Da biste rešili konflikt, morate ili da izaberete jednu stranu ili drugu ili da ručno spojite sadržinu fajla. Na primer, ovaj konflikt se može rešiti tako što ćete ceo gornji blok zameniti ovime:

<div id="footer">

please contact us at email.support@github.com

</div>

Ovo rešenje ima pomalo iz obe sekcije, a linije sa ======= i >>>>>>> su potpuno obrisane. Nakon što rešite svaku od ovakvih sekcija u svakom fajlu sa konfliktom, pokrenite git add na svaki fajl da biste ga označili kao razrešen. Stejdžovanje fajla obeležava konflikt kao razrešen u Gitu.

Ako želite da koristite grafički alat da rešite ovakve probleme, možete da pokrenete komandu git mergetool, koja pokreće odgovarajući vizuelni alat koji vas vodi kroz konflikte:

$ git mergetool

This message is displayed because 'merge.tool' is not configured.

See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:

opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge

Merging:

index.html

Normal merge conflict for 'index.html':

{local}: modified file

{remote}: modified file

Hit return to start merge resolution tool (opendiff):

Ako hoćete da koristite neki drugi alat za spajanje koji nije podrazumevani (Git je izabrao opendiff u ovom slučaju jer je komanda pokrenuta na Meku), videćete sve podržane alate izlistane na vrhu posle "one of the following tools". Samo ukucajte ime alata koji biste radije koristili.

|  |  |
| --- | --- |
| Note | Ako su vam potrebni napredniji alati da biste rešili nezgodnije konflikte pri spoju, pogledaćemo neke od njih u [Advanced Merging](https://git-scm.com/book/sr/v2/ch00/_advanced_merging). |

Kada izađete iz alata za rešavanje konflikta pri spoju, Git će vas pitati da li je spoj bio uspešan. Ako kažete skripti da jeste, fajl će biti stejdžovan i označen kao razrešen. Možete da pokrenete git statusda biste se uverili da su svi konflikti razrešeni:

$ git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: index.html

Ako ste zadovoljni time, i ako ste potvrdili da je sve što je imalo konflikte sada na stejdžu, možete da ukucate git commit da finalizirate spojni komit. Podrazumevana komit poruka izgleda nekako ovako:

Merge branch 'iss53'

Conflicts:

index.html

#

# It looks like you may be committing a merge.

# If this is not correct, please remove the file

# .git/MERGE\_HEAD

# and try again.

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# All conflicts fixed but you are still merging.

#

# Changes to be committed:

# modified: index.html

#

Možete da modifikujete tu poruku sa detaljima o tome kako ste razrešili spoj ako mislite da će to biti korisno drugima koji budu gledali ovaj spoj u budućnosti — zašto ste uradili to što ste uradili, ako nije očigledno.

# 3.3 Grananje u Gitu - Upravljanje granama

## Upravljanje granama

Sada kada ste napravili, spojili i obrisali neke grane, hajde da pogledamo neke alate za upravljanje granama koje će postati korisni kada počnete da stalno koristite grane.

Komanda git branch ima i druge namene pored toga što može da stvara i briše grane. Ako je pokrenete bez opcija, dobićete jednostavnu listu sa trenutnim granama:

$ git branch

iss53

\* master

testing

Obratite pažnju na zvezdicu koja se javlja ispred imena grane master: to je indikator da je to grana koju ste trenutno čekautovali (tj. grana na koju pokazuje pokazivač HEAD). Ovo znači da, ako sada komitujete, grana master će biti pomerena napred novim stvarima koje ste napravili. Da biste videli poslednji komit na svakoj grani, možete pokrenuti git branch -v:

$ git branch -v

iss53 93b412c fix javascript issue

\* master 7a98805 Merge branch 'iss53'

testing 782fd34 add scott to the author list in the readmes

Korisne --merged i --no-merged opcije mogu da filtriraju listu na grane koje već jeste ili još uvek niste spojili u granu na kojoj se trenutno nalazite. Da biste videli koje grane su već spojene u granu na kojoj ste sada, treba da pokrenete git branch --merged:

$ git branch --merged

iss53

\* master

Pošto ste se već spojili sa iss53 ranije, ta grana se nalazi na listi. Grane na ovoj listi koje nemaju \*ispred sebe su u opštem slučaju spremne da budu obrisane sa git branch -d; već ste ih pripojili drugoj grani, tako da nećete ništa da izgubite.

Da biste videli sve grane koje sadrže rad koji još uvek niste spojili, pokrenite komandu git branch --no-merged:

$ git branch --no-merged

testing

Ovo prikazuje vašu drugu granu. Zato što se na njoj nalazi rad koji još uvek nije spojen, pokušaj brisanja ove grane sa git branch -d neće uspeti:

$ git branch -d testing

error: The branch 'testing' is not fully merged.

If you are sure you want to delete it, run 'git branch -D testing'.

Ako stvarno želite da obrišete tu granu i izgubite sav taj deo koda, možete da forsirate to sa -D, kao što predlaže pomoćna poruka.

# 3.4 Grananje u Gitu - Tokovi rada sa grananjem

## Tokovi rada sa grananjem

Sada kada znate osnove o granama i njihovom spajanju, šta možete i šta treba da radite sa njima? U ovom odeljku ćemo se pozabaviti nekim najopštijim tokovima rada koje ovakav način grananja u Gitu omogućava, kako biste mogli da odlučite da li biste voleli da takav način rada uključite u svoj lični.

### Dugotrajne grane

Pošto Git koristi jednostavan trostruki spoj, spajanje jedne grane u drugu više puta tokom dugog vremenskog perioda generalno nije problem uraditi. Ovo znači da možete da imate nekoliko grana koje su uvek otvorene i koje se koristite za različite etape ciklusa razvića; možete regularno da spajate neke od njih u druge.

Mnogi Git developeri koriste ovakav način rada, odnosno imaju kôd koji je u potpunosti stabilan u mastergrani i to je verovatno jedini kôd koji se dosad našao u zvaničnim verzijama softvera koji se razvija na projektu. Imaju druge paralelne grane koje se zovu develop i next na kojima rade i gde se testira stabilnost za buduće verzije — tako kôd ne mora uvek da bude stabilan, ali kada dođe u stabilno stanje, može da se spoji sa master granom. Koristi se da bi se sa njim spajale grane koje nemaju dugi životni vek poput one iss53 koju smo obradili ranije; tu se proverava da li su svi problemi rešeni i da li nisu nastali neki novi.

U stvarnosti, ovde se radi o pokazivačima koji kližu duž linije komitova koji se prave. Stabilne grane su niže niz liniju u istoriji komitova, a komitovi koji sadrže kod sa diskutabilnom funkcionalnošću nalaze se pri vrhu.



Figure 26. Linearni pogled na grananje koje se oslanja na tehniku progresivne stabilnosti.

U opštem slučaju je jednostavnije razmišljati o njima kao o radnim silosima, gde skupovi komitova maturiraju ka stabilnijem silosu kada se u potpunosti testiraju.



Figure 27. "Silos" pogled na grananje koje se oslanja na tehniku progresivne stablnosti.

Po uzoru na ovo, mogu se napraviti slični tokovi rada sa više nivoa stabilnosti. Neki veći projekti imaju i granu proposed ili pu (proposed updates) koja ima integrisane grane koje još uvek nisu spremne da se premeste u next ili master. Ideja je da grane predstavljaju razne verzije stabilnosti; kada dosegnu stabilniji nivo, utapaju se u granu na višem nivou. Opet, postojanje više dugotrajnih grana nije neophodno, ali često može da pomogne, pogotovo kada se radi sa veoma velikim ili složenim projektima.

### Tematske grane

Tematske grane su, međutim, korisne za projekete bilo koje veličine. Tematska grana je grana kratkog životnog veka koju kreirate i koristite samo za jednu odliku rada. Ovo je nešto što verovatno nikad ranije niste radili sa VCS-om zato što je generalno previše skupo da kreirate i upravljate granama. Ali u Gitu je prirodno da se grane kreiraju, da se radi na njima, da se spoje a da se zatim obrišu nekoliko puta tokom dana.

Videli ste ovo u prehodnom odeljku sa iss53 i hotfix granama koje ste napravili. Odradili ste nekoliko komitova sa njima i obrisali ste ih čim ste ih spojili u glavnu granu. Ova tehnika omogućava da brzo i potpuno menjate kontekst u kome radite — pošto je rad podeljen u silose gde sve promene u jednoj grani imaju veze samo sa tom temom, lakše da je da videti šta se desilo sa kodom tokom kasnijeg pregleda. Promene možete da zadržite tu minutima, danima ili mesecima, i da ih spojite kada budu spremne, bez obzira na redosled kojim su kreirani i kojim je rađeno na njima.

Razmotrite primer gde radite neki posao (na master-u), odgranate se na tiket (iss91), radite na tome neko vreme, odgranate se na još jednu granu gde probate da rešite istu stvar na drugi način (iss91v2), vraćate se na master granu gde radite još neko vreme, a onda pravite granu gde radite neke stvari za koje niste sigurni da su dobra ideja (glupa-ideja). Vaša istorija komitova će izgledati ovako nekako:



Figure 28. Više tematskih grana

Sada odlučujete da vam se drugo rešenje problema sviđa više (iss91v2), a kada glupa-ideja dopre do vaših saradnika, ispostavlja da se da je ideja zapravo bila genijalna. Odbacujete originalnu iss91 granu (gubite komitove c5 i c6) i spajate druge dve. Istorija sada izgleda ovako:



Figure 29. Istorija nakon spanja grana glupa-idea i iss91v2.

Detaljnije ćemo obraditi razne druge mogućnosti kako se mogu realizovati tokovi rada za Git projekat u [Distribuirani Git](https://git-scm.com/book/sr/v2/ch00/_distributed_git), tako da obavezno pročitajte i to poglavlje pre nego što se odlučite kakvu ćete šemu grananja da koristite.

Bitno je zapamtiti da su grane potpuno lokalne dok radite sve ovo. Dok se granate i spajate, sve što radite dešava se samo u vašem Git repozitorijumu - ne postoji nikakav vid komunikacije sa serverom.

# 3.5 Grananje u Gitu - Udaljene grane

## Udaljene grane

Udaljene grane su reference (pokazivači) u vašim udaljenim repozitorijumima, uključujući grane, tagove, i tako dalje. Možete da pogledate celu listu udaljenih referenci eksplicitno sa git ls-remote <remote>, ili git remote show <remote> za udaljene grane kao i za više informacija. Ipak, češći način je korišćenje udaljenih praćenih grana (remote-tracking branches).

Udaljene praćene grane su reference na stanja udaljenih grana. One su lokalne reference koje ne možete da pomerite; one se pomeraju automatski kada radite bilo kakav vid komunikacije preko mreže. Udaljene praćene grane rade kao obeleživači koji treba da vas podsete gde su grane u vašim repozitorijumima na daljinu bile kada ste se poslednji put povezali sa njima.

Uzimaju formu (remote)/(branch). Na primer, ako želite da pogledate kako je master grana na origin udaljenom repozitorijumu izgledala poslednji put kada ste stupili u komunikaciju sa njom, proverili biste origin/master granu. Ako ste radili na tiketu sa partnerom koji je slao izmene na grani iss53, vi imate svoju lokalnu granu iss53; ali grana na serveru pokazuje na komit origin/iss53.

Ovo je možda malo zbunjujuće, zato hajde da pogledamo primer. Recimo da imate Git server na svojoj mreži na git.ourcompany.com. Ako klonirate odavde, Gitova komanda clone će to automatski nazvati origin, povući će sve podatke, napraviće pokazivač na mesto gde je master grana i nazvati je origin/master lokalno. Git vam takođe daje vlastitu master granu koja počinje na istom mestu kao i origin-ova master grana, tako da imate odakle da krenete da radite.

|  |  |
| --- | --- |
| Note | origin nije posebna grana  Kao što grana sa imenom master nema nikakvo posebno značenje na Gitu, tako nema ni origin. Dok je master podrazumevano ime za početnu granu koja se dobija sa git init i to je jedini razlog zbog koga se koristi, origin je podrazumevano ime za rimout kada pokrenete git clone. Ako pokrenete git clone -o booyah, onda će booyah/master biti podrazumevana grana na daljinu. |



Figure 30. Server i lokalni repozitorijumi posle kloniranja.

Ako radite nešto na lokalnoj master grani, i, u međuvremenu, neko drugi pošalje nešto na git.ourcompany.com i apdejtuje master granu, onda se vaše istorije kreću napred drugačijim tokom. Takođe, sve dok ne stupite u kontakt sa serverom, origin/master pokazivač se neće pomeriti.



Figure 31. Lokalni i udaljeni rad mogu da divergiraju.

Da biste sinhronizovali vaš rad, treba da pokrenete git fetch origin komandu. Ova komanda će da potraži server koji je origin (u ovom slučaju je to git.ourcompany.com), pribavi sve podatke odatle koje još uvek nemate, i apdejtuje lokalnu bazu podataka, pomerajući vaš origin/master pokazivač na novu aktuelnu poziciju.



Figure 32. git fetch apdejtuje udaljene reference

Da bismo demonstrirali situaciju sa nekoliko udaljenih servera i objasnili kako izgledaju udaljene grane za te udaljene projekte, hajde da pretpostavimo da imate još jedan interni Git server koji koristi samo jedan od vaših sprint timova. Ovaj server se nalazi na git.team1.ourcompany.com. Možete da ga dodate kao novu udaljenu referencu u projekat na kom trenutno radite tako što ćete pokrenuti git remote add kao što smo već objasnili u [Osnove Gita](https://git-scm.com/book/sr/v2/ch00/_git_basics_chapter). Ovu udaljenu granu nazovite teamone, što će predstavljati kratko ime za taj ceo URL.



Figure 33. Dodavanje još jednog udaljenog servera

Sada možete da pokrenete git fetch teamone da pribavite sve što server na daljinu teamone ima a vi još uvek nemate. Pošto taj server ima podskup podataka koji origin server ima trenutno, Git ne pribavlja podatke već postavlja udaljenu praćenu granu koja se zove teamone/master koja pokazuje na komit koji teamone ima na svojoj master grani.



Figure 34. Udaljena praćena grana za teamone/master

### Guranje

Kada želite da granu podelite s ostatkom sveta, morate da je gurnete ka rimoutu kome imate pristup. Vaše lokalne grane se neće automatski sinhronizovati sa rimoutovima kojima pišete - morate eksplicitno da pošaljete ("gurnete") grane koje treba da podelite. Na taj način, možete da koristite privatne grane za ono što ne želite da podelite sa ostalima, i da šaljete samo tematske grane na kojima kolaborirate.

Ako imate granu koja se zove serverfix na kojoj želite da radite sa ostalima, možete da je gurnete naviše na isti način na koji ste gurnuli i prvu granu. Pokrenite git push <remote> <branch>:

$ git push origin serverfix

Counting objects: 24, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (15/15), done.

Writing objects: 100% (24/24), 1.91 KiB | 0 bytes/s, done.

Total 24 (delta 2), reused 0 (delta 0)

To https://github.com/schacon/simplegit

\* [new branch] serverfix -> serverfix

Ovo je mala prečica. Git automatski proširuje ime grane serverfix u refs/heads/serverfix:refs/heads/serverfix, što znači "uzmi moju lokalnu serverfix granu i pošalji je kao apdejt udaljenoj servfix grani". Pogledaćemo deo refs/heads detaljnije u [Git iznutra](https://git-scm.com/book/sr/v2/ch00/_git_internals), ali u suštini možete da ga izostavite. Možete da uradite i git push origin serverfix:serverfix, što radi istu stvar — kaže "Uzmi moj serverfix i napravi ga da bude udaljen serverfix". Možete da koristite ovaj format da gurnete lokalnu granu na udaljenu granu koja se naziva drugačije. Ako ne želite da se zove serverfix na rimoutu, možete da pokrenete git push origin serverfix:awesomebranch da gurnete lokalnu serverfix granu na awesomebranch granu na udaljenom projektu.

|  |  |
| --- | --- |
| Note | Nemojte da kucate šifru svaki put  Ako koristite HTTPS URL za guranje, Git server će vas pitati za korisničko ime i šifru radi overe autentičnosti. Po podrazumevanim podešavanjima, pitaće vas u terminalu za ovu informaciju kako bi server znao da li vam je dozvoljeno da obavite operaciju push.  Ako ne želite da kucate ove podatke svaki put kada želite to, možete da podesite "akreditivni keš". Najjednostavniji način je da ih jednostavno zadržite u memoriji na nekoliko minuta, to lako možete podesiti pokretanjem git config --global credential.helper cache.  Za više informacija o raznim akreditivnim opcijama za keširanje koje su dostupne, pogledajte [Credential Storage](https://git-scm.com/book/sr/v2/ch00/_credential_caching). |

Sledeći put kada jedan od vaših kolaboratora pribavi sadržaj sa servera, dobiće referencu na mesto gde je serverova verzija serverfix-a pod udaljenom granom origin/serverfix:

$ git fetch origin

remote: Counting objects: 7, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 3 (delta 0)

Unpacking objects: 100% (3/3), done.

From https://github.com/schacon/simplegit

\* [new branch] serverfix -> origin/serverfix

Bitno je da primetite da kada uradite pribavljanje koje dovlači nove udaljene grane sa praćenjem, nemate automatski njihove lokalne kopije nad kojima možete da radite. Drugim rečima, u ovom slučaju, nemate novu serverfix granu - imate samo origin/serverfix pokazivač koji ne možete da menjate.

Da biste spojili ovaj sadržaj sa granom na kojoj trenutno radite, možete da pokrenete git merge origin/serverfix. Ako želite svoju ličnu serverfix granu na kojoj možete da radite, možete da je bazirate na udaljenoj praćenoj grani:

$ git checkout -b serverfix origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

Switched to a new branch 'serverfix'

Ovo vam daje lokalnu granu na kojoj možete da radite koja počinje na mestu gde je origin/serverfix.

### Grane pratilje

Čekautovanje lokalne grane sa udaljene praćene grane automatski kreira nešto što se zove "grana pratilja" (ili ponekad "uzvodna grana"). Grane pratilje su lokalne grane koje imaju direktnu vezu sa udaljenom granom. Ako ste na grani pratilji i ukucate git pull, Git automatski zna sa kojeg servera treba da pribavi podatke i sa kojom granom treba da se spoji.

Kada klonirate repozitorijum, u opštem slučaju se automatski kreira master grana koja prati origin/master. Međutim, možete da postavite i druge grane za praćenje ukoliko želite — one koje prate grane na drugim rimoutovima, ili ne prate master granu. Jednostavan slučaj je primer koji ste upravo videli, pokretanje git checkout -b [branch] [remotename]/[branch]. Ova operacija je toliko česta da postoji skraćenica --track:

$ git checkout --track origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

Switched to a new branch 'serverfix'

Da biste podesili lokalnu granu sa drugim imenom od onog koje koristi udaljena grana, možete lako da iskoristite prvu verziju sa drugim imenom lokalne grane:

$ git checkout -b sf origin/serverfix

Branch sf set up to track remote branch serverfix from origin.

Switched to a new branch 'sf'

Sada će vaša lokalna grana sf automatski povlačiti sa origin/serverfix-a.

Ako već imate lokalnu granu i želite da je podesite na udaljenu granu koju ste upravo povukli, ili želite da promenite uzvodnu granu koju pratite, možete da koristite -u ili --set-upstream-to opciju uz komandu git branch da biste joj eksplicitno zadali u bilo kom trenutku.

$ git branch -u origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

|  |  |
| --- | --- |
| Note | upstream prečica  Kada imate podešenu granu pratilju, možete da je referencirate koristeći prečice @{upstream} ili @{u}. Znači ako ste na master grani koja prati origin/master, možete da kažete nešto kao git merge @{u} umesto git merge origin/master ukoliko želite. |

Ako želite da vidite koje ste grane pratilje podesili, iskoristite -vv opciju uz git branch. Ovo će izlistati vaše lokalne grane sa više informacija, uključujući i to šta svaka od grana prati i da li je lokalna grana ispred, iza ili oba.

$ git branch -vv

iss53 7e424c3 [origin/iss53: ahead 2] forgot the brackets

master 1ae2a45 [origin/master] deploying index fix

\* serverfix f8674d9 [teamone/server-fix-good: ahead 3, behind 1] this should do it

testing 5ea463a trying something new

Dakle, ovde možemo da vidimo da naša iss53 grana prati origin/iss53 i da je "ispred" za dva, što znači da imamo dva komita lokalno koja nisu gurnuta na server. Možemo da vidimo i to da naša mastergrana prati origin/master i da je to aktuelna verzija sa servera. Dalje, vidimo da naša serverfixgrana prati serverfix-fix-good granu na teamone serveru i da je ispred za tri i iza za jedan, što znači da postoji jedan komit na serveru sa kojim se još nismo spojili, ali i da postoje tri komita lokalno koja još nismo gurnuli. Konačno, vidimo da naša testing grana ne prati nijednu udaljenu granu.

Važno je zapaziti da su ovi brojevi relativni u odnosu na trenutak kada ste ste poslednji put pribavili podatke sa servera. Ova komanda ne stupa u vezu sa serverom, govori vam samo o onome što lokalno ima, keširano sa servera. Ako želite aktuelne ahead i behind brojeve, moraćete da pribavite podatke sa svim rimoutova odmah pre nego što pokrenete ovu komandu. To možete uraditi na sledeći način:

$ git fetch --all; git branch -vv

### Povlačenje

Dok će git fetch komanda pribaviti sve promene na serveru koje još uvek nemate, neće vam uopšte modifikovati radni direktorijum. Samo će prikupiti podatke i dozvoliće vam da ih sami spojite. Ipak, postoji komanda koja se zove git pull koja je se u većini slučajeva može protumačiti kao git fetch za kojom sledi git merge. Ako imate granu pratilju podešenu kao što je demonstrirano u prethodnom odeljku, bilo da ste je eksplicitno podesili ili ste je dobili uz clone ili checkout komandu, git pull će pogledati koji server i granu vaša grana trenutno prati, pribaviće podatke sa tog servera i onda će probati da se spoji u tu granu pratilju.

U opštem slučaju je bolje da jednostavno koristite fetch i merge komande eksplicitno, jer komanda git pull zna da bude zbunjujuća.

### Brisanje udaljenih grana

Pretpostavimo da ste završili sa udaljenom granom — recimo da ste vi i vaši kolaboratori završili rad na nekom delu koda i spojili ste je sa udaljenom master granom (ili s kojom god granom spajate stabilan kod). Možete da obrišete udaljenu granu koristeći --delete opciju uz git push. Ako želite da obrišete vašu serverfix granu sa servera, pokrenite sledeće:

$ git push origin --delete serverfix

To https://github.com/schacon/simplegit

- [deleted] serverfix

U suštini, ovo samo uklanja pokazivač sa servera. Gitov server će u opštem slučaju zadržati podatke tamo neko vreme dok se ne pokrene garbage collector, zato će obnova podataka biti laka ako dođe do slučajnog brisanja grane.

3.6 Grananje u Gitu - Rebaziranje

**Rebaziranje**

U Gitu, postoje dva glavna načina za integraciju promena iz jedne grane u drugu: merge i rebase. U ovom odeljku ćete naučiti šta je rebaziranje, kako se radi, zašto je to prilično dobra stvarčica, i kada treba a kada ne treba da je koristite.

**Osnovno rebaziranje**

Ako se vratite na raniji primer iz [Osnove spajanja](https://git-scm.com/book/sr/v2/ch00/_basic_merging), videćete da ste divergirali svoj rad i napravili komitove na različitim granama.



Figure 35. Jednostavna divergentna istorija

Najlakši način da integrišete grane, što smo već pokrili, jeste pomoću komande merge. Ona će uraditi trostruki spoj između dva poslednja snimka sa grana (C3 i C4) i njihovog najskorašnjijeg zajedničkog pretka (C2), stvarajući novi snimak (i komit).



Figure 36. Spajanje radi integracije divergentne istorije rada.

Međutim, postoji još jedan način: možete da uzmete zakrpu promene koja je predstavljena u C4 i da je ponovo primenite preko C3. U Gitu se ovo zove *rebaziranje*. Pomoću rebase komande, možete da uzmete sve promene koje su komitovane na jednu granu i da ih ponovite na drugoj.

U ovom primeru, pokrenuli biste sledeće:

$ git checkout experiment

$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: added staged command

Radi tako što ode na zajedničkog pretka dveju grana (onih na kojoj ste sad i one sa kojom rebazirate), uzima razliku koja je stvorena komitovima na grani na kojoj se nalazite, pamti te razlike u fajlovima koje čuva samo trenutno, resetuje trenutnu granu prema istom komitu na kome je grana ka kojoj rebazirate, i konačno primenjuje svaku promenu.



Figure 37. Rebaziranje promena koje su predstavljene u C4 i C3.

U ovom trenutku, možete da se vratite nazad na master granu i da uradite spoj tehnikom motanja unapred.

$ git checkout master

$ git merge experiment



Figure 38. Motanje master grane unapred.

Sada, snimak na koji pokazuje C4' je potpuno isti kao i onaj na koji je pokazivao C5 u primeru spoja. Nema razlike u krajnjem proizvodu integracije, ali rebaziranjem se postiže čistija istorija. Ako istražite log rebazirane grane, izgleda kao linearna istorija: izgleda kao da se sav rad odvijao serijski, iako su se stvari zapravo odvijale paralelno.

Ovo ćete često raditi da biste se postarali da se komitovi primene na udaljenu granu kako valja — možda u projektu kojem želite da kontribuirate ali ne želite da ga održavate. U ovom slučaju, radili biste svoj posao na jednoj grani i onda biste rebazirali svoj rad u origin/master kada ste spremni da pošaljete svoje zakrpe glavnom projektu. Na ovaj način, održavalac ne mora da radi nikakvu integraciju — samo treba da premota unapred.

Obratite pažnju na to da je snimak na koji pokazuje konačni komit, bilo da je to poslednji od rebaziranih komitova za rebaziranje ili konačni spojni komit nakon spoja, jedan te isti — samo se istorija razlikuje. Rebaziranje ponavlja promene iz jedne linije rada u drugu i to redom kojim su pravljene, dok spajanje uzima krajnje tačke i meša ih u jedno.

**Interesantniji slučajevi rebaziranja**

Možete da rebazirate nad nečime što nije odredišna grana za rebaziranje. Uzmite istoriju kao [Istorija sa tematskom granom izvedenom iz druge tematske grane.](https://git-scm.com/book/sr/v2/ch00/rbdiag_e), na primer. Odgranali ste se tematskom granom (server) da biste dodali neku funkcionalost u vezi sa serverom u projekat, i napravili komit. Onda se odgranali od toga da biste napravili neke promene sa strane klijenta (client) i komitovali nekoliko puta. Konačno, vratili ste se na server granu i napravili još nekoliko komitova.



Figure 39. Istorija sa tematskom granom izvedenom iz druge tematske grane.

Pretpostavimo da ste odlučili da želite da spojite promene na grani client sa glavnom granom, ali želite da odložite promene na server grani dok ih bolje ne testirate. Možete da uzmete promene sa clientgrane koje nisu na serveru (C8 i C9) i da ih ponovite na master grani koristeći opciju --onto komande git rebase:

$ git rebase --onto master server client

Ovo u suštini kaže, "Proveri granu client, shvati koje zakrpe postoje u odnosu na zajedničkog pretka grana client i server, i onda ih ponovo primeni na master". Malo je složeno, ali rezultat je odličan.



Figure 40. Rebaziranje tematske grane izvedene iz druge tematske grane.

Sada možete da premotate unapred granu master (vidite [Motanje master grane unapred da biste obuhvatili promene sa grane client.](https://git-scm.com/book/sr/v2/ch00/rbdiag_g)):

$ git checkout master

$ git merge client



Figure 41. Motanje master grane unapred da biste obuhvatili promene sa grane client.

Recimo da ste odlučili da povučete i server granu. Možete da rebazirate server granu u master granu a da ne morate da je prvo čekautujete sa git rebase [osnovna-grana] [tematska-grana] — što čekautuje tematsku granu (u ovom slučaju server) i primenjuje pronađene promene na osnovnu granu (master).

$ git rebase master server

Ovo ponavlja promene nađene na grani server na grani master, kao što se vidi na [Rebaziranje servergrane na master grani.](https://git-scm.com/book/sr/v2/ch00/rbdiag_h).



Figure 42. Rebaziranje server grane na master grani.

Onda premotate unapred osnovnu granu (master):

$ git checkout master

$ git merge server

Možete da obrišete grane client i server jer je sav rad obavljen na njima sada integrisan i više vam neće biti potreban, a istorija celokupnog rada će izgledati kao na [Konačna istorija komitova](https://git-scm.com/book/sr/v2/ch00/rbdiag_i):

$ git branch -d client

$ git branch -d server



Figure 43. Konačna istorija komitova

**Opasnosti prilikom rebaziranja**

Ah, ali blaženstvo rebaziranja nije bez mana, što se može sumirati samo jednom rečenicom:

**Ne rebazirajte komitove koji postoje van vašeg repozitorijuma.**

Ako se držite te smernice, sve će biti u redu. Ako ne, ljudi će vas mrzeti, a porodica i prijatelji će vas prezirati.

Kada rebazirate stvari, napuštate postojeće komitove i stvarate nove koji su slični ali drugačiji. Ako gurnete komitove negde i ostali povuku s njih i baziraju svoj rad nad njima, a onda pišete preko tih komitova sa git rebase i gurnete ih ponovo, vaši kolaboratori će morati da ponovo spoje sav svoj rad i onda će nastati haos kada probate da povučete njihov rad nazad ka sebi.

Pogledajmo primer koji pokazuje kako rebaziranje koje ste napravili javnim može da izazove probleme. Pretpostavimo da ste napravili klon sa centralnog servera i onda radili nešto nad time. Istorija komitova izgleda ovako:



Figure 44. Kloniran repozitorijum nad kojim ste obavili neki posao.

Sada, neko drugi uradi još nekoliko stvari i uključi to u spoj, a zatim gurne sve na centralni server. Vi to preuzmete i spojite novu udaljenu granu na ono što ste uradili, pa istorija izgleda nekako ovako:



Figure 45. Preuzimanje drugih komitova i spajanje sa ličnim radom.

Osoba koja je gurnula spojen rad zatim odluči da se vrati nazad i rebazira ono što je odradila; on ili ona uradi git push --force da bi se pisalo preko istorije na serveru. Vi onda preuzmete sa tog servera, dovlačeći nove komitove.



Figure 46. Neko gurne rebazirane komitove, napuštajući komitove nad kojima ste bazirali vaš rad.

Sada ste oboje u neprilici. Ako uradite git pull, napravićete spojni komit koji uključuje obe linije istorije, i repozitorijum će izgledati ovako:



Figure 47. Ponovno spajanje istog rada u novi spojni komit.

Ako pokrenete git log dok vam istorija izgleda ovako, videćete dva komita koji imaju istog autora, vreme i poruku, što zna da zbunjuje. Štaviše, ako gurnete ovu istoriju nazad na server, ponovo ćete uvesti sve te rebazirane komitove na centralni server, što će dalje zbunivati ljude. Pretpostavlja se da drugi programer ne želi da se C4 i C6 nađu u istoriji; zato su uopšte i radi rebaziranje.

**Rebaziranje za rebaziranje**

Ako se ipak nađete u ovakvoj situaciji, Git ima još neke čarolije koje vam mogu pomoći. Ako neko iz tima nasilno gurne promene koje preklope ono nad čime ste vi bazirali svoj rad, izazov koji vam se nameće je da provalite šta je vaše a šta je on dodao.

Ispostavlja se da pored SHA-1 čeksume vezane za komit, Git računa i čeksumu koja je bazirana samo na zakrpi koja je uvedena sa komitom. Ovo se zove "identifikacioni broj zakrpe".

Ako povučete rad koji je prepisan i rebazirate navrh toga sa novim komitovima vašeg partnera, Git često ume sam da provali šta je jedinstveno vaše i da primeni to nazad na vrh nove grane.

Na primer, u prethodnom scenariju, ako umesto spoja kada smo bili kod [Neko gurne rebazirane komitove, napuštajući komitove nad kojima ste bazirali vaš rad.](https://git-scm.com/book/sr/v2/ch00/_pre_merge_rebase_work) pokrenemo git rebase teamone/master, Git će:

* odrediti koji rad je jedinstven za našu granu (C2, C3, C4, C6 i C7),
* odrediti šta nisu spojni komitovi (C2, C3 i C4),
* odrediti šta nije prepisano u odredišnu granu (samo C2 i C3, pošto je C4 ista zakrpa kao i C4') i
* primeniti te komitove na teamone/master granu.

Zato ćemo, umesto rezultata koji vidimo na [Ponovno spajanje istog rada u novi spojni komit.](https://git-scm.com/book/sr/v2/ch00/_merge_rebase_work), dobiti nešto što više podseća na [Rebaziranje na nasilno gurnuto rebaziranje.](https://git-scm.com/book/sr/v2/ch00/_rebase_rebase_work).



Figure 48. Rebaziranje na nasilno gurnuto rebaziranje.

Ovo će da upali samo ako C4 i C4' koje je vaš partner napravio čine skoro identičnu zakrpu. U suprotnom, rebaziranjem neće moći da se ustanovi da je to duplikat i biće dodata još jedna zakrpa koja podseća na C4 (koja verovatno neće uspeti da se čisto primeni, jer će promene već bar delimično biti tamo).

Ovo možete da uprostite i pokretanjem git pull --rebase umesto običnog git pull. Ili možete da uradite to ručno sa git fetch za kojim sledi git rebase teamone/master u ovom slučaju.

Ako koristite git pull i želite da --rebase bude podrazumevana opcija, možete da podesite pull.rebase vrednost iz konfiguracionog fajla na true sa git config --global pull.rebase true.

Ako rebaziranje posmatrate kao način da pospremite i radite sa komitovima pre nego što ih gurnete, i ako samo rebazirate komitove koji nikad nisu bili dostupni javno, sve će biti u redu. Ako rebazirate komitove koji su već gurnuti javno, i ljudi baziraju svoj rad nad tim komitovima, onda ćete se naći u frustrirajućim situacijama i bićete meta prezira svojih saradnika.

Ako vi ili partner u nekom trenutku shvatite da je ovakav sled događaja neophodan, postarajte se da svi znaju da treba da pokrenu git pull --rebase da probaju da učine stvar barem malo jednostavnijom.

**Rebaziranje protiv spajanja**

Sada kada ste videli kako deluje rebaziranje a kako spajanje, možda se pitate šta je bolje. Pre nego što damo odgovor na ovo, hajde da načinimo korak unazad i popričamo malo o tome šta je zapravo istorija.

Jedna tačka gledišta na ovo je to da je istorija komitova vašeg repozitoijuma **zapis onoga što se zapravo dogodilo**. To je istorijski dokument, vredan na svoj način, i ne bi trebalo da se čačka. Iz ovog ugla, menjanje istorije komita je skoro pa bogohuljenje; vi *lažete* o onome što se zapravo dogodilo. Šta onda raditi kada se dogodi serija zbrkanih komitova? Tako su se stvari dogodile, i repozitorijum treba da sačuva to za potomstvo.

Suprotna tačka gledišta je da je istorija komitova **priča o tome kako je projekat napravljen.** Ne biste objavili prvu skicu knjige, a upustvo za održavanje softvera zaslužuje brižljivo redigovanje. Ovaj tabor koristi alate kao što je rebaziranje i filter grane da bi ispričao priču onako kako je najbolje da je pročita budući čitalac.

Sada, što se tiče pitanja da li je spajanje ili rebaziranje bolje: nadamo se da ćete videti da stvari nisu tako jednostavne. Git je moćan alat, dopušta vam da uradite mnoge stvari sa istorijom, ali svaki tim i svaki projekat je drugačiji. Sada kada znate kako obe ove stvari rade, na vama je da odlučite šta je bolje za vašu konkretnu situaciju.

U opštem slučaju, najbolji način da iskoristite prednost oba sveta je da rebazirate lokalne promene koje ste napravili ali još niste podelili pre nego što ih gurnete kako biste očistili istoriju, ali da nikad ne rebazirate ništa što ste negde gurnuli.

5.1 Distribuirani Git - Distribuirani tokovi rada

Sada kada imate podešen udaljen Git repozitorijum kao tačku na kojoj svi programeri dele svoj kôd, i pošto ste upoznati sa osnovnim Git komandama u lokalnom toku rada, vreme je da bacimo pogled na to kako iskoristiti neke od distribuiranih tokova rada u Gitu.

U ovom poglavlju, videćete kako da radite sa Gitom u distribuiranom okruženju kao kontributor i integrator. Tačnije, naučićete kako da uspešno doprinese kôd projektu i da to učinite tako da rasteretite sebe i održavaoca projekta što je više moguće, kao i kako da održavate projekat uspešno sa većim brojem programera koji doprinose sadržaju.

**Distribuirani tokovi rada**

Za razliku od centralizovanih sistema za kontrolu verzije (CVCS), distributivna priroda Gita vam omogućava da budete mnogo fleksibilniji po pitanju načina na koji programeri kolaboriraju u projektima. Kod centralizovanih sistema, svi programeri su čvorovi koji manje-više jednako rade na centralnom habu. Kod Gita, međutim, svaki programer je potencijalno i čvor i hab — odnosno, svaki programer može i da doprinese kodu drugim repozitorijumima i da održava javni repozitorijum po kome drugi mogu da baziraju svoj rad i kome mogu da doprinesu. Ovo otvara ogroman spektar mogućnosti za tok rada za vaše projekte i/ili za vaš tim, zato ćemo preći nekoliko čestih paradigmi koje koriste ovu fleksibilnost. Pogledaćemo prednosti i moguće mane svakog dizajna; možete da izaberete samo jedan od njih koji ćete korstiti, ili možete da pomešate osobine iz svakog od njih.

**Centralizovan tok rada**

U centralizovanim sistemima, generalno postoji jedinstven kolaboracioni model — centralizovani tok rada. Jedan centralni hab, ili repozitorijum, može da prihvati kod, a svi drugi sinhronišu svoj rad prema njemu. Programeri su čvorovi — potrošači tog haba — i sinhronišu se prema tom jednom mestu.



Figure 54. Centralizovani tok rada.

Ovo znači da ako dva developera kloniraju sa haba i obojica naprave promene, prvi developer koji gurne svoje promene nazad na hab može to da uradi bez problema. Drugi developer mora da se spoji u rad prvog pre nego što gurne svoje promene, kako ne bi pisao preko promena koje je napravio prvi. Ovaj koncept je na Gitu isti kao i na *Subversion*-u (ili bilo kom drugom CVCS-u), i ovaj model radi odlično u Gitu.

Ako ste se već navikli na centralizovani tok rada u svojoj kompaniji ili timu, možete lako da nastavite da koristite taj tok rada sa Gitom. Jednostavno podesite jedan repozitorijum, i dajte svim članovima tima dozvolu za guranje na njega; Git neće dozvoliti korisnicima da pišu preko tuđih promena. Recimo da Marko i Milica počinju da rade istovremeno. Marko završava svoje promene i gura ih na server. Onda Milica pokušava da gurne svoje promene, ali server ih odbija. Server joj kaže da pokušava da gurne promene koje se ne uklapaju u metodu motanja unapred i da neće moći da uradi to dok ne pribavi podatke i ne uradi spoj. Ovaj tok rada odgovara mnogim ljudima jer je to paradigma na koju su navikli.

Ovo nije ograničeno samo na male timove. Sa Gitovim modelom za grananje, moguće je da na stotine developera uspešno radi na jednom projektu koristeći istovremeno na desetine grana.

**Tok rada sa rukovodiocem integracija**

Pošto Git dopušta da imate više udaljenih repozitorijuma, moguće je dizajnirati tok rada u kome svaki developer ima pristup čitanja sopstvenom javnom repozitorijumu i čitanja tuđih. Ovakav scenario često uključuje i kanonički repozitorijum koji predstavlja "zvaničan" projekat. Kako biste doprineli takvom projektu, treba da napravite sopstveni javni klon projekta i na njega gurate izmene. Onda treba da pošaljete zahtev održavaocu glavnog projekta da povuče vaše izmene. Održavalac onda može da doda vaš repozitorijum kao rimout, da testira promene lokalno, spoji ih u sopstvenu granu, i onda gurne nazad na repozitorijum. Ovaj proces radi na sledeći način:

1. Održavalac projekta gurne promene na svoj javni repozitorijum.
2. Kontributor klonira taj repozitorijum i pravi promene.
3. Kontributor gura promene na ličnoj javnoj kopiji.
4. Kontributor šalje održavaocu mejl sa molbom da povuče promene.
5. Održavalac dodaje kontributorov repozitorijum kao rimout i spaja lokalno.
6. Održavalac gura spojene promene na glavni repozitorijum.



Figure 55. Tok rada sa rukovodiocem integracija.

Ovo je veoma čest tok rada sa alatima koji su bazirani na habovima kao što je GitHub ili GitLab, gde je lako forkovati projekat i gurnuti promene u svoj fork koje će svi videti. Jedna od glavnih prednosti ovog pristupa je to što možete da nastavite sa svojim radom, a održavalac glavnog repozitorijuma može da povuče vaše promene u bilo kom trenutku. Kontributirori ne moraju da čekaju da projekat pripoji njihove promene — svako radi svojim tempom.

**Tok rada sa diktatorima i poručnicima**

Ovo je varijanta toka rada sa više repozitorijuma. Generalno ga koriste ogromni projekti sa stotinama kolaboratora; jedan poznat primer je Linuks kernel. Razni rukovodioci integracijama su zaduženi za određene delove repozitorijuma; oni su poručnici. Svi poručnici imaju jednog rukovodioca integracijama poznat kao blagonakloni diktator. Repozitorijum blagonaklonog diktatora služi kao referentni repozitorijum sa kog svi kolaboratori treba da povuku. Ovaj proces radi na sledeći način (pogledajte [Tok rada sa blagonaklonim diktatorom.](https://git-scm.com/book/sr/v2/ch00/wfdiag_c)):

1. Obični developeri rade na svojim tematskih granama i rebaziraju svoj rad na vrh master-a. Master grana je diktatorova.
2. Poručnici spajaju tematske grane developerâ u svoju master granu.
3. Diktator spaja master grane poručnika u diktatorovu master granu.
4. Diktator gura svoj master na referentni repozitorijum kako bi drugi developeri mogli da ga rebaziraju.



Figure 56. Tok rada sa blagonaklonim diktatorom.

Ovakav tok rada nije čest, ali može da bude koristan kod velikih projekata, ili u okruženjima u kojima je hijerarhija jako izražena. Dozvoljava da vođa projekta (diktator) delegira veliki deo posla i sakuplja velike podskupove koda sa više mesta pre nego što ih integriše.

**Rezime tokova rada**

To su bili neki često korišćeni tokovi rada koje je moguće sprovesti u delo koristeći distibuirane sisteme kao što je Git, ali jasno je da su moguće mnoge varijacije koje treba prilagoditi određenom toku rada u praksi. Sada kada (bi trebalo da) možete da odlučite koja kombinacija tokova rada će vam odgovarati, pokrićemo neke specifičnije primene oko toga kako da obavite neke glavne zadatke ovih tokova rada. U sledećem odeljku, naučićete nešto o nekoliko čestih obrazaca po kojima se doprinosi projektu.

# 5.2 Distribuirani Git - Kako doprineti projektu

## Kako doprineti projektu

Glavni problem kod opisivanja načina na koji treba doprinositi projektu je to što postoji veliki broj varijcija na koji se to može učiniti. Pošto je Git veoma fleksibilan, ljudi rade na mnogo načina, i problematično je opisati kako treba da doprinosite — svaki projekat je drugačiji. Neke od varijabli koje su uključene u to je broj ljudi koji aktivno doprinose, izabrani tok rada, pristup vašim komitovima, i mogućnost eksterne metode za doprinos.

Prva varijabla je broj ljudi koji aktivno učestvuje — pored ovoga, nameće se pitanje i koliko često. U mnogim slučajevima, imaćete dva ili tri developera sa nekoliko komitova na dan, ili možda manje za ne tako aktivne projekte. Za veće kompanije ili projekte, broj developera bi mogao da pređe nekoliko hiljada, sa stotinama ili hiljadama komitova na dan. Ovo je važno jer sa više developera možete da naiđete na više problema jer kôd neće uvek moći da se glatko spoji. Promene koje podnesete bi mogle da budu zastarele ili pregažene radom koji je neko spojio dok ste vi radili ili dok su vaše promene čekale na red da se prihvate ili primene. Kako da održite svoj kôd konstantno aktuelnim, a komitove validnim?

Sledeća varijabla je tok rada koji projekat koristi. Da li je centralizovan, gde svaki developer ima jednak pristup za pisanje glavnoj liniji? Da li projekat ima održavaoca ili rukovodioca integracijama koji proverava sve zakrpe? Da li se zakrpe pregleđuju i odobravaju? Da li ste vi uključeni u taj proces? Da li je primenjen sistem sa poručnicima, i da li morate prvo njima da pošaljete svoj rad?

Sledeći problem je pristup vaših komitova. Tok rada koji je potreban da biste doprineli projektu je mnogo drugačiji ako imate pristup pisanja projektu nego ako ga nemate. Ako nemate pristup pisanja, kako se odlučuje da li će vaš rad biti prihvaćen u projekat? Da li uopšte postoji takva politika? Koliko stvari menjate u jednom trenutku? Koliko često doprinosite?

Sva ova pitanja mogu da utiču na to kako efikasno treba doprinositi projektu i kakvi tokovi rada su prioritetni ili dostupni vama. Pokrićemo aspekte svega ovoga u seriji slučajeva korišćenja, krenuvši od najjednostavnijeg do najkomplikovanijeg; trebalo bi da možete da konstruišete specifične tokove rada koji se javljaju u praksi na osnovu ovih primera.

### Smernice za komitove

Pre nego što pogledamo specifične slučajeve korišćenja, evo nekih brzih napomena o komit porukama. Imati dobre smernice za kreiranje komitova i pridržavati se njih će učiniti rad sa Gitom i kolaboraciju sa ostalima mnogo jednostavnijom. Git projekat snabdeva dokument koji opisuje brojne dobre savete za kreiranje komitova za slanje zakrpi — možete da ih pročitate u Gitovom izvornom kodu u fajlu Documentation/SubmittingPatches.

Prvo, ne želite da sabmitujete nikakve greške oko karaktera za razmak, tzv. razmak-greške (whitespace errors). Git ima jednostavan način da proverite ovo — pre nego što komitujete, pokrenite git diff --check, što će identifikovati moguće razmak-greške i izlistati vam ih.



Figure 57. Izlaz komande git diff --check.

Ako pokrenete tu komandu pre komitovanja, videćete ima li razmak-grešaka koje biste komitovali koje bi smetale drugim developerima.

Dalje, trudite se da svaki komit bude logički izdvojen skup promena. Ako možete, probajte da promene učinite lako svarljivim — nemojte da kodirate ceo vikend na pet različitih problema i da ih onda sve strpate u jedan komit u ponedeljak. Čak i ako ne komitujete tokom vikenda, koristite stejdž u ponedeljak da podelite svoj rad na barem jedan komit po problemu koji ste rešavali, sa korisnim porukama. Ako neke promene modifikuju isti fajl, probajte da korstite git add --patch da parcijalno stejdžujete fajl (detaljnije odrađeno u [Interactive Staging](https://git-scm.com/book/sr/v2/ch00/_interactive_staging)). Snimak projekta na vrhu grane je isti bilo da komitujete jednom ili pet puta, sve dok sve promene budu dodate na kraju, zato se trudite da stvari učinite lakšim svojim programerima-kolegama kada budu gledali vaše promene. Sa ovim pristupom je lakše da se kasnije locira potencijalna greška ili da se vrati na neko od pređašnjih stanja. [Rewriting History](https://git-scm.com/book/sr/v2/ch00/_rewriting_history) opisuje brojne korisne Git trikove za pisanje preko istorije i interaktivno stejdžovanje fajlova — koristite ove alate da pomognete sebi da izgradite čistu i jasnu istoriju pre nego što pošaljete svoj rad drugima.

Još jedna stvar koju treba da imate na umu jeste komit poruka. Sticanje navike da komit poruke budu kvalitetne učiniće korišćenje Gita i kolaboraciju sa drugima mnogo lakšom. Kao opšte pravilo, poruke treba da počnu jednom linijom koja nije duža od 50 karaktera i koja koncizno opisuje promenu, za kojom sledi prazna linija, a zatim detaljnije objašnjenje. Git projekat zahteva da detaljnije objašnjenje uključi motivaciju za promenu i da iskaže kontrast koji ima implementacija u odnosu na prethodno ponašanje — ovu smernicu je dobro pratiti. Takođe je dobra ideja koristiti imperativni prezent u porukama. Drugim rečima, izdajte naredbe. Umesto "adding test for" ili "I added tests for", korsitite "add test for". Evo originalne verzije obrasca koji je napisao Tim Poup:

Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary. Wrap it to

about 72 characters or so. In some contexts, the first

line is treated as the subject of an email and the rest of

the text as the body. The blank line separating the

summary from the body is critical (unless you omit the body

entirely); tools like rebase can get confused if you run

the two together.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet,

preceded by a single space, with blank lines in

between, but conventions vary here

Ako sve vaše komit poruke izgledaju ovako, stvari će biti mnogo jednostavnije za vas i developere sa kojima radite. Git projekat ima dobro formatirane komit poruke — pokrenite git log --no-merges tamo da vidite kako izgleda lepo formatirana istorija komitova u projektu.

U sledećim primerima, i kroz većinu ostatka knjige, radi kratkoće ova knjiga neće imati lepo formatirane poruke; umesto toga, koristićemo -m opciju uz git commit. Radite kako vam kažemo, a ne kako mi radimo.

### Mali privatni tim

Najjednostavnija postavka na koju ćete verovatno naići je privatni projekat sa jednim ili dva developera. "Privatni" u ovom kontekstu znači da je zatvorenog koda — nije dostupan ostatku sveta. Vi i svi ostali developeri imate dozovlu da gurate promene na repozitorijum.

U ovakvom okruženju, možete da pratite tok rada sličan onom koji imate kada koristite Subversion ili drugi centralizovan sistem. I dalje dobijate prednosti zbog stvari kao što su oflajn komitovanje i znatno jednostavnije grananje i spajanje, ali tok rada može da bude veoma sličan; glavna razlika je to što se spojevi dešavaju na strani klijenta umesto na strani servera tokom komitova. Da vidimo kako bi stvar mogla da izgleda kada dva developera počnu da rade na deljivom repozitorijumu. Prvi develpoer, Džon, klonira repozitorijum, napravi izmene i komituje lokalno. (Poruke protokola su zamenjene sa ... da bi se skratile.)

# John's Machine

$ git clone john@githost:simplegit.git

Initialized empty Git repository in /home/john/simplegit/.git/

...

$ cd simplegit/

$ vim lib/simplegit.rb

$ git commit -am 'removed invalid default value'

[master 738ee87] removed invalid default value

1 files changed, 1 insertions(+), 1 deletions(-)

Drugi developer, Džesika, radi istu stvar — klonira repozitorijum i komituje promenu:

# Jessica's Machine

$ git clone jessica@githost:simplegit.git

Initialized empty Git repository in /home/jessica/simplegit/.git/

...

$ cd simplegit/

$ vim TODO

$ git commit -am 'add reset task'

[master fbff5bc] add reset task

1 files changed, 1 insertions(+), 0 deletions(-)

Sad Džesika gura svoj rad na server:

# Jessica's Machine

$ git push origin master

...

To jessica@githost:simplegit.git

1edee6b..fbff5bc master -> master

Džon pokušava da gurne i svoje promene:

# John's Machine

$ git push origin master

To john@githost:simplegit.git

! [rejected] master -> master (non-fast forward)

error: failed to push some refs to 'john@githost:simplegit.git'

Džonu nije dozvoljeno da gurne svoje promene jer je Džesika to u međuvremenu uradila. Ovo je posebno važno razumeti ako ste navikli na Subversion, jer ćete primetiti da njih dvoje nisu editovali isti fajl. Kada se edituju različiti fajlovi, Subversion automatski radi takve spojeve na serveru, dok u Gitu morate da spojite komitove lokalno. Džon mora da pribavi Džesikine promene i da ih spoji pre nego što mu bude dozvoljeno da gurne svoje:

$ git fetch origin

...

From john@githost:simplegit

+ 049d078...fbff5bc master -> origin/master

Sada Džonov lokalni repozitorijum izgleda ovako nekako:



Figure 58. Džonova divergentna istorija.

Džon ima referencu na promene koje je Džesika gurnula, ali mora da spoji sopstveni rad sa njima pre nego što mu se dozvoli guranje:

$ git merge origin/master

Merge made by recursive.

TODO | 1 +

1 files changed, 1 insertions(+), 0 deletions(-)

Spoj prolazi glatko — Džonova istorija komitova sada izgleda ovako:



Figure 59. Džonov repozitorijum posle spajanja origin/master.

Sada Džon može da testira svoj kod da bi se uverio da radi kako valja, i onda može da gurne svoj novi spojeni rad na server:

$ git push origin master

...

To john@githost:simplegit.git

fbff5bc..72bbc59 master -> master

Konačno, Džonova istorija komitova izgleda ovako:



Figure 60. Džonova istorija komitova posle guranja na origin server.

U međuvremenu, Džesika je radila na tematskoj grani. Napravila je tematsku granu issue54 i napravila tri komita na toj grani. Još nije pribavila Džonove promene, tako da njena istorija komitova izgleda ovako:



Figure 61. Džesikina tematska grana.

Džesika želi da se sinhroniše sa Džonom, pa pribavlja podatke:

# Jessica's Machine

$ git fetch origin

...

From jessica@githost:simplegit

fbff5bc..72bbc59 master -> origin/master

Ovo povlači rad koji je Džon u međuvremenu gurnuo. Džesikina istorija sada izgleda ovako:



Figure 62. Džesikina istorija posle pribavljanja Džonovih promena.

Džesika misli da je njena tematska grana spremna, pa želi da zna šta treba da spoji u svoj rad da bi mogla da gurne promene. Pokreće git log da sazna:

$ git log --no-merges issue54..origin/master

commit 738ee872852dfaa9d6634e0dea7a324040193016

Author: John Smith <jsmith@example.com>

Date: Fri May 29 16:01:27 2009 -0700

removed invalid default value

issue54..origin/master sintaksa je filter za log koji traži od Gita da pokaže samo listu komitova koji su na drugoj grani (u ovom slučaju origin/master) a nisu na prvoj grani (u ovom slučaju issue54). Detaljnije ćemo obraditi ovu sintaksu u [Commit Ranges](https://git-scm.com/book/sr/v2/ch00/_commit_ranges).

Zasad, sa izlaza vidimo da postoji samo jedan komit koji je Džon napravio a da se Džesika nije spojila sa njim. Ako spoji origin/master, to je jedini komit koji će promeniti njen lokalni rad.

Sada Džesika može da spoji svoju tematsku granu u svoju master granu, da spoji Džonove promene (origin/master) u svoju master granu, i da onda gurne promene na server. Prvo, skače nazad na master granu da integriše sav svoj rad:

$ git checkout master

Switched to branch 'master'

Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.

Može prvo da spoji bilo origin/master ili issue54 — obe se nalaze uzvodno, pa redosled nije važan. Krajnji snimak će biti identičan koji god redosled da izabere; samo će istorija biti neznatno drugačija. Bira da se prvo spoji sa issue54:

$ git merge issue54

Updating fbff5bc..4af4298

Fast forward

README | 1 +

lib/simplegit.rb | 6 +++++-

2 files changed, 6 insertions(+), 1 deletions(-)

Ne javljaju se problemi; kao što se vidi, radilo se o jednostavnom motanju unapred. Sada se Džesika spaja u Džonov rad (origin/master):

$ git merge origin/master

Auto-merging lib/simplegit.rb

Merge made by recursive.

lib/simplegit.rb | 2 +-

1 files changed, 1 insertions(+), 1 deletions(-)

Sve se spaja bez problema, i Džesikina istorija izgleda ovako:



Figure 63. Džesikina istorija nakon spajanja sa Džonovim promenama.

Sada je origin/master dostupan sa Džesikine master grane, tako da bi trebalo da može uspešno da gurne promene (pod pretpostavkom da Džon nije ponovo gurnuo neke svoje promene u međuvremenu):

$ git push origin master

...

To jessica@githost:simplegit.git

72bbc59..8059c15 master -> master

Svi developeri su komitovali nekoliko puta i uspešno spojili svoj rad.



Figure 64. Džesikina istorija nakon guranja svih promena nazad na server.

To je jedan od najjednostavnijih tokova rada. Radite neko vreme, u opštem slučaju na tematskoj grani, i spojite se u master granu kada ste spremni za integraciju. Kada želite da podelite taj rad, spojite to u sopstvenu master granu, i onda pribavite i spojite origin/master ako se promenio, i na kraju gurnete master granu na server. Opšta sekvenca izgleda nekako ovako:



Figure 65. Opšta sekvenca događaja za jednostavan tok rada sa više developera.

### Privatni tim sa rukovodstvom

U ovom scenariju, pogledaćemo uloge kontributora u većoj privatnoj grupi. Naučićete kako da radite u okruženju gde male grupe sarađuju na delovima koda i onda se takvi doprinosi na nivou tima integrišu od strane nekog drugog.

Recimo da Džon i Džesika rade zajedno na jednom delu, dok Džesika i Džozi rade na drugom. U ovom slučaju, kompanija koristi neku vrstu toka rada sa rukovodstvom za integracije gde se rad pojedinačnih grupa integriše samo od strane određenih inženjera, i master granu glavnog repozitorijuma mogu da apdejtuju samo oni. U ovom scenariju, sav rad se obavlja na timski baziranim granama i kasnije ga povlače integratori.

Hajde da pratimo Džesikin tok rada dok radi na dve strane, odnosno dok paralelno kolaborira sa dva različita developera u ovom okruženju. Pod pretpostavkom da već ima kloniran repozitorijum, odlučuje da prvo radi na featureA. Pravi novu granu za ovo i radi na njoj:

# Jessica's Machine

$ git checkout -b featureA

Switched to a new branch 'featureA'

$ vim lib/simplegit.rb

$ git commit -am 'add limit to log function'

[featureA 3300904] add limit to log function

1 files changed, 1 insertions(+), 1 deletions(-)

Sada treba da podeli svoj rad sa Džonom, pa gura komitove sa featureA grane na server. Džesika nema dozvolu da gurne na master granu — samo integratori mogu to — pa mora da gurne na drugu granu da bi sarađivala sa Džonom:

$ git push -u origin featureA

...

To jessica@githost:simplegit.git

\* [new branch] featureA -> featureA

Džesika šalje mejl Džonu da mu kaže da je gurnula neke stvari na granu koja se zove featureA i da on sada može da pogleda to. Dok čeka na povratnu informaciju od Džona, Džesika odlučuje da počne da radi na featureB sa Džozi. Za početak, počinje novu granu, i bazira je na master grani sa severa:

# Jessica's Machine

$ git fetch origin

$ git checkout -b featureB origin/master

Switched to a new branch 'featureB'

Sada Džesika pravi nekoliko komitova na grani featureB:

$ vim lib/simplegit.rb

$ git commit -am 'made the ls-tree function recursive'

[featureB e5b0fdc] made the ls-tree function recursive

1 files changed, 1 insertions(+), 1 deletions(-)

$ vim lib/simplegit.rb

$ git commit -am 'add ls-files'

[featureB 8512791] add ls-files

1 files changed, 5 insertions(+), 0 deletions(-)

Džesikin repozitorijum izgleda ovako:



Figure 66. Džesiina inicijalna istorija komitova.

Spremna je da gurne svoj rad, ali dobija mejl od Džozi da je grana sa nekim inicijalnim radom već gurnuta na server pod imenom featureBee. Džesika prvo mora da spoji te promene sa sobom pre nego što može da gurne promene na server. Ona pribavlja Džozine promene sa git fetch:

$ git fetch origin

...

From jessica@githost:simplegit

\* [new branch] featureBee -> origin/featureBee

Džesika sada može da spoji ovo sa svojim radom pomoću git merge:

$ git merge origin/featureBee

Auto-merging lib/simplegit.rb

Merge made by recursive.

lib/simplegit.rb | 4 ++++

1 files changed, 4 insertions(+), 0 deletions(-)

Ovde nastaje mali problem — treba da gurne spojeni rad na svojoj featureB grani na featureBee granu na server. To može da uradi tako što će specificirati lokalnu granu, zatim dve tačke (:) i na kraju udaljenu granu komandi git push:

$ git push -u origin featureB:featureBee

...

To jessica@githost:simplegit.git

fba9af8..cd685d1 featureB -> featureBee

Ovo se zove refspec. Pogledajte [Refspek](https://git-scm.com/book/sr/v2/ch00/_refspec) za detaljniju diskusiju o Gitovim refspekovima i ranim stvarima koje možete da uradite pomoću njih. Primetite i -u zastavicu; ovo je skraćeno od --set-upstream, što konfiguriše grane za lakše guranje i povlačenje kasnije.

Sada, Džon pušta mejl Džesiki i kaže joj da je gurnuo neke promene na featureA i moli je da ih pogleda. Ona pokreće git fetch da povuče te promene:

$ git fetch origin

...

From jessica@githost:simplegit

3300904..aad881d featureA -> origin/featureA

Onda može da vidi šta se promenilo sa git log:

$ git log featureA..origin/featureA

commit aad881d154acdaeb2b6b18ea0e827ed8a6d671e6

Author: John Smith <jsmith@example.com>

Date: Fri May 29 19:57:33 2009 -0700

changed log output to 30 from 25

Konačno, spaja Džonov rad u svoju featureA granu.

$ git checkout featureA

Switched to branch 'featureA'

$ git merge origin/featureA

Updating 3300904..aad881d

Fast forward

lib/simplegit.rb | 10 +++++++++-

1 files changed, 9 insertions(+), 1 deletions(-)

Džesika želi da napravi neku malu promenu, pa komituje ponovo i gurne ovo nazad na server:

$ git commit -am 'small tweak'

[featureA 774b3ed] small tweak

1 files changed, 1 insertions(+), 1 deletions(-)

$ git push

...

To jessica@githost:simplegit.git

3300904..774b3ed featureA -> featureA

Džesikina istorija komitova sada izgleda ovako:



Figure 67. Džesikina isotorija nakon komitovanja na featureA granu.

Džesika, Džozi i Džon informišu integratore da su grane featureA i featureBee na serveru spremne za integraciju sa glavnom linijom. Nakon što integratori spoje ove grane u glavnu liniju, pribavljanje će dovesti novi spojni komit, pa će istorija izgledati ovako:



Figure 68. Džesikina istorija nakon spajanja obeju tematskih grana.

Mnoge grupe prelaze na Git zbog mogućnosti da imaju više timova koji rade paralelno, spajajući različite linije rada tokom procesa. Mogućnost manjih podgrupa timova da doprinose preko udaljenih grana a da ne smetaju čitavom timu je velika beneficija Gita. Sekvenca toka rada koju ste ovde videli izgleda ovako:



Figure 69. Osnovan sekvenca ovakvog toka rada privatnog tima sa rukovodstvom.

### Forkovan javni projekat

Doprinos javnim projektima teče malo drugačije. Pošto nemate dozvolu da direktno menjate grane na projektu, morate da date svoj rad održavaocima projekta na neki drugi način. Prvi primer opisuje kontribuiranje preko forkovanja na Git hostovima koji podržavaju lako forkovanje.

Mnogi hostinzi podržavaju ovo (uključujući GitHub, BitBucket, Google Code, repo.or.cz, i drugi), i mnogi održavaoci projekta očekuju ovakav stil kontribuiranja. Za početak, verovatno treba da klonirate glavni repozitorijum, napravite tematsku granu za zakrpu ili seriju zakrpa na kojoj radite, i da tu radite svoj posao. Sekvenca u suštini izgleda ovako:

$ git clone (url)

$ cd project

$ git checkout -b featureA

# (work)

$ git commit

# (work)

$ git commit

|  |  |
| --- | --- |
| Note | Možda ćete želeti da iskoristite rebase -i da zgnječite rad na jedan komit, ili da preuredite rad u komitovima da bi održavalac jednostavnije pregledao zakrpu — vidite [Rewriting History](https://git-scm.com/book/sr/v2/ch00/_rewriting_history) za više informacija o interaktivnom rebaziranju. |

Kada je vaš rad na grani obavljen i kada budete spremni da ga prosledite održavaocima, odite na originalnu stranicu projekta i kliknite na Fork dugme; tako pravite svoj lični fork projekta po kome možete pisati. Onda treba da dodate ovaj novi URL repozitorijuma kao drugi rimout, u ovom slučaju nazvan myfork:

$ git remote add myfork (url)

Onda treba da gurnete svoj rad na njega. Lakše je gurnuti tematsku granu na kojoj radite na svoj repozitorijum, nego da se spajate sa master granom i onda to gurnete. Razlog je to taj što, ako rad nije prihvaćen ili je čeripikovan, ne morate da motate svoju master granu unazad. Ako održavaoci spoje, rebaziraju ili čeripikuju vaš rad, svakako ćete ga kad-tad dobiti nazad tako što ćete povući sa njihovog repozitorijuma:

$ git push -u myfork featureA

Kada je vaš rad gurnut na fork, treba da obavestite održavaoca. Ovo se često radi zahtevom za povlačenje (pull request), i možete da ga generišete ili preko vebsajta — GitHub ima svoj mehanizam za zahteve za povlačenje koji ćemo obraditi u [GitHub](https://git-scm.com/book/sr/v2/ch00/_github) — ili možete da pokrenete git request-pullkomandu i pošaljete izlaz na mejl održavaocu projekta manuelno.

Komanda request-pull uzima osnovnu granu u koju želite da povučete tematsku granu i URL Git repozitorijuma sa kog želite da povuku, i štampa sažetak svih promena koje tražite da se povuku. Na primer, ako Džesika želi da Džonu pošalje zahtev za povlačenje, a uradila je dva komita na tematskoj grani koju je upravo gurnula, može da pokrene ovo:

$ git request-pull origin/master myfork

The following changes since commit 1edee6b1d61823a2de3b09c160d7080b8d1b3a40:

John Smith (1):

added a new function

are available in the git repository at:

git://githost/simplegit.git featureA

Jessica Smith (2):

add limit to log function

change log output to 30 from 25

lib/simplegit.rb | 10 +++++++++-

1 files changed, 9 insertions(+), 1 deletions(-)

Izlaz može da se pošalje održavaocu — on mu govori odakle je rad izgranat, rezimira komitove i kaže mu odakle da povuče ovaj rad.

Na projektu gde vi niste održavalac, generalno je lakše da imate granu kao što je master koja uvek prati origin/master i da radite svoj posao na tematskim granama koje lako možete da odbacite ako budu bile odbijene. Imati teme za rad izolovane u tematskim granama takođe čini rebaziranje lakšim ako vrh glavnog repozitorijuma u međuvremenu bude pomeren i vaši komitovi se više ne uklpaju glatko. Na primer, ako želite da pošaljete drugu temu rada na projekat, nemojte nastavljati da radite na tematskoj grani koju ste upravo gurnuli — počnite iz početka, tj. od master grane repozitorijuma:

$ git checkout -b featureB origin/master

# (work)

$ git commit

$ git push myfork featureB

# (email maintainer)

$ git fetch origin

Sada je svaka od tema sadržana unutar silosa — slično kao red zakrpa (patch queue) — koje možete da rebazirate, modifikujete i pišete preko njih a da se teme ne mešaju međusobno i da ne zavise jedna od druge, ovako:



Figure 70. Incijalna istorija komitova sa radom featureB.

Recimo da je održavalac projekta povukao gomilu drugih zakrpa i probao prvo vašu granu, ali ona se više ne spaja bez problema. U tom slučaju, možete da probate da rebazirate tu granu na vrh origin/master-a, da rešite konflikte za održavaoca, i onda ponovo pošaljete svoje promene.

$ git checkout featureA

$ git rebase origin/master

$ git push -f myfork featureA

Ovo piše preko vaše istorije koja će sada izgledati kao [Istorija komitova posle rada featureA.](https://git-scm.com/book/sr/v2/ch00/psp_b).



Figure 71. Istorija komitova posle rada featureA.

Pošto ste rebazirali granu, morate da specificirate -f komandi push da biste mogli da zamenite featureA granu na serveru komitom koji nije njegov sledbenik. Alternativa bi bila da gurnete ovaj novi rad na drugu granu na serveru (koja se recimo zove featureAv2).

Pogledajmo još jedan mogući scenario: održavalac je pogledao vaš rad u drugoj grani i sviđa mu se koncept ali bi voleo da napravite promenu u nekom detalju oko implementacije. Iskoristićete ovu priliku i da pomerite rad da bude baziran na trenutnoj master grani projekta. Počinjete novu granu baziranu na trenutnoj origin/master grani, gnječite featureB promene tamo, rešavate sve konflikte, pravite implementacionu promenu i onda gurate to kao novu granu:

$ git checkout -b featureBv2 origin/master

$ git merge --squash featureB

# (change implementation)

$ git commit

$ git push myfork featureBv2

Opcija --squash uzima sav rad na spojenoj grani i gnječi ga u jedan skup promena, praveći tako stanje repozitorijuma kao da se dogodio pravi spoj, a da zapravo niste napravili komit. Ovo znači da će vaš budući komit imati samo jednog roditelja i dozvoljava vam da uvedete sve promene sa druge grane i onda napravite još promena pre nego što zabeležite novi komit. Opcija --no-commit takođe može da bude korisna da odložite spojni komit u slučaju podrazumevanog procesa spajanja.

Sada možete da pošaljete održavaocu poruku da ste napravili zahtevane izmene i da mogu da nađu te izmene u vašoj featureBv2 grani.



Figure 72. IStorija komitova nakon rada na featureBv2.

### Javni projekat preko mejla

Mnogi projekti imaju utvrđene procedure za prihvatanje zakrpi — moraćete da proverite specifična pravila za svaki projekat, jer će se razlikovati. Pošto ima nekoliko starijih većih projekata koje prihvataju zakrpe preko spisaka adresa developera, preći ćemo taj primer sada.

Tok rada je sličan kao u prethodnom slučaju — napravite tematske grane za svaku zakrpu na kojoj radite. Razlika je u tome kako ih šaljete projektu. Umesto da forkujete projekat i objavite svoju ličnu verziju po kojoj možete pisati, generisaćete mejl verzije svake serije komitova i poslati ih putem spiska adresa developera.

$ git checkout -b topicA

# (work)

$ git commit

# (work)

$ git commit

Sada imate dva komita koja želite da pošaljete spisku adresa. Koristite git format-patch da generišete fajlove formatirane kao mbox koje šaljete spisku putem mejla — tako pretvarate svaki komit u mejl gde se prva linija komit poruke shvata kao tema mejla a ostatak mejla kao preostali deo komit poruke plus zakrpa koju taj komit uvodi se shvata kao telo. Dobra stvar u vezi s ovime je to što prihvatanje zakrpe koja je generisala za mejl sa format-patch čuva sve inforamcije o komitu.

$ git format-patch -M origin/master

0001-add-limit-to-log-function.patch

0002-changed-log-output-to-30-from-25.patch

Komanda format-patch štampa imena zakrpa-fajlova koje kreira. Zastavica -M govori Gitu da potraži preimenovane fajlove. Fajlovi na kraju izgledaju ovako:

$ cat 0001-add-limit-to-log-function.patch

From 330090432754092d704da8e76ca5c05c198e71a8 Mon Sep 17 00:00:00 2001

From: Jessica Smith <jessica@example.com>

Date: Sun, 6 Apr 2008 10:17:23 -0700

Subject: [PATCH 1/2] add limit to log function

Limit log functionality to the first 20

---

lib/simplegit.rb | 2 +-

1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/lib/simplegit.rb b/lib/simplegit.rb

index 76f47bc..f9815f1 100644

--- a/lib/simplegit.rb

+++ b/lib/simplegit.rb

@@ -14,7 +14,7 @@ class SimpleGit

end

def log(treeish = 'master')

- command("git log #{treeish}")

+ command("git log -n 20 #{treeish}")

end

def ls\_tree(treeish = 'master')

--

2.1.0

Možete i da edituje ove zakrpa-fajlove i da dodate više informacija koje ne želite da se pojave u komit poruci, ali želite da vide svi sa liste adresa. Ako dodate tekst između --- linije i početka zakrpe (linija diff --git), onda će developeri moći da ga pročitaju; ali primenjivanje zakrpe to neće uključiti.

Da pošaljete ovo na spisak adresa, možete da nalepite fajl u svom mejl programu ili da ga pošaljete preko komandne linije. Nalepljivanje teksta često stvara probleme pri formatiranju, pogotovo sa "pametnijim" klijentima koji ne održavaju nove linije i druge znakove razmaka kako treba. Srećom, Git ima alat koji će vam pomoći da pošaljete dobro formatirane zakrpe preko IMAP-a, što bi moglo da vam bude lakše. Demonstriraćemo kako da pošaljete zakrpu preko Gmail-a, što je mejl agent kog znamo najbolje; možete da pročitate detaljne instrukcije o mnogim mejl programima na kraju gorepomenutog fajla Documentation/SubmittingPatches u izvornom kodu Gita.

Prvo, treba da podesite imap odeljak u ~/.gitconfig fajlu. Možete da podesite svaku vrednost posebno serijom git config komandi, ili da ih dodate manuelno, ali na kraju će vaš config fajl izgledati nekako ovako:

[imap]

folder = "[Gmail]/Drafts"

host = imaps://imap.gmail.com

user = user@gmail.com

pass = p4ssw0rd

port = 993

sslverify = false

Ako vaš IMAP server ne koristi SSL, poslednje dve linije verovatno nisu neophne, i vrednost host će biti impa:// umesto imaps://. Kad je to gotovo, možete da pomoću git imap-send postavite seriju zakrpa u Drafts folder na specificiranom IMAP serveru:

$ cat \*.patch |git imap-send

Resolving imap.gmail.com... ok

Connecting to [74.125.142.109]:993... ok

Logging in...

sending 2 messages

100% (2/2) done

Sada bi trebalo da možete da odete u Drafts folder, da promenite To polje u spisak adresa kome šaljete zakrpu, eventualno CC održavaoca ili osobu koje je odgovorna za taj odeljak, i da pošaljete mejl.

Možete da šaljete zakrpe i preko SMTP servera. Kao i ranije, možete da podesite svaku vrednost ponaosob serijom git config komandi, ili da ih dodate ručno u sendmail odeljku ~/.gitconfig fajla:

[sendemail]

smtpencryption = tls

smtpserver = smtp.gmail.com

smtpuser = user@gmail.com

smtpserverport = 587

Kada je ovo gotovo, možete pomoću git send-email da pošaljete svoje zakrpe:

$ git send-email \*.patch

0001-added-limit-to-log-function.patch

0002-changed-log-output-to-30-from-25.patch

Who should the emails appear to be from? [Jessica Smith <jessica@example.com>]

Emails will be sent from: Jessica Smith <jessica@example.com>

Who should the emails be sent to? jessica@example.com

Message-ID to be used as In-Reply-To for the first email? y

Git onda izbaci gomilu log informacija koje izgledaju nekako ovako za svaku zakrpu koju šaljete:

(mbox) Adding cc: Jessica Smith <jessica@example.com> from

\line 'From: Jessica Smith <jessica@example.com>'

OK. Log says:

Sendmail: /usr/sbin/sendmail -i jessica@example.com

From: Jessica Smith <jessica@example.com>

To: jessica@example.com

Subject: [PATCH 1/2] added limit to log function

Date: Sat, 30 May 2009 13:29:15 -0700

Message-Id: <1243715356-61726-1-git-send-email-jessica@example.com>

X-Mailer: git-send-email 1.6.2.rc1.20.g8c5b.dirty

In-Reply-To: <y>

References: <y>

Result: OK

### Rezime

Ovaj odeljak je pokrio brojne česte tokove rada i način na koji se pristupa veoma različitim tipovima Git projekata na koje ćete verovatno naići, i predstavljeni su neki novi alati koji će vam pomoći da radite to. Sada ćemo pogledati kako da radite sa druge strane novčića: održavanje Git projekta. Naučićete kako da budete blagonakloni diktator ili rukovodilac integracijama.

# 5.3 Distribuirani Git - Održavanje projekta

## Održavanje projekta

Sem što treba da znate kako da efikasno doprinesete projektu, verovatno ćete morati da naučite i kako da ga održavate. Ovo može da se sastoji od prihvatanja i primenjivanja zakrpa generisane sa format-patchi poslate vama putem mejla, ili od integrisanja promena na udaljenim granama za repozitorijume koje ste dodali kao rimoutove svom projektu. Bilo da održavate kanoničan repozitorijum ili želite da pomognete tako što ćete verifikovati ili prihvatati zakrpe, treba da znate kako da prihvatite rad na način koji je najpregledniji drugim kontributorima i da možete da ga održavate na duge staze.

### Rad sa tematskim granama

Kada razmišljate o integrisanju novog rada, u opštem slučaju je dobra ideja da stvari prvo isprobate na tematskoj grani — na trenutnoj grani koju ste napravili specifično radi testiranja da li novi delovi koda rade. Na ovaj način je lako uneti male izmene posebno u zakrpu i ostaviti je tako ako ne radi dok ne budete imali vremena da se kasnije posvetite njoj. Održavalac Git projekta neretko pravi i nejmspejs za grane — na primer sc/ruby\_client, gde je sc skraćenica za osobu koja doprinosi radu. Kao što se sećate, možete da napravite grane bazirane na master grani na sledeći način:

$ git branch sc/ruby\_client master

Ili, ako želite da odmah i skočite na nju, možete da iskoristite checkout -b opciju:

$ git checkout -b sc/ruby\_client master

Sada ste spremni da dodate svoj doprinos ovoj tematskoj grani i da odlučite da li želite da je spojite u odgovarajuću dugotrajnu granu.

### Primenjivanje zakrpa preko mejlova

Ako dobijete zakrpu preko mejla i treba da je integrišete u svoj projekat, treba da primenite zakrpu na tematsku granu i da je procenite. Postoje dva načina za primenjivanje zakrpe koju ste dobiji butem mejla: pomoću git apply ili git am.

#### Primenjivanje zakrpe sa apply

Ako ste dobili zakrpu od nekog ko ju je generisao sa git diff ili Juniksovom diff komandom (što nije preporučljivo, pogledajte sledeći odeljak), možete da je primenite git apply komandom. Pod pretpostavkom da ste sačuvati zakrpu na /tmp/patch-ruby-client.patch, možete da je primenite ovako:

$ git apply /tmp/patch-ruby-client.patch

Ovo modifikuje fajlove u radnom direktorijumu. Ovo je skoro identično kao i pokretanje patch -p1komande za prihvatanje zakrpe, mada je više paranoična i prihvata manje nejasnijih parova nego patch. Rukuje i dodavanjem, brisanjem i preimenovanjem fajlova kako je opisano u git diff formatu, što patch neće da uradi. Konačno, git apply je model sa principom "primeni sve ili odbaci sve" gde će se ili primeniti sve ili ništa, dok patch može parcijalno da primenjuje zakrpe, ostavljajući radni direktorijum u čudnom stanju. git apply je i generalno mnogo konzervativniji nego patch. Neće kreirati komit umesto vas — kada ga pokrenete, morate da stejdžujete komit i ručno komitujete novonastale promene.

Možete da koristite git apply i da vidite da li će se zakrpa uklopiti kako valja pre nego što zapravo probate da je primenite — samo pokrenite git apply --check sa zakrpom:

$ git apply --check 0001-seeing-if-this-helps-the-gem.patch

error: patch failed: ticgit.gemspec:1

error: ticgit.gemspec: patch does not apply

Ako nema izlaza, to znači da će se zakrpa prihvatiti kako valja. Ova komanda takođe daje ne-nula izlaz u slučaju neuspeha, tako da možete da je koristite u skriptama ako želite.

#### Primenjivanje zakrpe sa am

Ako je kontributor korisnik Gita i ako je bio dovoljno dobar da iskoristi format-patch komandu da generiše svoju zakrpu, onda će vaš posao biti mnogo lakši jer zakrpa sadrži informacije o autoru i komit poruke. Ako možete, ohrabrujte svoje kontributore da korsite format-patch umesto diff za generisanje zakrpa koje vam šalju. Treba da morate da koristite git apply samo za legat zakrpe i takve stvari.

Da biste primenili zakrpu koja je generisana sa format-patch, koristite git am. Tehnički, git am je stvoren da čita mbox fajlove; to su jednostavni fajlovi običnog tekstualnog formata koji služe za čuvanje jedne ili više mejl poruka u jednom tekstualnom fajlu. Izgledaju nekako ovako:

From 330090432754092d704da8e76ca5c05c198e71a8 Mon Sep 17 00:00:00 2001

From: Jessica Smith <jessica@example.com>

Date: Sun, 6 Apr 2008 10:17:23 -0700

Subject: [PATCH 1/2] add limit to log function

Limit log functionality to the first 20

Ovo je početak izlaza komande format-patch koju ste upravo videli u prethodnom odeljku. Ovo je takođe validan mbox mejl format. Ako vam je neko poslao mejl sa zakrpom koristeći git send-email, a vi skinete to u mbox format, onda možete da usmerite git am na taj mbox fajl, i počeće da primenjuje sve zakrpe koje vidi. Ako pokrenete mejl klijent koji može da čuva nekoliko mejlova u mbox formatu, možete da sačuvate čitavu seriju zakrpi u fajl i onda da iskoristite git am da ih primenjuje jednu po jednu.

Međutim, ako je neko okačio zakrpu koju je generisao sa format-patch na sistem za tikete ili nešto slično, možete da sačuvate fajl lokalno i onda da prosledite taj fajl koji je sačuvan na disku komandi git am da biste je primenili:

$ git am 0001-limit-log-function.patch

Applying: add limit to log function

Kao što vidite, zakrpa je primenjena bez problema i automatski je kreiran novi komit. Informacije o autoru su uzete iz Form i Date hedera sa mejla, a poruka komita je iz Subject-a i tela (pre zakrpe) mejla. Na primer, ako je ova zakrpa primenjena sa mbox primera odozgo, generisani komit bi izgledao nekako ovako:

$ git log --pretty=fuller -1

commit 6c5e70b984a60b3cecd395edd5b48a7575bf58e0

Author: Jessica Smith <jessica@example.com>

AuthorDate: Sun Apr 6 10:17:23 2008 -0700

Commit: Scott Chacon <schacon@gmail.com>

CommitDate: Thu Apr 9 09:19:06 2009 -0700

add limit to log function

Limit log functionality to the first 20

Informacija Commit ukazuje na osobu koja je primenila zakrpu i na vreme kada je to učinjeno. Informacija Author je osoba koja je prvobitno napravila zakrpu i to vreme.

Ali moguće je i da se zakrpa ne primeni bez problema. Možda je glavna grana divergirala predaleko od grane za koju je zakrpa napravljena, ili zakrpa zavisi od druge zakrpe koju još uvek niste primenili. U tom slučaju, git am proces neće uspeti i pitaće vas šta želite da uradite:

$ git am 0001-seeing-if-this-helps-the-gem.patch

Applying: seeing if this helps the gem

error: patch failed: ticgit.gemspec:1

error: ticgit.gemspec: patch does not apply

Patch failed at 0001.

When you have resolved this problem run "git am --resolved".

If you would prefer to skip this patch, instead run "git am --skip".

To restore the original branch and stop patching run "git am --abort".

Ova komanda stavlja konfliktne markere u svim fajlovima sa kojima ima problema, slično kao kod konflikta pri spajanju ili rebaziranja. Problem se takođe rešava na isti način — editovanjem fajla da bi se rešio konflikt; zatim se stejdžuje novi fajl, i onda se pokrene git am --resolved da bi se krenulo na sledeću zakrpu:

$ (fix the file)

$ git add ticgit.gemspec

$ git am --resolved

Applying: seeing if this helps the gem

Ako želite da Git proba da malo inteligentnije reši konflikt, možete da mu prosledite opciju -3, što tera Git da proba trostruki spoj. Ova opcija nije uključena po podrazumevanim podešavanjima jer ne radi ako vam komit na kome zakrpa kaže da je bazirana nije u repozitorijumu. Ako imate taj komit — ako je grana bazirana na javnom komitu — onda je opcija -3 generalno mnogo pametnija o primenjivanju zakrpe sa konfliktom:

$ git am -3 0001-seeing-if-this-helps-the-gem.patch

Applying: seeing if this helps the gem

error: patch failed: ticgit.gemspec:1

error: ticgit.gemspec: patch does not apply

Using index info to reconstruct a base tree...

Falling back to patching base and 3-way merge...

No changes -- Patch already applied.

U ovom slučaju, ova zakrpa je već primenjena. Bez opcie -3, izgleda kao konflikt.

Ako primenjujete nekoliko zakrpa iz mbox-a, možete i da pokrenete am komandu u interaktivnom modu, koja staje na svakoj zakrpi koju pronađe i pita vas da li biste želeli da je primenite.

$ git am -3 -i mbox

Commit Body is:

--------------------------

seeing if this helps the gem

--------------------------

Apply? [y]es/[n]o/[e]dit/[v]iew patch/[a]ccept all

Ovo je lepo ako imate sačuvan poveći broj zakrpa, jer možete da vidite zakrpu prvo ako se ne sećate šta je u pitanju, ili da ne primenite zakrpu ako ste to već uradili.

Kada su sve zakrpe za vašu temu primenjene i komitovane na vašu granu, možete da izaberete da li ćete i kako da ih integrišete u dugotrajniju granu.

### Čekautovanje udaljenih grana

Ako je doprinos došao od Git korisnika koji je podesio sopstven repozitorijum, gurnuo neki broj promena na njega, i onda vam poslao URL do repozitorijuma i ime udaljene grane na kojoj se nalaze promene, možete da ih dodate kao rimout i da uradite spojeve lokalno.

Recimo, ako je vam je Džesika poslala mejl u kome kaže da ima odličnu novu stvar u ruby-client grani svog repozitorijuma, možete da je testirate dodavanjem rimouta i čekautovanjem te grane lokalno:

$ git remote add jessica git://github.com/jessica/myproject.git

$ git fetch jessica

$ git checkout -b rubyclient jessica/ruby-client

Ako vam pošalje još jedan mejl kasnije i kaže vam da postoji još jedna grana sa korisnim stvarima, možete da ih pribavite i čekautujete jer već imate podešvanja za rimout.

Ovo je najkorisnije ako radite sa osobom konzistentno. Ako neko ima samo jednu zakrpu kojom doprinosi s vremena na vreme, onda je prihvatanje preko mejla brže nego zahtevanje da svi imaju svoj server i da svi kontinualno dodaju i brišu rimoutove da bi dobili nove zakrpe. Takođe, malo je verovatno da želite da imate na stotine rimoutova, svaki za nekog ko doprinese samo jednom ili dvaput. Ipak, skripte i hostovani servisi mogu ovo da učine jednostavnijim — zavisi najviše od toga kako razvijate program i kako vaši saradnici kodiraju.

Druga prednost ovog pristupa je to što dobijate i istoriju komitova. Mada možete da imate ozbiljne probleme sa spajanjem, znate gde je u istoriji baziran njihov rad; ispravan trostruki spoj je podrazumevani tako da ne morate da ubacujete -3 i da se nadate da je zakrpa generisana za javni komit kome imate pristup.

Ako ne radite sa osobom konzistentno, ali ipak želite da povučete sa njih na ovaj način, možete da navedete URL udaljenog repozitorijuma git pull komandi. Ovo radi jednovremeno povlačenje i ne čuva URL kao udaljenu referencu:

$ git pull https://github.com/onetimeguy/project

From https://github.com/onetimeguy/project

\* branch HEAD -> FETCH\_HEAD

Merge made by recursive.

### Kako utvrditi šta je uvedeno

Sada imate tematsku granu koja sadrži doprinesen rad. U ovom trenutku možete da odlučite šta ćete da radite sa njim. Ovaj odeljak se ponovo osvrće na nekoliko komandi kako biste videli kako da ih iskoristite da biste videli tačno šta ćete uraditi ako spojite ovo u glavnu granu.

Često je korisno da pogledate izveštaj svih komitova koji su u ovoj grani ali nisu na master grani. Možete da izbacite komitove sa master grane tako što ćete dodati --not ispred imena grane. Ovo radi istu stvar kao i master..contrib format koji smo koristili ranije. Na primer, ako vam kontributor pošalje dve zakrpe i napravite granu koja se zove contrib i primenite te zakrpe tamo, možete da pokrenete ovo:

$ git log contrib --not master

commit 5b6235bd297351589efc4d73316f0a68d484f118

Author: Scott Chacon <schacon@gmail.com>

Date: Fri Oct 24 09:53:59 2008 -0700

seeing if this helps the gem

commit 7482e0d16d04bea79d0dba8988cc78df655f16a0

Author: Scott Chacon <schacon@gmail.com>

Date: Mon Oct 22 19:38:36 2008 -0700

updated the gemspec to hopefully work better

Da biste videli koje promene svaki od komitova uvodi, setite se da možete da prosledite -p opciju komandi git log i onda ćete moći da vidite razliku koja je uvedena pri svakom komitu.

Da biste videli punu razliku onoga što bi se dogodilo ako biste spojili ovu tematsku granu sa drugom granom, možda ćete morati da upotrebite čudan trik da biste dobili ispravne rezultat. Razmislite o pokretanju ove komande:

$ git diff master

Ova komanda vam daje razliku, ali može da vas zavara. Ako se vaša master grana kretala napred od kada ste napravili tematsku granu od nje, onda ćete naizgled dobiti čudne rezultate. Ovo se dešava jer Git direktno poredi snimke poslednjeg komita sa tematske grane na kojoj ste sada i poslednjeg komita master grane. Na primer, ako ste dodali liniju u fajlu na master grani, direktno poređenje snimaka će izgledati kao da će tematska grana da obriše tu liniju.

Ako je master direktan predak tematske grane, ovo nije problem; ali ako su dve istorije divergirale, razlika će izgledati kao da dodajete sve nove stvari u tematsku granu i brišete sve jedinstveno mastergrani.

Ono što stvarno hoćete da vidite su promene koje su dodate na tematskoj grani — rad koji ćete uvesti ako spojite ovu granu sa master-om. To ćete uraditi tako što ćete reći Gitu da uporedi poslednji komit na tematskoj grani sa prvim zajedničkim pretkom koji ima sa master granom.

Tehnički, to možete da uradite tako što ćete eksplicitno naći zajedničkog pretka i onda pokrenuti diffnad time:

$ git merge-base contrib master

36c7dba2c95e6bbb78dfa822519ecfec6e1ca649

$ git diff 36c7db

Međutim, ovo nije zgodno za rad, pa Git nudi još jednu skraćenicu kojom radite istu stvar: sintaksu sa tri tačke. U kontekstu diff komande, možete da stavite tri tačke posle druge grane da biste uradili diffizmeđu poslednjeg komita grane na koje se nalazite i njegovog zajedničkog pretka sa drugom granom:

$ git diff master...contrib

Ova komanda vam prikazuje samo rad koji je trenutna tematska grana uvela od zajedničkog pretka sa master-om. To je veoma korisna sintaksa koju treba upamtiti.

### Integrisanje doprinesenog rada

Kada je sav rad na tematskoj grani spreman za integrisanje u glavniju granu, postavlja se pitanje kako izvesti to. Sem toga, koji sveobuhvatni tok rada želite da koristite na svom projektu? Imate puno izbora, pa ćemo pokriti nekoliko njih.

#### Tokovi rada sa spajanjem

Jedan jednostavan tok rada spaja vaš rad sa master granom. U ovom scenariju, imate master granu koja u suštini sadrži stabilan kod. Kada morate da radite sa tematskim granama koje ste vi napravili ili koje su plod nečijeg doprinosa a vi ste ih potvrdili, spajate ih u svoju master granu, brišete tematsku granu i onda nastavljate proces. Ako imamo repozitorijum sa radom u dve grane koje se zovu ruby\_client i php\_client koji izgleda kao [Istorija sa nekoliko tematskih grana](https://git-scm.com/book/sr/v2/ch00/merwf_a) i spojimo ruby\_clientprvo pa onda php\_client, istorija koju ćete na kraju imati izgleda kao [Posle spajanja tematske grane](https://git-scm.com/book/sr/v2/ch00/merwf_b).



Figure 73. Istorija sa nekoliko tematskih grana



Figure 74. Posle spajanja tematske grane

Ovo je verovatno najjednostavniji tok rada, ali može da bude problematičan ako radite na većim ili stabilnijim projektima gde želite da bude jako obazrivi oko toga šta uvodite u izmene.

Ako imate važniji projekat, možda ćete želeti da koristite dvofazni ciklus spoja. U ovom scenariju, imate dve dugotrajne grane, master i develop, u kojima odlučujete da se master apdejtuje samo kada nastane veoma stabla situacija a sav novi kod se integriše u develop granu. Regularno gurate obe ove grane u javni repozitorijum. Svaki put kada imate novu tematsku granu koju treba spojiti ([Before a topic branch merge.](https://git-scm.com/book/sr/v2/ch00/merwf_c)), spajate je u develop ([After a topic branch merge.](https://git-scm.com/book/sr/v2/ch00/merwf_d)); onda, kada tagujete izdanje, premotate master na koje god mesto da se nalazi sada stablna develop grana ([[merwf\_\_e]](https://git-scm.com/book/sr/v2/ch00/merwf__e)).



Figure 75. Before a topic branch merge.



Figure 76. After a topic branch merge.



Figure 77. After a project release.

Ovako, kada ljudi kloniraju repozitorijum vašeg projekta, mogu ili da čekautuju master da bi izbildovali najstabliniju verziju i lako uvek imaju aktuelnu verziju, ili mogu da čekautuju develop, što je napredniji ali slabije testiran kod. Možete i da nastavite ovaj koncept, da imate integrate granu gde se sav rad spaja u jedno. Onda, kada kôd na toj grani postane stabilan i prođe testiranje, spajate ga na develop granu; i kada se to pokaže kao stabilno neko vreme, premotate master granu unapred.

#### Tokovi rada sa spajanjem velikih grana

Git projekat ima četiri dugotrajne grane: master, next i pu (predloženi apdejtovi) za novi rad i maint za bekportove za održavanje. Kada novi rad bude predstavljen od strane kontributorâ, on se skuplja u tematske grane u održavaocevom repozitorijumu na način sličan onome što smo već opisali. (pogledajte [Managing a complex series of parallel contributed topic branches.](https://git-scm.com/book/sr/v2/ch00/merwf_f)). U ovom trenutku, teme se procenjuju da bi se odredilo da li je bezbedno da se uključe u glavni deo projekta ili još treba da se radi na njima. Ako su bezbedne, spajaju se u next, i ta grana se gura kako bi svi mogli da probaju integrisane teme zajedno.



Figure 78. Managing a complex series of parallel contributed topic branches.

Ako na temama još treba da se radi, spajanje se obavlja u pu. Kada bude bilo odlučeno da su potpuno stabilne, teme se ponovo spajaju sa master i onda se ponovo grade od tema koje su bile u next ali još nisu maturirale do master-a. Ovo znači da se master skoro uvek kreće napred, next se s vremena na vreme rebazira, a pu se često rebazira.



Figure 79. Merging contributed topic branches into long-term integration branches.

Kada se tematska grana konačno spoji sa master, ona se briše iz repozitorijuma. Git projekat takođe ima i maint granu koja se forkuje od poslednjeg izdanja da bi se obezbedile bekportovane zakrpe za slučaj da je potrebno novo izdanje koje samo rešava sitnije probleme a ne uključuje novi sadržaj (maintenance release). Zato, kada klonirate Git repozitorijum, imate četiri grane koje možete da čekautujete i da sagledate projekat u raznim fazama razvoja, u zavisnosti od toga koliko stabilnu verziju želite da imate i kako želite da doprinesete projektu; a održavalac je struktuisao tok rada tako da kritičko sagledavanje novih dorpinosa bude olakšano.

#### Tokovi rada sa rebaziranjem i čeripikovanjem

Drugi održavaoci preferiraju da rebaziraju ili čeripikuju (selektivno izaberu ono sa najviše beneficija od ponuđenog) doprinesen rad sa vrha master grane umesto da se spoje u njega, da bi održali uglavnom linearnu istoriju. Kada imate rad u tematskim granama i odlučili ste da želite da ga integrišete, pomerate se na tu granu i pokrećete komandu za rebaziranje da biste ponovo izgradili promene na trenutni vrh master grane (ili develop, i tako dalje). Ako to bude radilo kako treba, možete da premotate svoju master granu unapred, i dobićete linearnu istoriju projekta.

Drugi način da pomerite predstavljeni rad sa jedne grane na drugu je da čeripikujete. Čeripikovanje je u Gitu kao rebaziranje za jedan komit. Uzima zakrpu koja je uvedena u komit i pokušava da je ponvo primeni na granu na kojoj se trenutno nalazite. Ovo je korisno ako imate veći broj komitova na tematskoj grane a želite da integrišete samo jdan od njih, ili imate samo jedan komit na tematskoj grani i preferirate da selektivno izaberete samo jedan od njih (da ga čeripikujete) umesto da rebazirate. Na primer, pretpostavimo da imate projekat koji izgleda ovako:



Figure 80. Primer istorije pre čeripikovanja.

Ako želite da povučete komit e43a6 u master granu, možete da pokrenete

$ git cherry-pick e43a6fd3e94888d76779ad79fb568ed180e5fcdf

Finished one cherry-pick.

[master]: created a0a41a9: "More friendly message when locking the index fails."

3 files changed, 17 insertions(+), 3 deletions(-)

Ovo povlači istu promenu koja je predstavljena u e43a6, ali dobijate novu SHA-1 vrenost za komit, jer je odgovarajući datum drugačiji. Sada istorija izgleda ovako:



Figure 81. Istorija nakon čeripikovanja komita na tematskoj grani.

Sada možete da obrišete tematsku granu i odbacite komitove koje niste želeli da povučete.

#### Rerere

Ako radite puno spajanja i rebaziranja, ili održavate dugotrajnu tematsku granu, Git ima nešto što se zove "rerere" što može da pomogne.

Rerere je skraćenica od reuse recorded resolution (ponovo iskoristi zabeleženo rešenje) — to je način da se skrati ručno rešavanje konflikta. Kada je rerere uključen, Git će čuvati skup pre- i post-slika od uspešnih spojeva, i ako primeti da postoji konflikt koji liči na neki koji ste već razrešili, onda će iskoristiti rešenje od prošlog puta, i neće taj posao ostaviti vama.

Ovo svojstvo dolazi u dva dela: konfiguraciono podešavanje i komanda. Konfiguraciono podešavanje je rerere.enabled, i dovoljno je korisno da ga stavite u globalna podešvanja:

$ git config --global rerere.enabled true

Sad, kad god uradite spoj za koji rešite konflikt, rešenje će biti zabeleženo u kešu za slučaj da vam ponovo zatreba u budućnosti.

Ako bude bilo potrebene, možete da interagujete sa rerere kešom koristeći git rerere komandu. Kada se pozove sama, Git proverava bazu podataka rešenja i pokušava da nađe pogodak sa bilo kojim od trenutnih konflikata pri spoju i reši ih (mada se ovo radi automatski ako je rerere.enabled podešeno na true). Postoje i podkomande kojima možete da vidite šta će biti zabeleženo, da obrišete određeno rešenje iz keša i da obrišete ceo keš. Detaljnije ćemo preći rerere u [Rerere](https://git-scm.com/book/sr/v2/ch00/_rerere).

### Tagovanje izdanja

Kada ste odlučili da napravite presek i napravite novo izdanje, verovatno je dobra ideja da ostavite tag kako biste mogli da opet kreirate to izdanje u bilo kom trenutku nadalje. Novi tag možete napraviti onako kako smo objasnili u [Osnove Gita](https://git-scm.com/book/sr/v2/ch00/_git_basics_chapter). Ako odlučite da potpišete tag kao održavalac, tagovanje će možda izgledati ovako:

$ git tag -s v1.5 -m 'my signed 1.5 tag'

You need a passphrase to unlock the secret key for

user: "Scott Chacon <schacon@gmail.com>"

1024-bit DSA key, ID F721C45A, created 2009-02-09

Ako potpišete svoje tagove, možda ćete imati problem sa distribucijom javnog PGP ključa koji se koristi za potpisivanje tagova. Održavalac Git projekta rešava ovaj problem tako što uključuje svoj javni ključ kao blob u repozitorijumu i onda dodaje tag koji pokazuje direktno na taj sadržaj. Da biste uradili ovo, možete da otkrijete koji ključ želite tako što ćete pokrenuti gpg --list-keys:

$ gpg --list-keys

/Users/schacon/.gnupg/pubring.gpg

---------------------------------

pub 1024D/F721C45A 2009-02-09 [expires: 2010-02-09]

uid Scott Chacon <schacon@gmail.com>

sub 2048g/45D02282 2009-02-09 [expires: 2010-02-09]

Onda možete direktno da importujete ključ u Gitovu bazu podataka tako što ćete ga eksportovati i pajpovati ga kroz git hash-object, što piše novi blob sa tim sadržajem u Git i natrag vam daje SHA-1 bloba:

$ gpg -a --export F721C45A | git hash-object -w --stdin

659ef797d181633c87ec71ac3f9ba29fe5775b92

Sada kada imate sadržaj svog ključa u Gitu, možete da napravite tag koji pokazuje direktno na njega zako što ćete specificirati novu SHA-1 vrednost koju vam je dala komanda hash-object:

$ git tag -a maintainer-pgp-pub 659ef797d181633c87ec71ac3f9ba29fe5775b92

Ako pokrenete git push --tags, tag maintainer-pgp-pub će biti podeljen sa svima. Ako neko želi da verifikuje tag, može da direktno importuje vaš PGP ključ tako što će povući blob direktno sa baze podataka i importovati ga u PGP:

$ git show maintainer-pgp-pub | gpg --import

Taj ključ može biti iskorišćen za verifikovanje svih vaših potpisanih tagova. Takođe, ako priključite instrukcije u tag poruci, pokretanje git show <tag> će dati krajnjem korisniku specifičnije instrukcije oko verifikacije tagova.

### Generisanje bild broja

Pošto Git nema monotonu inkrementaciju brojeva kao v123 ili ekvivalent koji će ići uz svaki komit, ako želite da imate ime čitljivo za ljude uz svaki komit, možete da pokrenete git describe nad tim komitom. Git vam daje ime najbližeg taga sa brojem komitova navrh tog taga i delimičnu SHA-1 vrednost komita koji opisujete:

$ git describe master

v1.6.2-rc1-20-g8c5b85c

Na ovaj način možete da eksportujete snimak ili bild i imenujete ga na način razumljiv ljudima. Zapravo, ako bildujete Git preko izvornog koda koji ste klonirali iz Git repozitorijuma, git --version vam daje nešto što izgleda ovako. Ako opisujete komit koji ste direktno tagovali, daje vam ime taga.

Komanda git describe favorizuje pribeležene tagove (tagove kreirane sa -a ili -s zastavicom), pa bi tagove izdanja trebalo kreirati na ovaj način ako koristite git describe, da biste bili sigurni da je komit nazvan dobro kada bude opisan. Možete da koristite ovaj string i kao odredište za checkout ili showkomande, mada se oslanja na skraćenu SHA-1 vrednost sa kraja, tako da možda neće važiti doveka. Na primer, Linuks kernel je skoro skočio sa 8 na 10 karaktera da bi se postarao da postoji jedinstvenost među SHA-1 objektima, tako da su stari izlazi komande git describe nevažeći.

### Priprema za izdanje

Sada želite da izdate bild. Jedna od stavati koju ćete želeti da uradite jeste da kreirate arhivu poslednjeg snimka svog koda za one jadne duše koje koriste Git. Komanda koja vam ovo radi jeste git archive:

$ git archive master --prefix='project/' | gzip > `git describe master`.tar.gz

$ ls \*.tar.gz

v1.6.2-rc1-20-g8c5b85c.tar.gz

Ako neko otvori taj tarbol, dobiće poslednji snimak vašeg projekta pod direktorijumom projekta. Možete i da napravite zip arhivu na isti način, samo što ćete proslediti opciju --format=zip komandi git archive:

$ git archive master --prefix='project/' --format=zip > `git describe master`.zip

Sada imate lepu tarbol i zip arhivu izdanja svog projekta koju možete objaviti na svom sajtu ili poslati putem mejla drugim ljudima.

### Kratki log

Vreme je da pošaljete mejl ljudima na vašoj listi koji žele da znaju šta se dešava sa vašim projektom. Dobar način da brzo dobijete neku vrstu beleški promena (tzv. changelog) za ono što je dodato u projekat od poslednjeg izdanja ili mejla je da koristite git shortlog komandu. Ona će sumirati sve komitove u dometu koji specificirate; na primer, sledeća komanda će vam dati kratak pregled svih komitova od poslednjeg izdanja, ako se poslednje izdanje zove v1.0.1:

$ git shortlog --no-merges master --not v1.0.1

Chris Wanstrath (8):

Add support for annotated tags to Grit::Tag

Add packed-refs annotated tag support.

Add Grit::Commit#to\_patch

Update version and History.txt

Remove stray `puts`

Make ls\_tree ignore nils

Tom Preston-Werner (4):

fix dates in history

dynamic version method

Version bump to 1.0.2

Regenerated gemspec for version 1.0.2

Dobijate čist rezime svih komitova od verzije v1.0.1, grupisane po autoru, koji možete poslati kao mejl svojoj listi.

# 6.2 GitHub - Kako doprineti projektu

## Kako doprineti projektu

Sada kada vam je nalog podešen, hajde da prođemo kroz neke detalje koji bi mogli da vam pomognu da doprinesete postojećem projektu.

### Forkovanje projekata

Ako želite da doprinesete postojećem projektu za koji nemate dozvolu za guranje promena, možete da "forkujete" (račvanje) projekat. Ovo znači da će GitHub napraviti kopiju projekta koja je potpuno vaša; projekat sada živi u korisnikovom nejmspejsu i možete da gurate na njega.

|  |  |
| --- | --- |
| Note | Istorijski, termin "forkovanje" ima pomalo negativnu konotaciju jer je ranije opisivao situaciju kada neko uzme projekat sa otvorenim kodom i pokrene ga u drugom pravcu, i tako neretko stvori suparnički projekat i podeli kontributore na tabore. U GitHub-u, "fork" je jednostavno isti projekat u sopstvenom nejmspejsu, što vam dozvoljava da pravite promene za projekat javno i to jednostavno predstavlja otvorenu kontribuciju. |

Zbog ovoga projekti ne moraju da brinu o dodavanju korisnika za kolaboratore da bi im dali dozvolu za guranje promena. Ljudi mogu da forkuju projekat, guraju na njega, i doprinesu svoje promene nazad originalnom repozitorijumu tako što će kreirati nešto što se zove zahtev za povlačenjem (Pull Request), što ćemo pokriti kasnije. Ovo otvara temu za diskusiju sa pregledom koda, i vlasnik ili kontributor može da komunicira o promeni dok vlasnik ne bude zadovoljan; nakon toga vlasnik može da napravi spoj.

Da biste forkovali projekat, posetite stranicu projekta i kliknite na Fork dugme u gornjem desnom uglu stranice.

_Fork_ dugme.

Figure 89. Fork dugme.

Za nekoliko sekundi, bićete odvedeni na stranicu svog novog projekta, sa svojom sopstvenom kopijom koda po kojoj možete da pišete.

### Tok u GitHub-u

GitHub je dizajniran imajući u vidu jedan određeni tok rada za kolaboraciju, koji je centralizovan oko zahteva za povlačenjem. Ovaj način rada funkcioniše bilo da sarađujete sa malim timom prijatleja koji imaju samo jedan deljivi repozitorijum, ili sa globalno distrbuiranom kompanijom ili mrežom stranaca koji doprinose projektu preko brojnih forkova. Bazira se na toku rada [Tematske grane](https://git-scm.com/book/sr/v2/ch00/_topic_branch) koji smo pokrili u [Grananje u Gitu](https://git-scm.com/book/sr/v2/ch00/_git_branching).

Evo kako radi u opštem slučaju:

1. Napravite tematsku granu iz master.
2. Napravite neke komitove da poboljšate projekat.
3. Gurnite ovu granu na svoj GitHub projekat.
4. Otvorite zahtev za povlačenjem na GitHub-u.
5. Diskutujte, i opciono nastavite da komitujete.
6. Vlasnik projekta obradi zahtev za povlačenjem i zatvori ga.

U osnovi, ovo je tok rada sa rukovodiocem integracija koji smo pokrili u [Tok rada sa rukovodiocem integracija](https://git-scm.com/book/sr/v2/ch00/_integration_manager), ali umesto da koristi mejlove za komunikaciju i pregled promena, tim koristi GitHub-ove alate.

Hajde da prođemo kroz primer predlaganja promena projektu otvorenog koda koji je hostovan na GitHub-u koristeći ovaj tok rada.

#### Kako napraviti zahtev za povlačenjem

Toni traži kôd da pokrene na svom programabilnom mikrokontroleru Arduino i našao je odličan program na GitHub-u na <https://github.com/schacon/blink>.

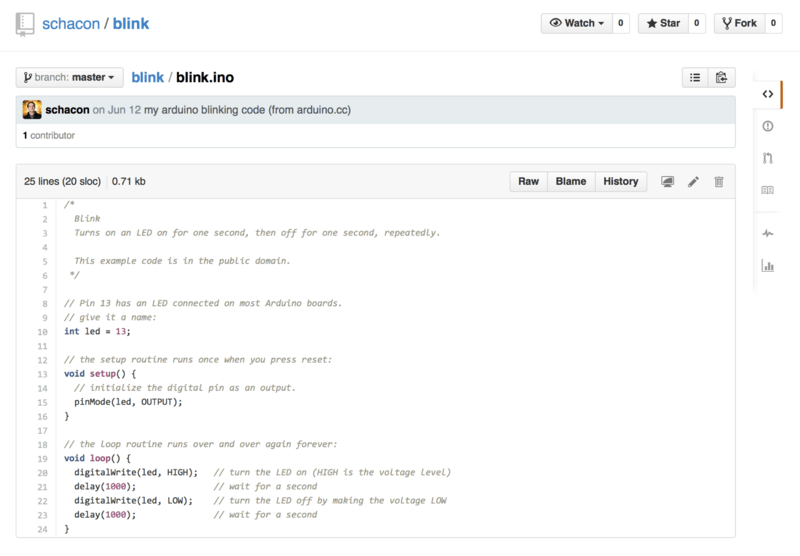


Figure 90. Projekat kome želimo da doprinesemo.

Jedini problem je što je brzina treptanja prebrza, mislimo da bi bilo mnogo bolje da se čekaju tri sekunde imesto jedne za svaku promenu stanja. Zato hajde da popravimo program i da ga pošaljemo nazad na projekat kao predloženu promenu.

Prvo, kliknemo na dugme Fork koje smo ranije pomenuli i dobijamo svoju ličnu kopiju projekta. Naše korisničko ime ovde je tonychacon tako da je naša kopija ovog projekta na adresi https://github.com/tonychacon/blink i tu možemo da je editujemo. Kloniraćemo projekat lokalno, napraviti tematsku granu, promeniti kôd i konačno gurnuti te promene nazad na GitHub.

$ git clone https://github.com/tonychacon/blink **(1)**

Cloning into 'blink'...

$ cd blink

$ git checkout -b slow-blink **(2)**

Switched to a new branch 'slow-blink'

$ sed -i '' 's/1000/3000/' blink.ino **(3)**

$ git diff --word-diff **(4)**

diff --git a/blink.ino b/blink.ino

index 15b9911..a6cc5a5 100644

--- a/blink.ino

+++ b/blink.ino

@@ -18,7 +18,7 @@ void setup() {

// the loop routine runs over and over again forever:

void loop() {

digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)

[-delay(1000);-]{+delay(3000);+} // wait for a second

digitalWrite(led, LOW); // turn the LED off by making the voltage LOW

[-delay(1000);-]{+delay(3000);+} // wait for a second

}

$ git commit -a -m 'three seconds is better' **(5)**

[slow-blink 5ca509d] three seconds is better

1 file changed, 2 insertions(+), 2 deletions(-)

$ git push origin slow-blink **(6)**

Username for 'https://github.com': tonychacon

Password for 'https://tonychacon@github.com':

Counting objects: 5, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 340 bytes | 0 bytes/s, done.

Total 3 (delta 1), reused 0 (delta 0)

To https://github.com/tonychacon/blink

\* [new branch] slow-blink -> slow-blink

1. Kloniramo naš fork projekta lokalno
2. Napravimo deskriptivnu tematsku granu
3. Izmenimo kôd
4. Proverimo da li je promena dobra
5. Komitujemo proemnu na tematsku granu
6. Gurnemo novu tematsku granu nazad na naš GitHub fork

Ako se sada vratimo na naš fork na GitHub-u, videćemo da je GitHub primetio da smo gurnuli novu tematsku granu i predstavlja nam veliko zeleno dugme da pogledamo promene i otvorimo zahtev za povlačenjem ka prvobitnom projektu.

Alternativno, možete da odete na stranicu Branches na https://github.com/<korisnik>/<projekat>/branches da locirate svoju granu i otvorite nov zahtev za povlačenjem odatle.

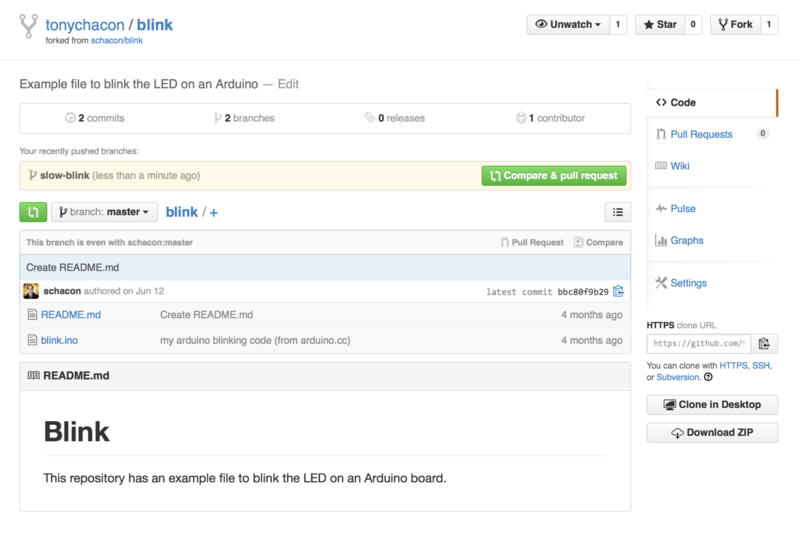


Figure 91. Dugme Pull Request

Ako kliknemo na to zeleno dugme, videćemo ekran gde nas pita da damo naslov i opis našem zahtevu za povlačenjem. Skoro uvek vredi potruditi se malo za ovo, pošto dobar opis pomaže vlasniku prvobitnog projekta da shvati šta želite da uradite, da li su vaše predložene promene ispravne, i da li će prihvatanje promena poboljšati prvobitni projekat.

Vidimo i listu komitova ka našoj tematskoj grani koji su "ispred" master grane (u ovom slučaju, samo jedan) i ujedinjenu razliku svih promena koje će biti napravljene ako vlasnik projekta odluči da spoji ovu granu.

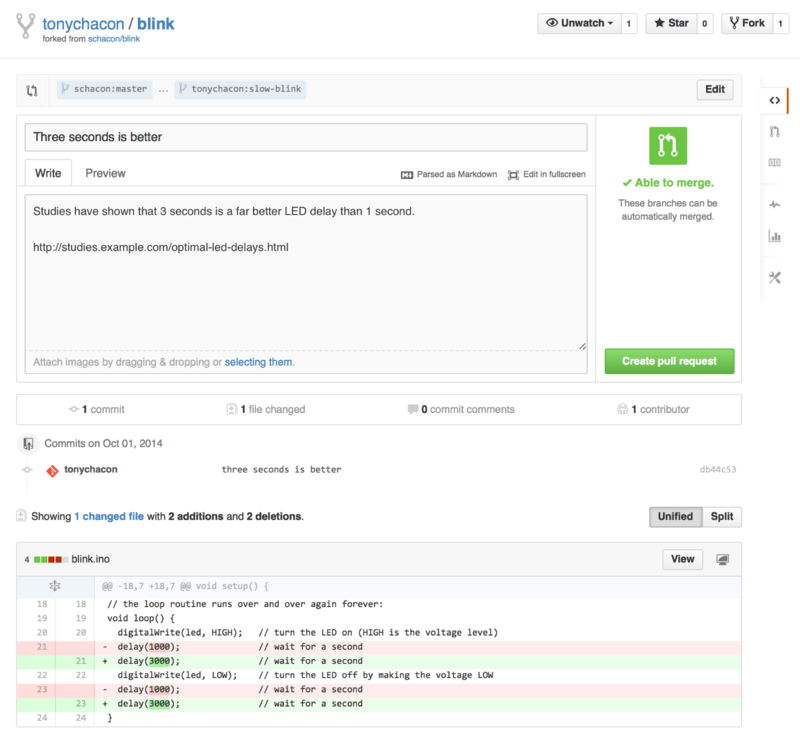


Figure 92. Stranica za pravljenje zahteva za povlačenjem

Kada kliknete na dugme Create pull request na ovom ekranu, vlasnik projekta od koga ste forkovali će dobiti notifikaciju da neko predlaže promenu i dobiće link do stranice koja ima sve informacije u vezi s tim.

|  |  |
| --- | --- |
| Note | Mada se zahtevi za povlačenjem najčešće koriste za projekte kao ovaj, kada kontributor ima kompletnu promenu koju je spreman da napravi, koriste se često i kod internih projekta na početku ciklusa razvića. Pošto možete da nastavite da gurate tematske grane čak i **nakon** što se otvori zahtev za povlačenjem, često se otvara rano i koristi se kao način za iteraciju nad radom kao tim u kontekstu, umesto da se otvori na samom kraju procesa. |

#### Iteracija nad zahtevom za povlačenjem

Sada vlasnik projekta može da pogleda predloženu promenu i da napravi spoj, da je odbije ili da komentariše. Recimo da mu se dopada ideja, ali bi volelo da svetlo bude ugašeno malo duže nego što je upaljeno.

Ovakav razgovor bi se obavio mejlovima u tokovima rada koji su predstavljeni u [Distribuirani Git](https://git-scm.com/book/sr/v2/ch00/_distributed_git), ali na GitHub-u se ovo dešava onlajn. Vlasnik projekta može da pregleda ujedinjenu razliku i ostavi komentar klikom na bilo koju liniju.

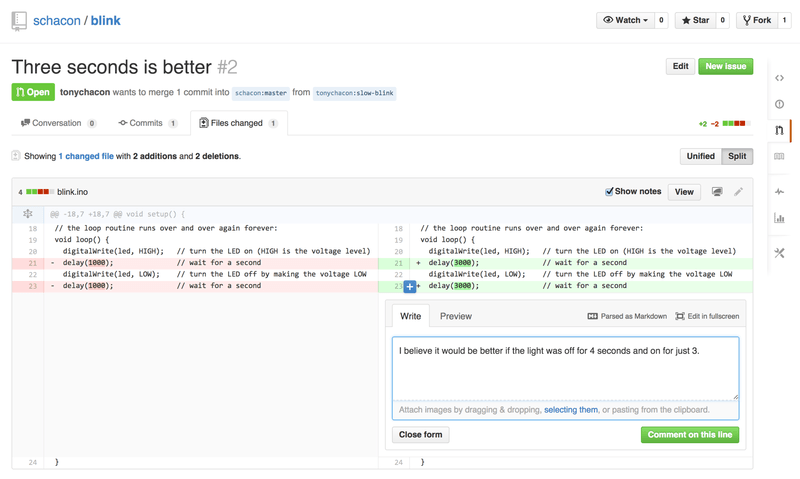


Figure 93. Komentarisanje specifične linije koda u zahtevu za povlačenjem.

Kada održavalac napravi ovaj komentar, osoba koja je otvorila zahtev za povlačenjem (zapravo, svako ko prati repozitorijum) dobiće obaveštenje. Kasnije ćemo pogledati kako promeniti ova podešavanja, ali ako su mu mejl obaveštenja uključena, Toni će dobiti ovakav mejl:



Figure 94. Komentari poslati kao mejl notifikacije.

Svako može da ostavi opšte komentare na zahtevu za povlačenjem. U [Stranica za diskusiju zahteva za povlačenjem.](https://git-scm.com/book/sr/v2/ch00/_pr_discussion) vidimo primer gde vlasnik projekta kometariše na liniju koda a onda ostavlja opšti komentar u odeljku za diskusiju. Možete da vidite da se komentari o kodu dovlače i u ovaj razgovor.

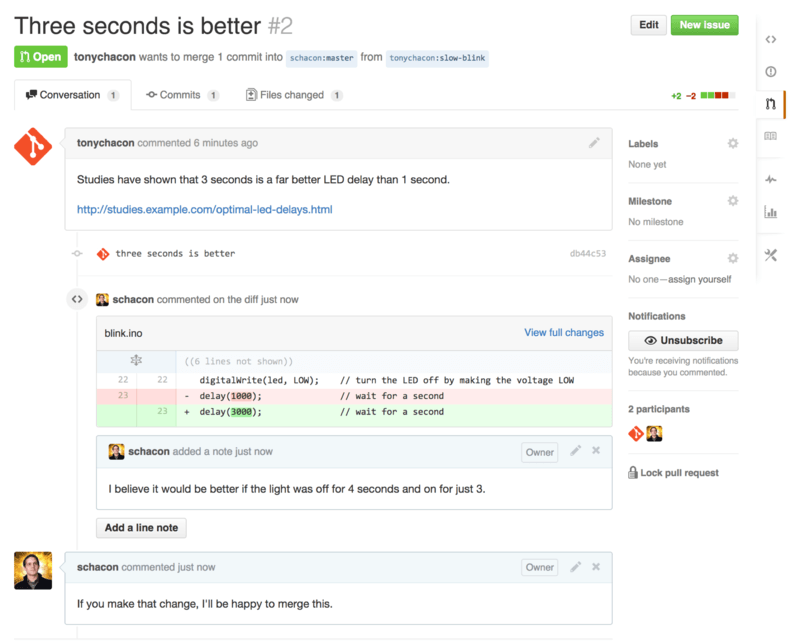


Figure 95. Stranica za diskusiju zahteva za povlačenjem.

Sada kontributor može da vidi šta treba da uradi da bi mu se promena prihvatila. Srećom, ovo je veoma jednostavno. Dok biste preko mejlova morali da opet odmotate seriju zakrpi i ponovo pošaljete sve spisku adresa, sa GitHub-om možete samo da komitujete na tematskoj grani opet i da gurnete promene, koje će automatski ažurirati zahtev za povlačenjem. U [Konačni zahtev za povlačenjem](https://git-scm.com/book/sr/v2/ch00/_pr_final) možete da vidite i to da je stari komentar na kôdu sakriven u ažuriranom zahtevu za povlačenjem, pošto je napravljena promena na toj liniji.

Dodavanje komitova na postojeći zahtev za povlačenjem ne okida notifikaciju, tako da nakon što Toni gurne svoje ispravke, odlučuje da ostavi komentar kako bi obavestio vlasnika projekta da je napravio zahtevanu promenu.

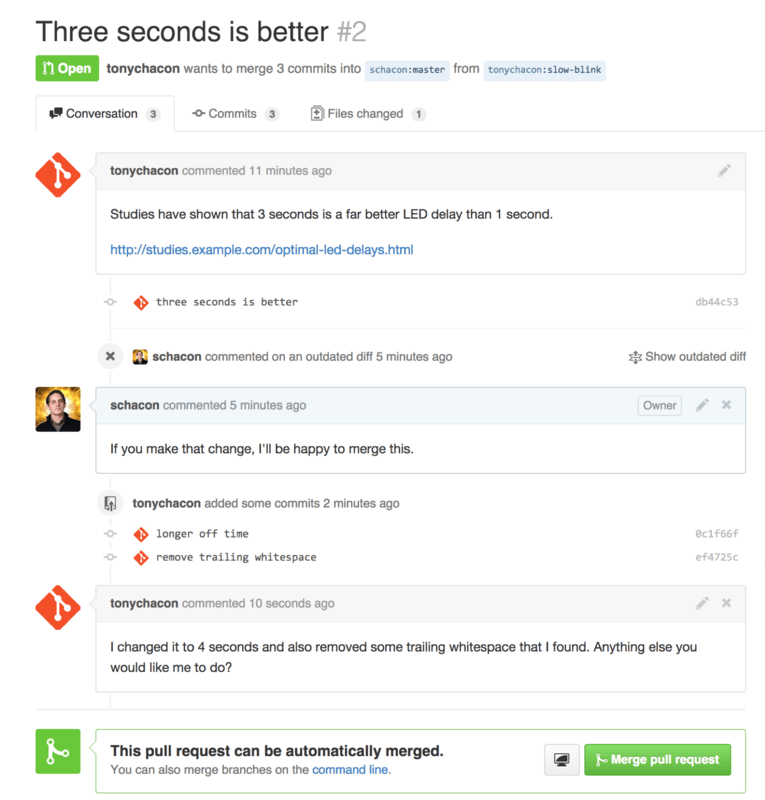


Figure 96. Konačni zahtev za povlačenjem

Zanimljiva stvar koju možete da primetite je da ako kliknete na tab Files Changed na ovom zahtevu za povlačenjem, dobićete "ujedinjenu" razliku — drugim rečima, ukupnu nagomilanu razliku koja bi bila uvedena u vašu glavnu granu ako bi se ova tematska grana spojila sa njom. U poređenju sa git diff, to je u suštini ono što vam automatski pokaže git diff master...<grana> za granu na kojoj je zahtev za povlačenjem baziran. Pogledajte [Kako utvrditi šta je uvedeno](https://git-scm.com/book/sr/v2/ch00/_what_is_introduced) za više o ovoj vrsti razlike.

Druga stvar koju ćete primetiti je to da GitHub proverava da li se zahtev za povlačenjem glatko spaja i nudi dugme koje radi spoj na serveru. Ovo dugme se pojavljuje samo ako imate pristup pisanju u repozitorijum i ako je trivijalni spoj moguć. Ako kliknete na njega, GitHub će uraditi "ne motaj napred" spoj, što znači da čak i ako bi spoj **mogao** da bude premotan unapred, ipak će se napraviti novi spojni komit.

Možete i jednostavno da povučete granu i obavite spoj lokalno, ako vam se tako više sviđa. Ako spojite ovu granu u master granu i gurnete to na GitHub, zahtev za povlačenjem će se automatski zatvoriti.

Ovo je osnovni tok rada koji koristi većina GitHub projekata. Prave se tematske grane, otvaraju se zahtevi za povlačenjem, vode se razgovori, eventualno se radi još malo na grani i na kraju se zahtev ili zatvori ili spoji.

|  |  |
| --- | --- |
| Note | Ne samo forkovi  Važno je primetiti da možete i da otvorite zahtev za povlačenjem između dve grane u istom repozitorijumu. Ako radite na nečemu s nekim i oboje imate pristup za pisanje projektu, možete da gurnete tematsku granu na repozitorijum i otvorite zahtev za povlačenjem u master granu tog istog projekta da biste pokrenuli pregled koda i diskusiju. Nema potrebe za forkovanjem. |

### Napredni zahtevi za povlačenjem

Sad kad smo pokrili osnove kontribuiranja projektu na GitHub-u, hajde da pogledamo nekoliko zanimljivih saveta i trikova o zahtevima za povlačenjem i kako možete da budete efikasniji dok ih koristite.

#### Zahtevi za povlačenjem kao zakrpe

Važno je da shvatite da mnogi projekti ne gledaju na zahteve za povlačenjem kao na seriju savršenih zakrpa koje treba da se primene tim redom, kao što to radi većina projekata baziranih na spisku adresa. Većina GitHub projekata gleda na grane za koje se zahteva povlačenje kao iterativne konverzacije o predloženoj promeni, koje kulminiraju u jedinstvenu razliku koja se prihvata spajanjem.

Ovo je važna razlika, jer se u opštem slučaju promene predlažu pre nego što se za kôd smatra da je savršen, što je mnogo ređi slučaj sa doprinosom baziranom na seriji zakrpa sa spiska adresa. Ovo omogućava da se najpre razgovara sa održavaocima tako da je dolazak dobrog rešenja uglavnom plod timskog rada zajednice. Kada se kôd predloži zahtevom za povlačenjem i održavaoci ili zajednica predloži promenu, serija zakrpa se generalno ne odmotava ponovo, već se razlika gurne kao novi komit na grani.

Na primer, ako se vratite nazad i opet pogledate [Konačni zahtev za povlačenjem](https://git-scm.com/book/sr/v2/ch00/_pr_final), primetićete da održavalac nije rebazirao svoj komit i poslao još jedan zahtev za povlačenjem. Umesto toga, dodao je nove komitove i gurnuo ih na postojeću granu. Na ovaj način, ako se vratite nazad i pogledate ovaj zahtev za povlačenjem u budućnosti, moći ćete lako da nađete kontekst zbog koga su odluke napravljene. Ako kliknite na dugme Merge na sajtu, namerno se pravi spojni komit koji referncira na zahtev za povlačenjem kako bi bilo lako da se vratite nazad i istražite originalnu diskusiju, ako to bude bilo potrebno.

#### Kako održati korak sa uzvodnim granama

Ako vaš zahtev za povlačenjem postane zastarao ili se iz nekog drugog razloga ne spaja glatko, treba da popravite to da bi održavalac mogao lako da načini spoj. GitHub će vam ovo testirati i obavestiće vas o tome na dnu svakog zahteva za povlačenjem da li je spoj trivijalan ili ne.

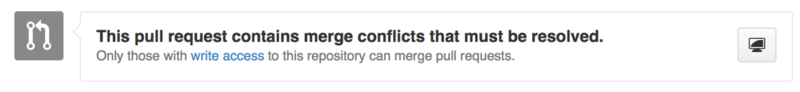


Figure 97. Zahtev za povlačenjem se ne spaja glatko.

Ako vidite nešto kao [Zahtev za povlačenjem se ne spaja glatko.](https://git-scm.com/book/sr/v2/ch00/_pr_fail), treba da popravite svoju granu tako da se pretvori u zeleno i da održavalac ne mora da radi višak posla.

Imate dva glavna izbora da uradite ovo. Možete ili da rebazirte svoju granu na vrh koje god grane da je odredišna (obično master grana repozitorijuma koji ste forkovali) ili da spojite odredišnu granu u svoju granu.

Većina developera na GtHub-u će izabrati drugu opciju, iz istih razloga koje smo prešli u prethodnom odeljku. Ono što je važno je istorija i konačni spoj, tako da vam rebaziranje ne daje mnogo toga sem nešto čistije istorije, a **mnogo** je podložnije greškama i teže.

Ako želite da se spojite u odredišnu granu da bi vaš zahtev za povlačenjem bio spojiv, treba da dodate prvobitni repozitorijum kao novi rimout, pribavite (fetch) podatke sa njega, spojite glavnu granu tog repozitorijuma u svoju tematsku granu, sredite sve ako dođe do problema i na kraju gurnete to nazad na istu granu za koju ste otvorili zahtev za povlačenjem.

Na primer, recimo da u primeru tinychacon koji smo koristili malopre, prvobini autor napravi promenu koja stvara konflikt sa zahtevom za povlačenjem. Hajde da prođemo kroz te korake.

$ git remote add upstream https://github.com/schacon/blink **(1)**

$ git fetch upstream **(2)**

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (3/3), done.

Unpacking objects: 100% (3/3), done.

remote: Total 3 (delta 0), reused 0 (delta 0)

From https://github.com/schacon/blink

\* [new branch] master -> upstream/master

$ git merge upstream/master **(3)**

Auto-merging blink.ino

CONFLICT (content): Merge conflict in blink.ino

Automatic merge failed; fix conflicts and then commit the result.

$ vim blink.ino **(4)**

$ git add blink.ino

$ git commit

[slow-blink 3c8d735] Merge remote-tracking branch 'upstream/master' \

into slower-blink

$ git push origin slow-blink **(5)**

Counting objects: 6, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (6/6), done.

Writing objects: 100% (6/6), 682 bytes | 0 bytes/s, done.

Total 6 (delta 2), reused 0 (delta 0)

To https://github.com/tonychacon/blink

ef4725c..3c8d735 slower-blink -> slow-blink

1. Dodaje prvobitni repozitorijum kao rimout sa imenom upstream
2. Pribavlja najnoviji rad sa tog rimouta
3. Spaja glavnu granu u vašu tematsku granu
4. Rešava konflikte koji su nastali
5. Gura nazad na istu tematsku granu

Kada to uradite, zahtev za povlačenjem će automatski biti ažuriran i ponovo proveren da bi se ustanovilo da li se glatko spaja.

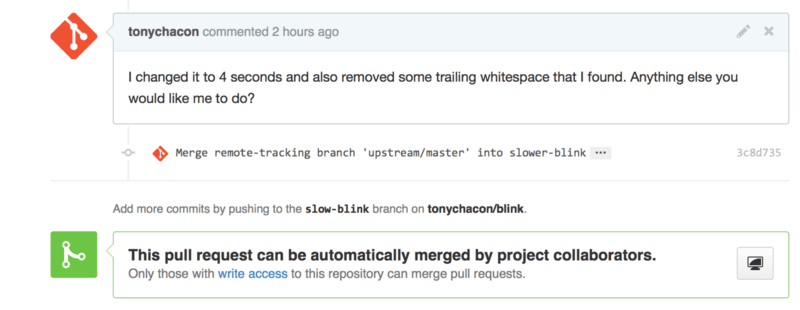


Figure 98. Zahtev za povlačenjem se sada spaja glatko.

Jedna od odličinih stvari kod Gita je to što ovakve stvari možete da radite konstantno. Ako imate veoma dugi projekat, možete jednostavno da spajate sa odredišnih grana iznova i iznova a da samo treba da se bavite konfliktima koji nastaju od trenutka kada ste poslednji put spojili, što čini proces veoma podesnim za rukovanje.

Ako apsolutno morate da rebazirate granu da biste je počistili, naravno, to možete uraditi, ali se strogo preporučuje da ne forsirate guranje preko grane na kojoj je zahtev za povlačenjem već otvoren. Ako su ga drugi ljudi povukli i radili nešto nad njim, dolazite do gomile problema koji su opisani u [Opasnosti prilikom rebaziranja](https://git-scm.com/book/sr/v2/ch00/_rebase_peril). Umesto toga, gurnite rebaziranu granu na novu granu na GitHub-u i otvorite potpuno novi zahtev za spajanjem koji referencira stari, a onda zatvorite prvobitni.

#### Reference

Vaše sledeće pitanje je s pravom "Kako da referenciram stari zahtev za povlačenjem?". Ispostavlja se da ima ogroman broj načina da referencirate skoro sve što pišete na GitHub-u.

Počnimo od toga kako da referencirate drugi zahtev za povlačenjem ili tiket. Svi zahtevi za povlačenjem i tiketi imaju svoje brojeve i jedinstveni su u sklopu projekta. Na primer, ne možete imati zahtev za povlačenjem #3 i tiket #3. Ako želite da referencirate zahtev za povlačenjem ili tiket sa bilo kog drugog, možete jednostavno da stavite #<broj> u bilo kom komentaru ili opisu. Možete i da budete specifičniji ako tiket ili zahtev za povlačenjem živi negde drugde; pišite korisničko-ime#<broj> ako referencirate tiket ili zahtev za povlačenjem u forku repozitorijuma koga se nalazite, ili korisničko-ime/repozitorijum#<broj> da referencirate nešto iz drugog repozitorijuma.

Pogledajmo primer. Recimo da samo rebazirali granu u prethodnom primeru, napravili novi zahtev za povlačenjem za nju, i sada želimo da referenciramo stari zahtev za povlačenjem sa novog. Takođe želimo da referenciramo tiket u forku repozitojuma i tiket u potpuno drugom projektu. Možemo da popunimo opis baš kao što je to učinjeno u [Reference u zahtevu za povlačenjem.](https://git-scm.com/book/sr/v2/ch00/_pr_references).

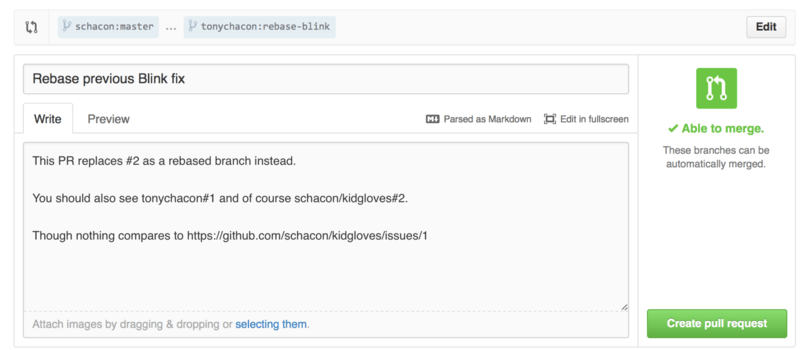


Figure 99. Reference u zahtevu za povlačenjem.

Kada pošaljemo ovaj zahtev za povlačenjem, videćemo da se sve to renderovalo kao [Cross references rendered in a Pull Request.](https://git-scm.com/book/sr/v2/ch00/_pr_references_render).

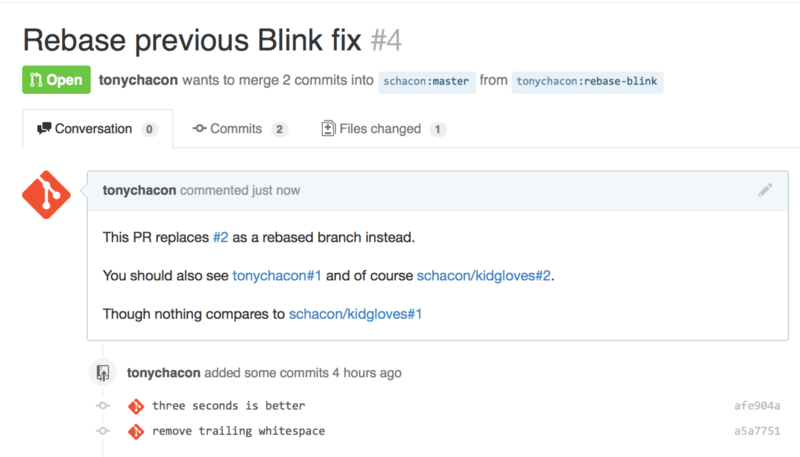


Figure 100. Cross references rendered in a Pull Request.

Primetite da je pun GitHub URL koji smo uneli smanjen samo na neophodne informacije.

Ako se sada Toni vrati nazad i zatvori prvobitni zahtev za povlačenjem, vidimo to da njegovim pominjanjem u novom, Git automatski kreira trekbek događaj (trackback event) u vremenskoj liniji zahteva za povlačenjem. Ovo znači da će svako ko poseti ovaj zahtev za povlačenjem i vidi da je zatvoren lako moći da dođe do onog kojim je zaenjen. Link će izgledati nekako kao na [Cross references rendered in a Pull Request.](https://git-scm.com/book/sr/v2/ch00/_pr_closed).

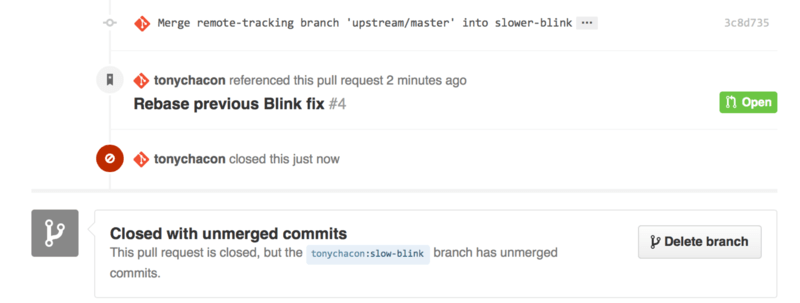


Figure 101. Cross references rendered in a Pull Request.

Pored brojeva tiketa, možete da referencirate i specifičan komit pomoću SHA-1. Morate da specificirate ceo SHA-1 sa 40 karaktera, ali ako GitHub vidi to u komentaru, linkovaće ga direktno na komit. Opet, možete da referncirate komitove u forkovima ili drugim repozitorijuma na isti način kao sa tiketima.

### Markdaun

Linkovanje drugih tiketa je samo početak zanimljivih stvari koje možete da radite sa skoro svakim tekstualnim poljem na GitHub-u. U opisima tiketa i zahtevima za spaanjem, komentarima, komentarima u kôdu i na mnogim drugim mestima, možete da koristite nešto što se zove Makrdaun sa ukusom GitHub-a (GitHub Flavored Markdown). Markdaun je kao pisanje običnog teksta koji se zatim renderuje sa stilovima.

Pogledajte [Primer Markdauna kako je napisan i kako je renderovan.](https://git-scm.com/book/sr/v2/ch00/_example_markdown) za primer kako se komentari ili tekst mogu napisati a onda renderovati Markdaunom.

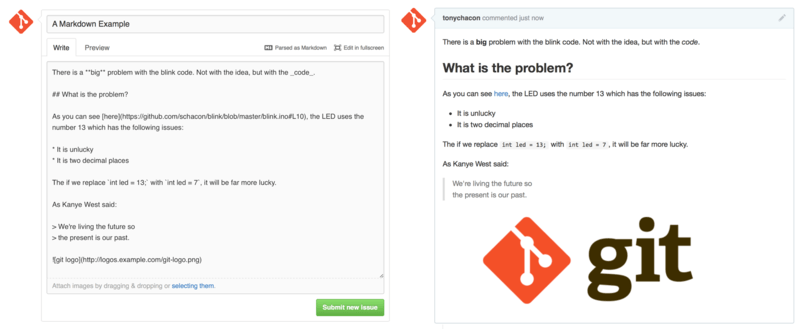


Figure 102. Primer Markdauna kako je napisan i kako je renderovan.

#### Markdaun sa ukusom GitHub-a

Markdaun sa ukusom GitHub-a dodaje još stvari koje možete da uradite pored osnovne Markdaun sintakse. Svi oni mogu biti veoma korisni kada pravite komentare ili opise za zahteve za povlačenjem ili tikete.

###### **LISTA ZADATAKA**

Prva veoma korisna odlika GitHub-ovog specifičnog Markdauna, pogotovo kod zahteva za povlačenjem, jeste lista zadataka. Lista zadataka je lista čekboksova stvari koje želite da budu urađene. Kada ih stavite u tiket ili zahtev za povlačenjem, to obično znači da želite da sve to bude urađeno pre nego što ih obeležite kao gotove.

Možete da napravite listu zadataka ovako:

- [X] Write the code

- [ ] Write all the tests

- [ ] Document the code

Ako ovo uključimo u opis na zahtevu za povlačenjem ili u opis tiketa, videćemo da se renderuje kao na [Lista zadataka renderovana u Markdaun komentaru.](https://git-scm.com/book/sr/v2/ch00/_task_lists).

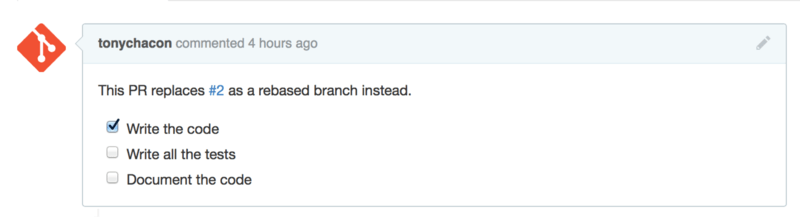


Figure 103. Lista zadataka renderovana u Markdaun komentaru.

Ovo se često koristi u zahtevima za povlačenjem radi ukazivanja na sve ono što biste želeli da se obavi na grani pre nego što zahtev za povlačenjem bude spreman za spoj. Ono što je ovde stvarno super je to što možete lako da kliknete na čekboksove da apdejtujete komentar — ne morate da editujete Markdaun direktno da biste obeležili zadatke kao obavljene.

Štaviše, GitHub će potražiti liste zadataka na tiketima i zahtevima za povlačenjem i pokazati ih kao metapodatke na stranicama koje ih izlistavaju. Na primer, ako imate zahtev za povlačenjem sa zadacima i bacite pogled na stranicu koja predstavlja listu svih zahteva za povlačenjem, videćete dokle se stiglo sa radom. Ovo pomaže ljudima da razbiju zahteve za povlačenjem u manje zadatke i pomaže drugim ljudima da prate napredovanje grane. Primer ovoga vidi se na [Sažetak liste zadataka u listi zahteva za povlačenjem.](https://git-scm.com/book/sr/v2/ch00/_task_list_progress).



Figure 104. Sažetak liste zadataka u listi zahteva za povlačenjem.

Ovo je neverovatno korisno kada otvorite zahtev za povlačenjem rano i koristite ga da pratite napredak kroz implementaciju.

###### **ISEČCI KODA**

Možete da dodate i isečke koda u komentare. Ovo je posebno korisno kada želite da predstavite nešto što možete da probate da uradite pre nego što ga stvarno implementirate kao komit na grani. Ovo se takođe često koristi da se doda primer koda koji ne radi ili šta bi sve ovaj zahtev za povlačenjem mogao da implementira.

Da biste dodali isečak koda, treba da ga ograničite u kontra-apostrofima (```).

```java

for(int i=0 ; i < 5 ; i++)

{

System.out.println("i is : " + i);

}

```

Ako dodate ime jezika kao što smo gore učinili sa java, GitHub će probati i da sintaksički oboji isečak. U slučaju gornjeg primera, render će izgledati kao na [Renderovan isečak koda.](https://git-scm.com/book/sr/v2/ch00/_md_code).



Figure 105. Renderovan isečak koda.

###### **CITIRNJE**

Ako odgovarate na manji deo većeg komentara, možete da selektivno citirate komentar tako što ćete linije početi karakterom >. Zapravo, ovo je toliko korisno i toliko često da postoji prečica sa tastature. Ako obeležite tekst u komentaru na koji želite da direktno odgovorite i pritisnite dirku r, u oblasti za kucanje komentara će se automatski dodati taj tekst, citiran.

Citati izgledaju nekako ovako:

> Whether 'tis Nobler in the mind to suffer

> The Slings and Arrows of outrageous Fortune,

How big are these slings and in particular, these arrows?

Kada se renderuje, komentar će izgledai kao na [Primer renderovanog citata.](https://git-scm.com/book/sr/v2/ch00/_md_quote).

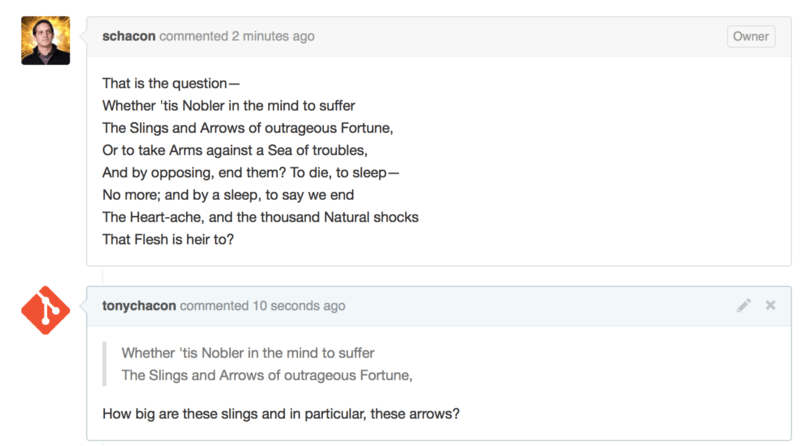


Figure 106. Primer renderovanog citata.

###### **EMOĐI**

Za kraj, možete da koristite i emođije u komentarima. Ovo se zapravo često koristi u komentarima koje vidite na mnogim GitHub-ovim tiketima i zahtevima za povlačenjem. Postoji čak i pomoćnik za emođije na GitHub-u. Ako kucate komentar i počnete sa karakterom :, autokomplit će vam pomoći da nađete ono što tražite.

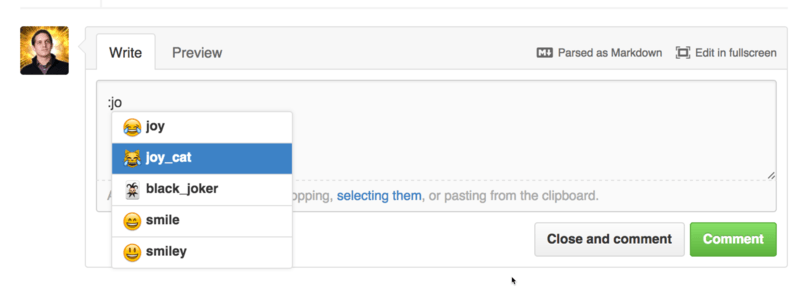


Figure 107. Autokomplit za emođije u akciji.

Emođiji imaju oblik :<ime>: bilo gde u komentaru. Na primer, možete da napišete nešto ovako:

I :eyes: that :bug: and I :cold\_sweat:.

:trophy: for :microscope: it.

:+1: and :sparkles: on this :ship:, it's :fire::poop:!

:clap::tada::panda\_face:

Kada se renderuje, izgledaće otprilike kao na [Komentar sa puno emođija.](https://git-scm.com/book/sr/v2/ch00/_md_emoji).

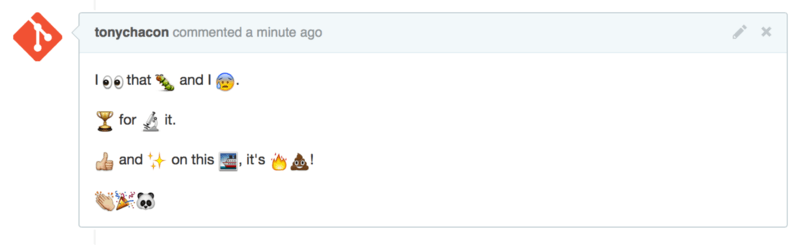


Figure 108. Komentar sa puno emođija.

Ovo nije nešto mnogo korisno, ali dodaje element zabave i emocije medijumu na kome je inače teško iskazati emocije.

|  |  |
| --- | --- |
| Note | Zapravo postoji veliki broj veb servisa koji koriste emođi karaktere ovih dana. Odličan spisak na koji se možete pozvati da nađete emođi koji oslikava ono što želite da kažete nalazi se na:  [http://www.emoji-cheat-sheet.com](http://www.emoji-cheat-sheet.com/) |

###### **SLIKE**

Ovo tehnički nije Markdaun sa ukusom GitHub-a, ali je neverovatno korisno. Pored dodavanja linkova ka slikama u Markdaun komentarima, što možete biti teško prilikom nalaženja i embedovanja URL-ova, GitHub vam omogućava i da prevučete i pustite slike na oblasti za tekst i da ih tako embedujete.

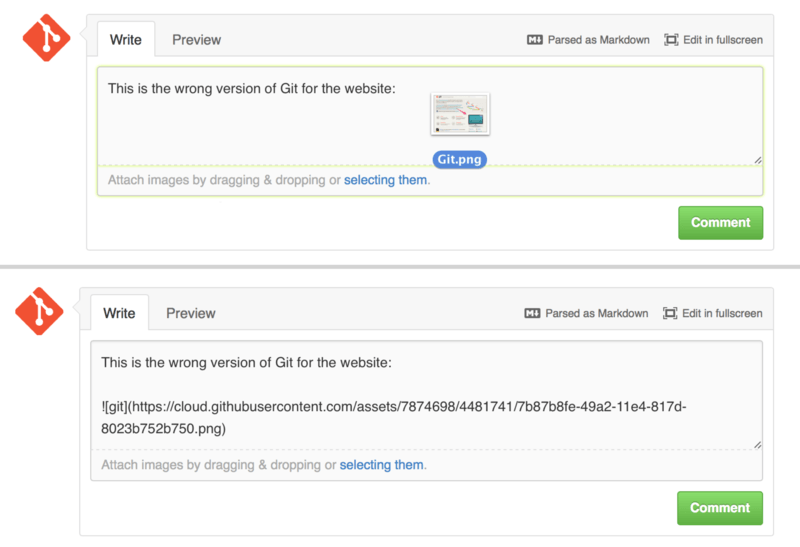


Figure 109. Prevucite i pustite slike da biste ih aploudovali i automatski embedovali.

Ako pogledate [Reference u zahtevu za povlačenjem.](https://git-scm.com/book/sr/v2/ch00/_pr_references), videćete mali tekst Parsed as Markdown. Klikom na to će vam se otvoriti veliki pano sa svime što možete uraditi sa Markdaunom na GitHub-u.

# 6.3 GitHub - Održavanje projekta

## Održavanje projekta

Sada kada znamo kako da doprinesemo projektu, hajde da pogledamo drugu stranu: kreiranje, održavanje i administracija sopstvenog projekta.

### Kreiranje novog repozitorijuma

Kreirajmo novi repozitorijum na kome ćemo deliti naš projekat. Počnite klikom na dugme New repositoriyu gornjem desnom uglu komandne table, ili na dugme + u alatnoj traci pored vašeg korisničkog imena kao što se vidi na [Padajući meni New repository.](https://git-scm.com/book/sr/v2/ch00/_new_repo_dropdown).

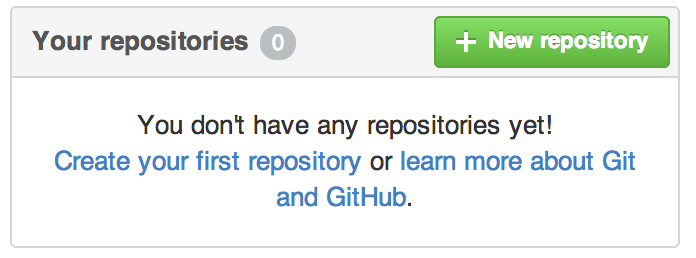


Figure 110. Oblast Your repositories.

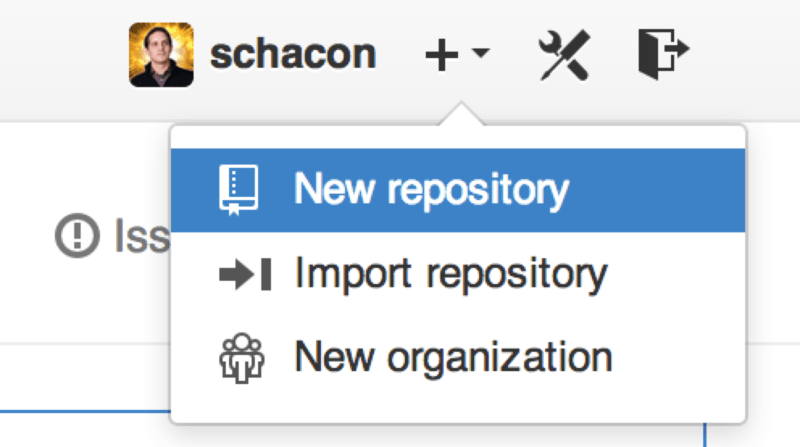


Figure 111. Padajući meni New repository.

Ovo vas vodi na formu New repository:

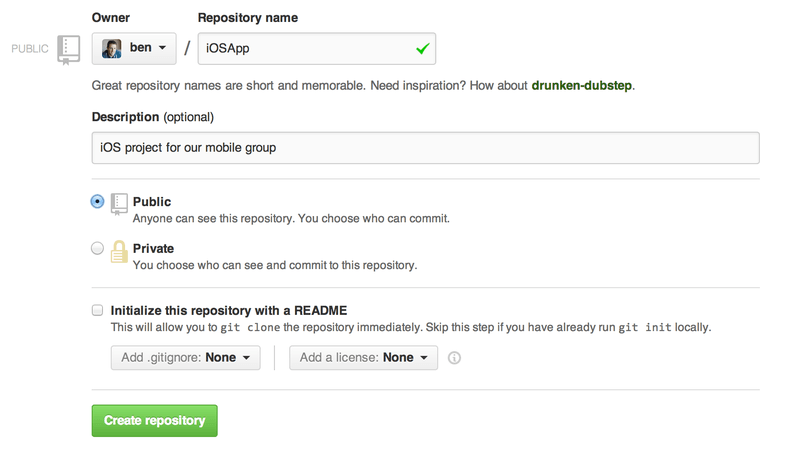


Figure 112. Forma New Repository.

Sve što ovde stvarno treba da uradite je da date ime projektu; ostala polja su potpuno neobavezna. Zasad, samo kliknite na dugme Create repository, i bum — imate nov repozitorijum na GitHub-u, nazvan <korisnik>/<ime\_projekta>.

Pošto još nemate nikakav kôd, GitHub će vam pokazati upustva za to kako da napravite potpuno nov Git repozitorijum, ili da ga povežete sa postojećim Git projektom. Nećemo se vraćati na ovo ovde; ako treba da se podsetite, bacite oko na [Osnove Gita](https://git-scm.com/book/sr/v2/ch00/_git_basics_chapter).

Sada kada je vaš projekat hostovan na GitHub-u, možete da date URL svakom s kim želite da podelite projekat. Svakom projektu na GitHub-u se može pristupiti preko HTTP-a https://github.com/<korisnik>/<ime\_projekta>, i preko SSH-a github@github.com:<korisnik>/<ime\_projekta>. Git može da pribavlja sa URL-ova i gura na njih, ali se kontrola pristupa vrši na osnovu akreditacije korisnika koji im pristupa.

|  |  |
| --- | --- |
| Note | Često se preferira deljenje URL-a baziranog na HTTP-u za javni projekat, pošto korsnik ne mora da ima GitHub nalog da bi mu pristupio za kloniranje. Korisnici će morati da imaju nalog i aploudovan SSH ključ da pristupe vašem projektu ako im date SSH URL. HTTP URL je takođe isti URL koji će prekopirati u brauzer da bi pogledali projekat tamo. |

### Dodavanje kolaboratora

Ako radite sa drugim ljudima i želite da im date pristup za komitove, treba da ih dodate kao kolaboratore. Ako Ben, Džef i Luiz svi imaju naloge na GitHub-u, i želite da im date pristup za guranje na vaš repozitorijum, možete da ih dodate na svoj projekat. To će im dati pristup za guranje, što znači da će moći i da čitaju podatke i da ih pišu na projektu i na Git repozitorijumu.

Kliknite na link Settings na dnu trake sa desne strane.

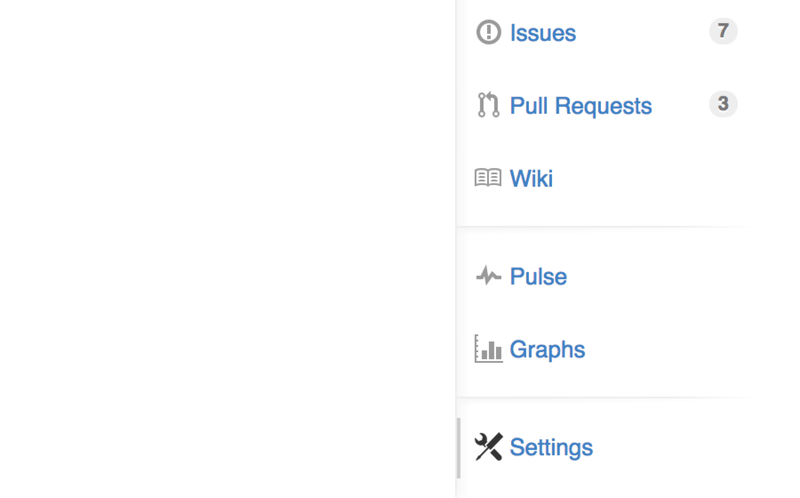


Figure 113. Link za podešavanje repozitorijuma

Onda odaberite Collaborators iz menija sa leve strane. Zatim samo ukucajte korisničko ime u polje za unos, i kliknite Add collaborator. Ovo možete da ponovote onoliko puta koliko je potrebno da biste dali pristup svima kome želite. Ako treba da opozovete pristup, samo kliknite na X sa desne strane odgovorajućeg reda.

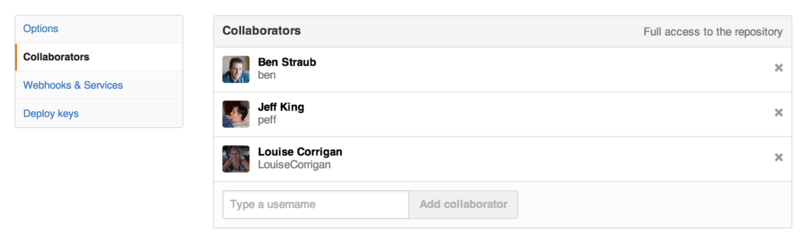


Figure 114. Kolaboratori repozitorijuma.

### Upravljenje zahtevima za povlačenjem

Sada kada imate projekat sa nekim kodom i možda čak nekoliko kolaboratora koji imaju pristup guranju promena, hajde da vidimo šta da radite kada dobijete zahtev za povlačenjem.

Zahtevi za povlačenjem mogu da dođu ili sa grane u forku vašeg repozitorijuma ili da dođu sa druge grane iz istog repozitorijuma. Jedina razlika je u tome što oni iz forka obično pripadaju drugim ljudima gde ne možete da gurnete na njihovu granu i oni ne mogu na vašu, dok kod internih zahteva za povlačenjem u opštem slučaju oba lica imaju pristup grani.

Za ove primere, pretpostavićemo da ste vi tonychacon i da ste napravili novi projekat sa Arduino kodom koji se zove fade.

#### Mejl obaveštenja

Neko dođe, napravi promene u kodu i pošalje vam zahtev za povlačenjem. Treba da dobijete mejl koji vas obaštava o novom zahtevu za povlačenjem i trebalo bi da izgleda nešto kao [Obaveštenje o novom zahtevu za povlačenjem putem mejla.](https://git-scm.com/book/sr/v2/ch00/_email_pr).

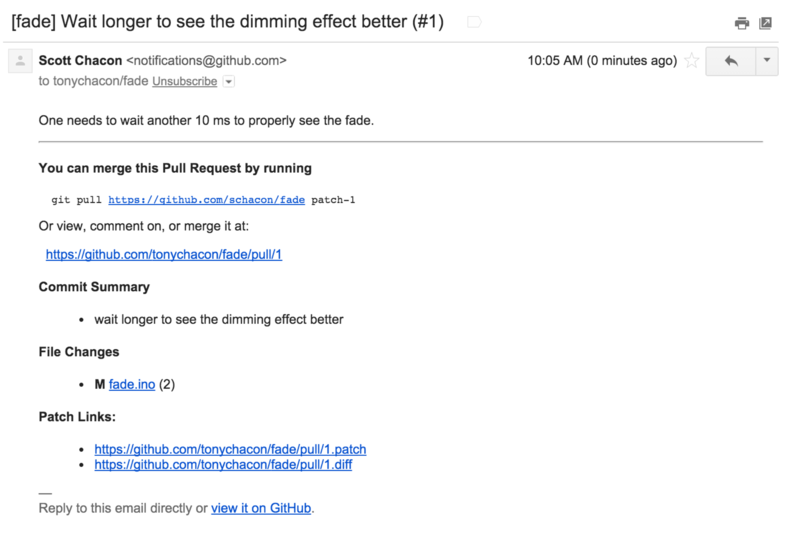


Figure 115. Obaveštenje o novom zahtevu za povlačenjem putem mejla.

Postoji nekoliko stvar koje treba da primetite u ovom mejlu. Daje vam kratku statistiku o razlikama — listu fajlova koje su se promenile u zahtevu za povlačenjem i za koliko. Daje vam link ka zahtevu za povlačenjem na GitHub-u. Daje vam i nekoliko URL-ova koje možete da iskoristite iz komandne linije.

Ako primetite liniju gde kaže git pull <url> patch-1, ovo je jednostavan način da se spojite na udaljenu granu a da ne dodajete ručno rimout. Brzo smo prešli ovo u [Čekautovanje udaljenih grana](https://git-scm.com/book/sr/v2/ch00/_checking_out_remotes). Ako želite, možete da kreirate i skočite na tematsku granu i onda pokrenete ovu komandu da spojite promene sa zahteva za povlačenjem.

Drugi zanimljivi URL-ovi su .diff i .patch, koji, kao što možete da pretpostavite, daju ujedinjenu razliku i zakrpa-verzije zahteva za povlačenjem. Tehnički možete da spojite zahtev za povlačenjem nekako ovako:

$ curl http://github.com/tonychacon/fade/pull/1.patch | git am

#### Kolaboracija na zahtevu za povlačenjem

Kako smo pokrili u [Tok u GitHub-u](https://git-scm.com/book/sr/v2/ch00/_github_flow), sada možete da porazgovarate sa osobom koja je otvorila zahtev za povlačenjem. Možete da komentarišete na specifične linije koda, da komentarišete cele komitove ili pak na čitav zahtev za povlačenjem, koristeći GitHub-ovu Markdaun sintaksu.

Svaki put kada neko drugi komentariše na zahtev za povlačenjem, dobićete obaveštenjem putem mejla kako biste znali da se dešava aktivnost. Svaki od njih će imati link na zahtevu za povlačenjem gde se aktivnost događa, a možete i da ukucate odgovor direktno u mejlu i tako objavite svoj komentar.

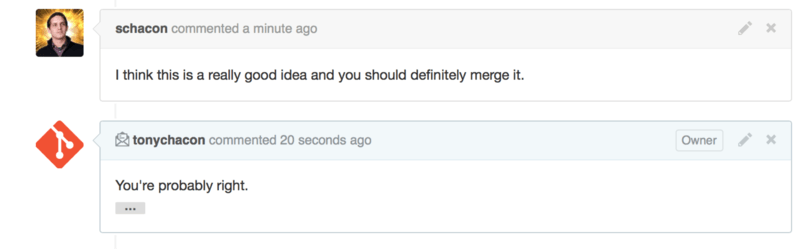


Figure 116. Responses to emails are included in the thread.

Kada kôd bude sređen i kada poželite da spojite, možete ili da povučete kôd i spojite ga lokalno, ili da iskoristite git pull <url> <branch> sintaksu koju smo videli ranije, ili da dodate fork kao rimout i onda pribavite promene sa njega i spojite.

Ako je spoj trivijalan, možete i samo da pritisnete dugme Merge na GitHub vebsajtu. Ovo će uraditi "ne motaj napred" spoj, što znači da će napraviti spojni komit čak i ako je spoj motanjem napred moguć. Ovo znači da će nastati novi spojni komit uvek kada pristisnete Merge dugme. Kao što vidite u [Dugme Merge i instrukcije za spajanje zahteva za povlačenjem ručno.](https://git-scm.com/book/sr/v2/ch00/_merge_button), GitHub vam daje sve ove inforamcije ako kliknete na hint link.

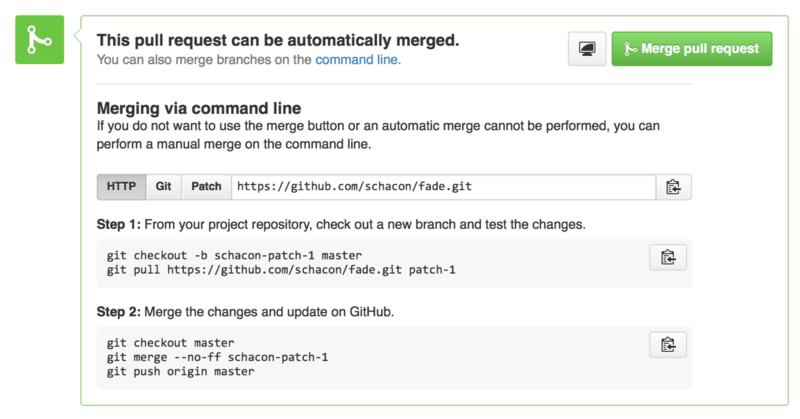


Figure 117. Dugme Merge i instrukcije za spajanje zahteva za povlačenjem ručno.

Ako ste odlučili da ne želite da spojite, možete i da samo zatvorite zahtev za povlačenjem i osoba koja ga je otvorila će biti obaveštena.

#### Reovi za zahtev za povlačenjem

Ako baratate sa **puno** zahteva za povlačenjem i ne želite da dodate gomilu rimoutova ili da svaki put obrađujete jedan po jedan zahtev, postoji jedan dobar trik koji vam GitHub nudi. Ovo je pomalo napredniji trik i obradićemo detalje kada malo više budemo govorili o tome u [Refspek](https://git-scm.com/book/sr/v2/ch00/_refspec), ali može da bude jako koristan.

GitHub zapravo oglašava grane koje su zahtev za povlačenjem za repozitoijum kao vrstu pseudo-grane na serveru. Po podrazumevanim podešavanjima ne dobijate ih kada napravite klon, ali su tu na opskurantan način i možete da im pristupite jednostavno.

Da bismo demonstrirali ovo, koristićemo komandu niskog nivoa (često se koristi termin "plambing" komanda, o čemu ćete više pročitati u [Vodovod i porcelan](https://git-scm.com/book/sr/v2/ch00/_plumbing_porcelain)) koja se zove ls-remote. Ova komanda se generalno ne koristi u svakodnevnim Git operacijama ali je korisna da nam pokaže koje reference postoje na serveru.

Ako pokrenemo komandu spram "blink" repozitorijuma koji smo ranije koristili, dobićemo listu svih grana i tagova i drugih referenci u repozitorijumu.

$ git ls-remote https://github.com/schacon/blink

10d539600d86723087810ec636870a504f4fee4d HEAD

10d539600d86723087810ec636870a504f4fee4d refs/heads/master

6a83107c62950be9453aac297bb0193fd743cd6e refs/pull/1/head

afe83c2d1a70674c9505cc1d8b7d380d5e076ed3 refs/pull/1/merge

3c8d735ee16296c242be7a9742ebfbc2665adec1 refs/pull/2/head

15c9f4f80973a2758462ab2066b6ad9fe8dcf03d refs/pull/2/merge

a5a7751a33b7e86c5e9bb07b26001bb17d775d1a refs/pull/4/head

31a45fc257e8433c8d8804e3e848cf61c9d3166c refs/pull/4/merge

Naravno, ako ste u svom repozitorijumu i pokrenete git ls-remote origin ili koji god rimout želite da proverite, pokazaće vam nešto slično ovome.

Ako je repozitorijum hostovan na GitHub’u i imate otvorene zahteve za povlačenjem , dobićete reference sa prefiksom refs/pull/. Ovo su u suštini grane, ali pošto nisu pod refs/heads/ ne dobijate ih kada klonirate ili pribavite podatke sa servera — proces pribavljanja ih obično ignoriše.

Postoje dve reference za svaki zahtev za povlačenjem — ona koja se završava sa /head pokazuje na isti komit kao i poslednji komit u grani zahteva za povlačenjem. Zato, ako neko otvori zahtev za povlačenjem u našem repozitorijumu i njihova grana se zove bug-fix i pokazuje na komit a5a775, onda u **našem**repozitorijumu nećemo imati granu bug-fix (oišto je ona u njihovom forku), ali hoćemo imati pull/<pr#>/head koja pokazuje na a5a775. Ovo znači da možemo dosta lako da povučemo svaku granu zahteva za povlačenjem a da ne dodajemo gomilu rimoutova.

Sada, možete da uradite nešto kao pribavljanje reference direktno.

$ git fetch origin refs/pull/958/head

From https://github.com/libgit2/libgit2

\* branch refs/pull/958/head -> FETCH\_HEAD

Ovo govori Gitu, "Poveži se na origin rimout, i preuzmi ref sa imenom refs/pull/958/head." Git vas sa zadovoljstvom sluša i preuzima sve što vam treba da konstruišete taj ref, i stavlja pokazivač na komit koji želite pod .git/FETCH\_HEAD. Možete to da ispratite sa git merge FETCH\_HEAD u granu u koju želite da testirate, ali ta poruka spojnog komita izgleda pomalo čudno. Takođe, ako razmatrate **mnogo** zahteva za povlačenjem, ovo postaje dosadno.

Postoji način da pribavite sve zahteve za povlačenjem, i da ih održite aktuelnim svaki put kada se povežete na rimout. Otvorite .git/config u svom omiljenom editoru i potražite origin rimout. Treba da izgleda nekako ovako:

[remote "origin"]

url = https://github.com/libgit2/libgit2

fetch = +refs/heads/\*:refs/remotes/origin/\*

Linija koja počinje sa fetch = je "refspek". To je način na koji se mapiraju imena na rimoutu sa imenima u lokalnom .git direktorijumu. Ova linija govori Gitu, "stvari na rimoutu pod refs/heads treba da idu u moj lokalni repozitorijum pod refs/remotes/origin". Možete da modifikujete ovaj odeljak da dodate druge refspekove:

[remote "origin"]

url = https://github.com/libgit2/libgit2.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

fetch = +refs/pull/\*/head:refs/remotes/origin/pr/\*

Poslednja linija kaže Gitu, "Svi refovi koji izgledaju kao refs/pull/123/head treba da se lokalno smeste pod refs/remotes/origin/pr&123". Sada, ako sačuvate taj fajl, i uradite git fetch:

$ git fetch

# …

\* [new ref] refs/pull/1/head -> origin/pr/1

\* [new ref] refs/pull/2/head -> origin/pr/2

\* [new ref] refs/pull/4/head -> origin/pr/4

# …

Sada se svi udaljeni zahtevi za povlačenjem predstavljaju lokalno pomoću refova otprilike kao praćenje grana; moguće ih je samo čitati, i ažuriraju se kada obavite pribavljanje. Ovako je super-jednostavno da isprobate kôd sa zahteva za spajanejm lokalno:

$ git checkout pr/2

Checking out files: 100% (3769/3769), done.

Branch pr/2 set up to track remote branch pr/2 from origin.

Switched to a new branch 'pr/2'

Oni koji imaju oko sokolovo primetiće da je head na kraju rimouta deo refspeka. Postoji i refs/pull/#/merge ref na GitHub’ovoj strani, što predstavlja komit koji bi nastao ako pritisnete mergedugme na sajtu. Ovo vam omogućava da testirate spoj pre nego što uopšte pritisnete dugme.

#### Zahtevi za povlačenjem na zahtevima za povlačenjem

Možete da otvarate zahteve za povlačenjem koji kao odredište imaju glavnu ili master granu, ali možete i da ih otvorite za bilo koju drugu granu na mreži. Zapravo, odredište može biti i drugi zahtev za povlačenjem.

Ako vidite zahtev za povlačenjem koji ide u dobrom pravcu i imate ideju za promenu koja zavisi od njega ili nista sigurni da li to dobra ideja, ili jednostavno nemate pristup guranju izmena na odredišnoj grani, možete da otvorite zahtev za povlačenjem dirketno na njega.

Kada otvorite zahtev za povlačenjem, pojaviće se kutija na vrhu strane koja specificira ka kojoj grani želite da budete povučeni i od koje zahtevate da povučete. Ako pritisnete dugme Edit sa desne strane kutije, možete da promenite ne samo grane već i fork.

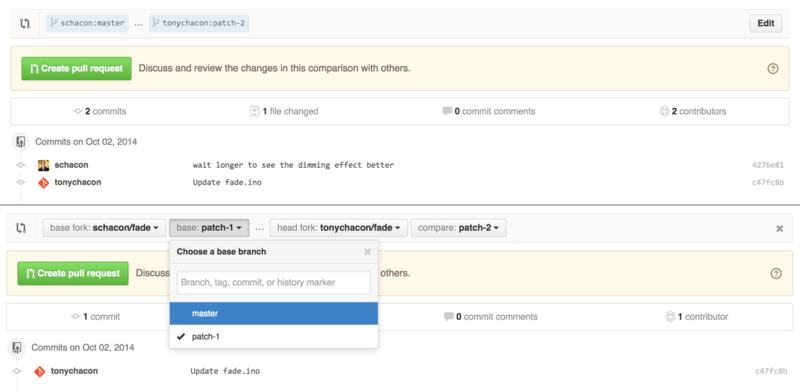


Figure 118. Manuelna promena odredišnog forka i odredišne grane zahteva za povlačenjem.

Ovde prilično lako možete da spojite svoju novu granu u drugi zahtev za povlačenjem ili drugi fork projekta.

### Pominjanje i obaveštenja

GitHub ima ugrađeen prilično dobar sistem obaveštenja koji je koristan kada imate pitanja ili vam treba povratna inforamcija od određenih osoba ili celog tima.

U bilo kom komentaru možete da ukucate karater @ i automatski će vam izlistati imena i korisnička imena ljudi koji su kolaboratori ili kontributori na projektu.

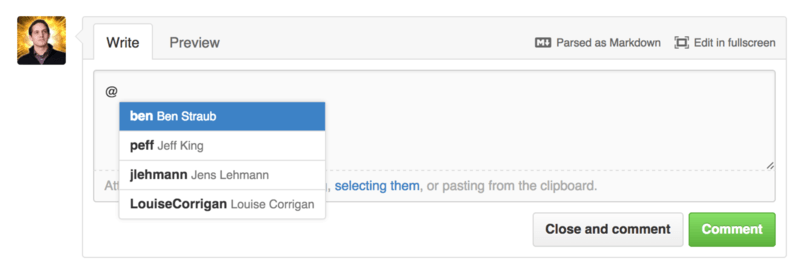


Figure 119. Počnite da kucate @ da biste pomenuli nekog.

Možete da pomenete i korisnika koji nije na listi, ali autokomplit obično čini stvari bržim.

Kada objavite komentar u kome pominjete korisnika, on ili ona će biti obavešteni. Ovo znači da ovo može biti veoma efikasan način pozivanja ljudi u razgovore. Ljudi na GitHub-u će često pozivati druge ljude iz tima da recenziraju zahteve za povlačenjem ili tikete.

Ako neko bude pomenut na zahtevu za povlačenjem ili tiketu, biće "pretplaćeni" i nastaviće da dobijaju notifikacije svaki put kada se dogodi neka aktivnost. Bićete pretplaćeni i na sve što sami otvorite, ako nadgledate (watch) repozitorijum ili ako komentarišete na nešto. Ako više ne želite da dobijate obaštenja, na stranici postoji dugme Unsubscribe na koje možete kliknuti i prestaćete da dobijate obaveštenja.

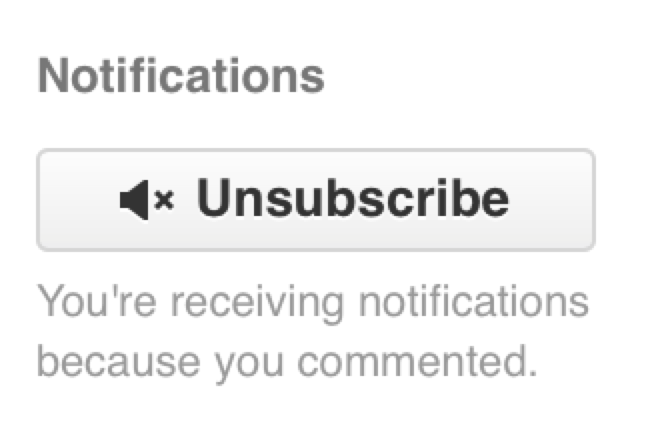


Figure 120. Prestanak praćenja zahtevâ za povlačenjem ili tiketâ.

#### Stranica sa obaveštenjima

Kada kažemo "obaveštenje" u kontekstu GitHub-a, mislimo na specifičan način pomoću koga GitHub pokušava da stupi u kontakt sa vama kada se dogode dođaji i postoji nekoliko različitih načina da ih konfigurišete. Ako odete na tab Notification center sa stranice za podešavanja, videćete neke od opcija koje imate.

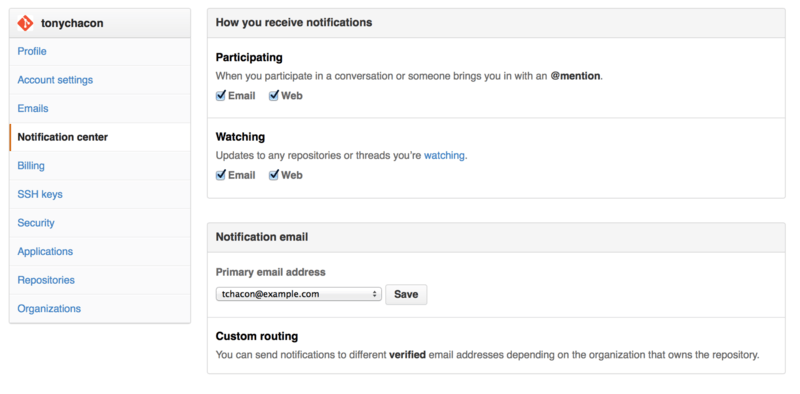


Figure 121. Opcije u centru za notifikacije.

Postoje dva izbora: dobijanje obaveštenja putem mejla ili putem veba i možete da izaberete jedno od njih, nijedno od njih ili oba kada aktivno učestvujete u stvarima i za aktivnosti na repozitorijumima koje nadgledate.

###### **VEB OBAVEŠTENJA**

Veb obaveštenja postoje samo na GitHub-u i možete da ih pogledate samo na GitHub-u. Ako vam je ova opcija selektirana u preferencama i okine se obaveštenje koje vam stigne, dobićete malu plavu tačku preko ikonice za obaveštenja u gornjem delu ekrana kao što se vidi na [Centar za obaveštenja.](https://git-scm.com/book/sr/v2/ch00/_not_center).

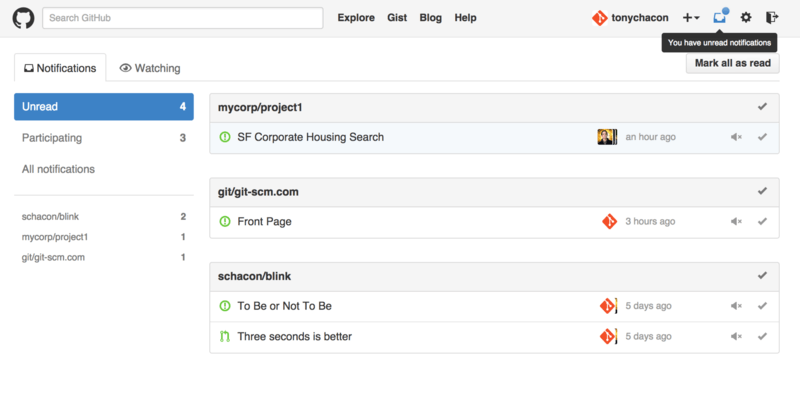


Figure 122. Centar za obaveštenja.

Ako kliknete tu, videćete listu svih stavki o kojima ste obavešteni, grupisanu po projektima. Možete da filtrirate obaveštanja po određenom projektu tako što ćete kliknuti na njegovo ime u traci sa leve strane. Možete i da potvrdite obaveštenje tako što ćete kliknuti na ikonicu za štikliranje pored bilo kog obaveštenja, ili da potvrdite sva obaveštenja u projektu tako što ćete klikinuti na ikonicu za štikliranje na vrhu grupe. Postoji i dugme za utišavanje obaveštenja pored svake ikonice za štikliranje na koje možete da kliknete ukoliko ne želite da dobijate dalja obaštenja u vezi s tom stavkom.

Svi ovi alati su veoma korisni za rukovanje velikim brojem obaveštenja. Mnogi iskusni korisnici će jednostavno ugasiti obaveštenja putem mejla u potpunosti i upravljati svim obaveštenjima preko ovog ekrana.

###### **OBAVEŠTENJA PUTEM MEJLA**

Obaveštenja putem mejla su još jedan način da rukujete obaveštenjima na GitHub-u. Ako vam je ova opcija uključena, dobijaćete mejl za svaku notifikaciju. Videli smo primer ovoga u [Komentari poslati kao mejl notifikacije.](https://git-scm.com/book/sr/v2/ch00/_email_notification) i [Obaveštenje o novom zahtevu za povlačenjem putem mejla.](https://git-scm.com/book/sr/v2/ch00/_email_pr). Mejlovi će se i namotavati kako valja, što je lepo ako koristite klijent koji podržava ovaj vid pregleda mejlova.

Postoji i poštena količina metapodataka ugrađena u zaglavlja mejlova koje vam GitHub šalje, što može da bude veoma korisno ako postavljate personalizovane filtere i pravila.

Na primer, ako pogledamo zaglavlja mejlova koji su poslati Toniju u mejlu koji je prikazan u [Obaveštenje o novom zahtevu za povlačenjem putem mejla.](https://git-scm.com/book/sr/v2/ch00/_email_pr), videćemo sledeće među ostalim informacijama:

To: tonychacon/fade <fade@noreply.github.com>

Message-ID: <tonychacon/fade/pull/1@github.com>

Subject: [fade] Wait longer to see the dimming effect better (#1)

X-GitHub-Recipient: tonychacon

List-ID: tonychacon/fade <fade.tonychacon.github.com>

List-Archive: https://github.com/tonychacon/fade

List-Post: <mailto:reply+i-4XXX@reply.github.com>

List-Unsubscribe: <mailto:unsub+i-XXX@reply.github.com>,...

X-GitHub-Recipient-Address: tchacon@example.com

Ovde ima nekoliko zanimljivih stvari. Ako želite da istaknete ili preusmerite mejlove za ovaj određeni projekat ili čak za zahtev za povlačenjem, informacija u Message-ID vam daje sve podatke u formatu <korisnik>/<projekat>/<vrsta>/<id>. Na primer, da je ovo bio tiket, u polju <type> bi pisalo issuesumesto pull.

Polja List-Pos i List-Unsubscribe znače da ako imate klijent za mejl koji razume ovo, možete lako da postujete na listu ili da ukinete pretplatu sa teme. To bi bilo isto kao klik na dugme Mute na veb verziji obaveštenjea ili Unsubscribe na samoj stranici zahteva za povlačenjem ili tiketa.

Takođe vredi napomenuti da ako imate uključena i mejl i veb obaveštenja i pročitate mejl verziju obaveštenja, veb verzija će biti označena kao pročitana takođe, ako su slike dozvoljene u vašem mejl klijentu.

### Posebni fajlovi

Ima nekoliko posebnih fajlova koje će GitHub primetiti ako postoje u repozitorijumu.

### README

Prvi je fajl README koji može biti u skoro svakom formatu koji GitHub prepoznaje kao prozu. Na primer, može da bude README, README.md, README.asciidoc, itd. Ako GitHub vidi README fajl u vašem izvornom kodu, renderovaće ga na početnoj stranici projekta.

Mnogi timovi koriste ovaj fajl da zapišu sve relevantne informacije o projektu za nekog ko je možda nov na repozitorijumu ili projektu. Obično se ovde pišu stvari kao:

* Čemu je projekat namenjen
* Kako ga konfigurisati i instalirati
* Primer koji pokazuje kako se koristi
* Licenca pod kojom je projekat dostupan
* Kako mu doprineti

Pošto će GitHub renderovati ovaj fajl, možete da ugradite slike ili linkove da bi dokument bio razumljiviji.

### CONTRIBUTING

Drugi poseban fajl koji GitHub prepoznaje je fajl CONTRIBUTING. Ako imate fajl CONTRIBUTING sa bilo kojom ekstenzijom, GitHub će prikazati [Otvaranje zahteva za povlačenjem kada postoji fajl CONTRIBUTING.](https://git-scm.com/book/sr/v2/ch00/_contrib_file) kad god neko otvori zahtev za povlačenjem.



Figure 123. Otvaranje zahteva za povlačenjem kada postoji fajl CONTRIBUTING.

Ideja je da specificirate određene stvari koje želite ili ne želite da budu uključene u zahtev za povlačenjem za vaš projekat. Na ovaj način ljudi će zapravo možda i pročitati smernice pre nego što otvore zahtev za povlačenjem.

### Admistracija projekta

Generalno nema mnogo administrativnih stvari koje možete da radite sa jednom projektom, ali postoji nekoliko stavki koje bi možda bile od interesovanja.

#### Menjanje podrazumevane grane

Ako koristite neku granu koja nije master kao podrazumevanu granu nad kojom želite da ljudi otvaraju zahteve za povlačenjem ili da je podrazumevano vide, možete to da promenite u podešavanjima repozitorijuma u tabu Options.

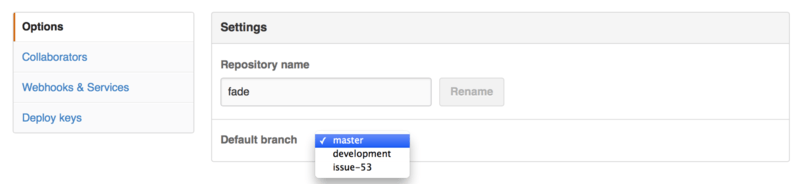


Figure 124. Promena podrazumevane grane za projekat.

Jednostavno promenite podrazumevanu granu u padajućem meniju i to će biti podrazumevana grana za sve veće operacije od tada pa nadalje, uključujući i to koja grana će se čekautovati po podrazumevanim podešavanjima kada neko klonira repozitorijum.

#### Prenos projekta

Ako biste želeli da svoj projekat preneste drugom korisniku ili organizaciji na GitHub-u, postoji opcija Transfer ownership na dnu istog taba Options na stranici sa podešavanjima vašeg repozitorijuma koja vam ovo omogućava.

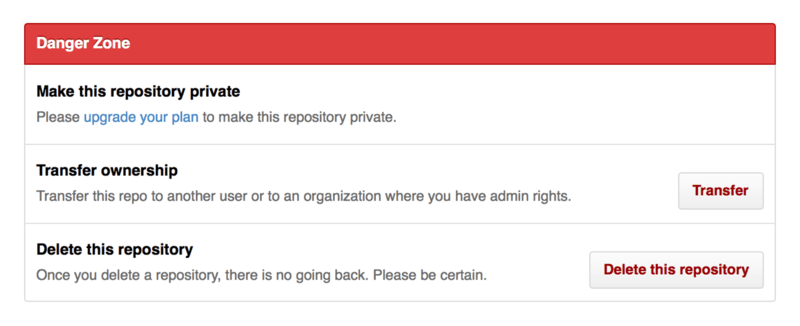


Figure 125. Prenos projekta drugom korisniku GitHub-a ili organizaciji.

Ovo je korisno ako napuštate projekat i neko drugi želi da ga preuzme, ili vam projekat postaje sve veći i želite da ga premestite u organizaciju.

Ovo ne samo da premešta repozitorijum zajedno sa svim pratiocima i zvezdama na drugo mesto, već podešava i preusmeravanje sa vašeg URL-a na novi. Preusmeravanje će raditi i prilikom kloniranja i pribavljanja iz Gita, ne samo na veb zahtevima.