# Tomo Krajina – Uvod u Git

### Git naredbe

Git config –global user.name – postavlja globalno korisničko ime git korisnika

Git config –global user.email

Git config –global color.ui auto – postavlja da se koriste različite boje u git komandnoj liniji

Git config –global merge.tool - postavlja default merge tool

.gitignore datoteka – u ovu datoteku se stavi sve što želimo da git ignorira

Kreiranje fajlova u linux-u: $echo "This is some text" > randomtext.txt

## Spremanje izmjena

$git diff – bez argumenata, prikazuje izmjene u projektu u odnosu na komitano stanje, tj. pregled nekomitanih izmjena – ne samo nekomitanih, već preciznije izmjena koje još nisu zabilježene u index. Ako se želi više linija oko izmjenjenog koda onda se koristi varijanta: $git diff –**U10**, prikazat će 10 linija oko izmjenjenog koda. Treba napomenuti da je osnovna namjena diff naredbe da može dati bili koju razliku, npr. Razliku između trenutne I remote grane ili razliku između dva čvora.

$git status – izmjene koje trebamo zabilježiti u index i kasnije commitati

Dodavanje cijelog direktorija u index: $git add naziv\_direktorija/\*, a sam direktorij se dodaje naredbom: $git add naziv\_direktorija/. Pojedini fajl u nekom direktoriju se dodaje u index: $git add dir\_name/file\_name

$git **reset** HEAD – neki\_fajl, – briše izmjene koje su dodane u indeks za neki\_fajl. Treba naglasiti da izmjene u fajlu ovom naredbom ostaju netaknute.

Ako se želi poništiti izmjene koje su već commitane, tj. aa se stanje fajla vrati onakvo kakvo je bilo nakon predzadnjeg commita, tada se treba pozvati naredba: $git **checkout** HEAD – neki\_fajl.txt

Ako imamo neki file u indeksu ili repozitoriju, on se iz repozitorja “skida” sljedećom naredbom:  
$git rm –cached ime\_datoteke. Nakon toga treba tu datoteku staviti u .gitignore, kako se više ne bi pojavljivala u $git status.

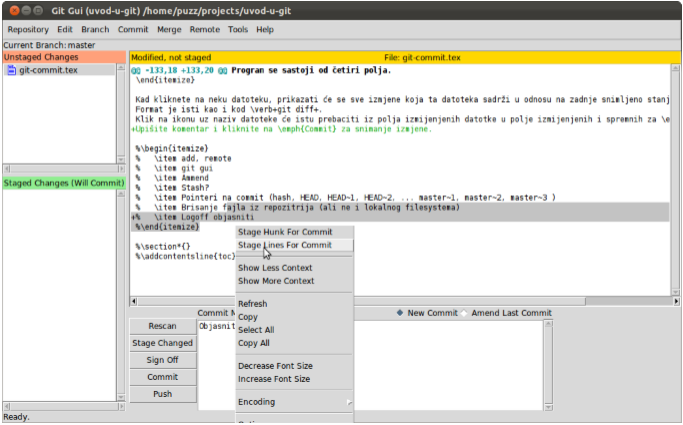
### Ispravljanje zadnjeg commita

U slučaju da se neka naknadna izmjena želi dodati u neki već postojeći commit, izmjeni se samo ta datoteka, njezine izmjene se dodaju u indeks I pozove se sljedeća naredba: $git commit –**amend** –m “komentar”. Ta će izmjena biti dodana u zadnji commit, kojemu će se istovremeno komentar promijeniti u komentar koji smo dodali sa tim commitom.

### Git GUI

Poziva se sa: $git gui

Sa git gui možemo commitati same neke dijelove fajla kao na donjoj slici:



### Clean

$git clean –n, daje spisak fajlova koji će se obrisati

$git clean –f, briše sve nepotrebne fajlove iz .gitignore + fajlove koji nisu prijavljeni u indexi

Moguće je izbrisati i samo određene datoteke ili samo određeni folder sa: $git clean –f – ime\_direktorija. Npr. Ako želimo izbrisati samo neki fajl u folderu dokumenti tada je naredba:  
$git clean –f – dokumenti/ime\_fajla.txt. Ako međutim žeimo izbrisati cijeli sadržaj nekog direktorija, tada je naredba: $git clean –f – dokumenti/

## Grananje

$git branch – popis grana projekta

$git **branch** ime\_nove\_grane – dodavanje nove grane

$git **checkout** ime\_grane – prebacivanje na drugu granu

Prebacivanje sa grane na granu najbolje je napraviti nakon što smo na grani koju napuštamo commitali sve izmjene.

### Brisanje grane

$git branch **–D** naziv\_grane

Brisanjem grane, ne brišemo i njezime commitove…

### Preuzimanje datoteka iz druge grane(iz trenutne grane)

Dakle prebacimo se u granu u koju želimo preuzeti datoteke i izvršimo naredbu:

$git checkout ime\_grane – ime\_datoteke ime\_datoteke2…

?čini se da kad se preuzme fajl iz druge grane da se pregazi istoimeni fajl u treutnoj grani…

## Preuzimanje izmjena iz jedne grane u drugu

### Git merge

Sve izmjene se iz navedene grane prebace u trenutnu: $git **merge** ime\_grane\_iz\_koje\_se\_merga

Nakon merganja, automatski se dodaje novi commit čvor u granu u koju smo povukli izmjene iz druge grane, tj. te se izmjene ne moraju posebno commitati

U slučaju da se desio konfikt prilikom merganja, tada se ručno editira fajl koji je izazvao konflikt, te se nakon toga ručno commitaju izmjene.

### Fast forward

(vrijedi za slučaj kada su se grane u jednom trenutku razdvojile, te se jedna grana nastavila razvijati, dok je u drugoj grani razvoj stao. Napokon se želi u granu koja je mirovala, pruzeti izmjene iz “dinamične” grane. )

To je slučaj merga kada se iz jedne grane preuzima cijela povijest u drugu granu, tj. ako su se one u jednom trenutku razdvojile, grana u koju se izmjene preuzimaju, preuzet će I sve čvorove koji su u međuvremenu nastali u drugoj grani. Fast forward prilikom merga se može izbjeći ako se koristi naredba: $git merge –**no-ff** ime\_grane\_za\_preuzimanje, tj. Sa ovom opcijom će biti preuzet samo zadnji commit čvor.

### Rebase

### Cherry-pick

Određeni čvorovi grane sadrže u sebi samo izmjene koje su u nekom trenutku commitane, tj. ako se preuzme samo jedan čvor, tada se preuzimaju samo izmjene prijevaljene u tom čvoru. To se postiže tzv. Cherry-pick-om.

$git cherry-pick **SHA1\_string\_commita** – poziva se iz tekuće grane u koju želimo importirati određene izmjene(samo iz jednog čvora). String\_commita je u nekoj drugoj grani.

## Tagovi

$git tag – izlist tagova

$git tag ime\_taga – dodavanje novog taga

$git tag –d ime\_taga – brisanje taga

Korisno je znati da naredbacheckout pored toga što služi za prebacivanje sa grane na granu, služi I za prebacivanje u neki čvor označen tagom, ali I referencom HEAD, te SHA1 stringom čvora, što može biti posebno korisno: $git **checkout** 1.0

$git checkout **HEAD –** vraćanje na zadnjiu verziju

## Ispod haube

SHA1 – hash funkcija, kojoj je vrlo teško odrediti ulaz na osnovu rezultata

### Grane

Grane su ustvari reference na njihove zadnje commitove

### Reference

#### .git/hooks

U ovom se direktoriju nalaze skripte koje se, kad se aktiviraju, izvršavaju na određeni događaj u repozotoriju, a aktiviraju se tako s aim se obriše ekstenzija “.sample”. primjer je ako se žele pokrenuti unit testovi prije svakog commita.

## Povijest

### Diff

$git diff master testna-grana -razlika između dvije grane. Prikazat će konkretne razlike u fajlovima između dva commita

$git diff **HEAD**~2 **HEAD**~1 – prikazuje sve razlike između pred-predzadnjeg I predzadnjeg commita.

### Log

$git log ime\_grane – prikazat će povijest izmjena do zanjeg commit-a u navedenoj grani

$git log naziv\_commita – prikazat će log do tog commit-a bez obzira i ako se on nalazi u grani različitoj od trenutne

$git log –n naziv\_commita – prikazat će povijest n izmjena do navedenog commita(obično zadnjeg). Međutim ako nas zanima povijest izmjena od n commita u prošlost od HEAD-a, tada je naredba jednostavno: $git log –n

### Whatchanged

$git **whatchanged** ime\_commita ime\_comita2 – prikazuje koji su se sve fajlovi mijenjali u nekom commitu.

### Pretraživanje povijesti

$git log –**grep**=search\_string – pretraživanje po komentaru commita

$git log **–S**search\_string – pretraživanje sadržaja fajlova u commtevima. Rezultat će biri SHA1 hash string commita gdje se search\_string prvi put pojavljuje. Nadalje, ako želimo vidjeti kako se je prvi put unio taj string, pozovemo naredbu: git diff 76cf8~1 76cf8

Najbolji način da vidimo što se je desilo u određenom commit je: $gitk 1dcf41c4331, gdje je 1dcf41c4331 SHA1 hash string commita.

Gitk je program koji prikazuje povijest trenutne grane

Sa gitk može se pogledati i samo dio povijesti trenutne grane: $gitk HEAD~n – prikazat će povijest do –n commitova u prošlost u toj grani

$git gitk ime\_grane – prikazuje povijest za samo tu granu

$gitk –all – prikazat će povijest svih grana

$gitk grana1 grana2 – prikazat će povijest samo za navedene dvije grane

Opcije Color words i markup words služe da bi se pokazali djelovi linija koji su mijenjani(umjesto cijelih linija).

### Blame

$git blame ime\_datoteke – prikazat će tko je kada i u kojem commitu mijenjao datoteku, po linijama

### Preuzimanje datoteke iz povijesti

Ako se nekom verzijom iz prošlosti želi pregaziti trenutna verzija nekog fajla, tada se koristi sljedeća naredba: $git checkout SHA1\_ime\_commita – ime\_fajla

SHA1\_ime\_commita može biti i npr. HEAD ~1 ili ime\_grane ~1, ako se npr fajl želi preuzeti iz predzadnjeg commita.

### „Teleportiranje“ u povijest

Kao što se sa checkout opcijom možemo prebacivati sa grane na granu, unutar iste grane se također pomoći checkout opcije prebacivati između commitova. Kako smo to vidjeli da se može pomoću tagova(oznaka određenih čvorova), tako se to može raditi i pomoću imena commita ili HEAD oznaka.

$git checkout HEAD ~10

### Reset

$git **reset** –**hard** HEAD ~1 – briše trenutni commit i vraća granu u stanje prethodnog commita, tj. Resetira granu na određeni commit naveden u naredbi.

### Revert

$git **revert** HEAD – vraća HEAD na stanje predzadnjeg commita, ali ne briše zadnji commit(s kojim nismo zadovoljni, pa vraćamo stanje projekta u predzadnji commit), već samo HEAD ~1 postaje novi HEAD – dakle zadnji commit ostaje u grani, ali nakon njega dolazi predzadnji commit, tako da izmjene ostaju u grani, ali su revertane, odnosno pregažene stanjem kakvo je bilo u predzadnjem commitu.

Isti efekt se može dobiti i ako umjesto HEAD notacije koristimo SHA1 oznake commitova i ako pretpostavimo da je commit 402b8ae39 bio negdje u sredini grafa onda ćemo nardebom: $git revert 402b8ae39 dobiti novi čvor u kojem se neće pojavljivati izmjene koje su uvedene u čvoru: git revert 402b8ae39, tj. Stanje projekta biti će kao da se izmjene iz tog commita nisu nikada desile.

### Izrazi sa referencama

Npr. HEAD^1 predstavlja čvor koji neposredno prethodi HEAD čvoru, tj. Ako ih ima više onda su oni: HEAD^1, HEAD^2... npr. Kada je neki čvor merge više grana.

### Reflog

Naredba: $git reflog – pokazat će SHA1 identifikatore svih commiteva u projektu(u svim granama), čak i ako su neke grane u međuvremenu izbrisane, pa to eventualno može pomoći da se neka izbrisana grana ponovo rekreira.

## Udaljeni repozitoriji

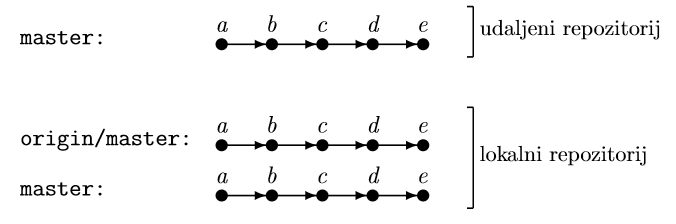
Svaki udaljeni repozitorij mora imati neki alijas. Npr. Origin ili vanjin-repozitorij, marta...

### Kloniranje repozitorija

Postupak kad prebacujemo kopiju udaljenog repozitorija na lokalno računalo

Postupak kopiranja udaljenog repozitorija se svodi na to da lokalno kopiramo udaljeni .git folder u da nakon toga u novoj lokaciji pokrenemo naredbu: $git checkout HEAD. Međutim, u slučaju kopiranja lokalni repozitorij nema svijest o odaljenom repozitoriju čija je ustvari kopija. Da bi bio „svjestan“, provodi se postupak kloniranja: $git clone adresa\_udaljenog\_repozitorija, tj. Klonirani repo. Čuva informaciju o repozitoriju od kojeg je nastao. Udaljeni repo će u tom slučaju za novokreiranu kopiju imati ime: „origin“.

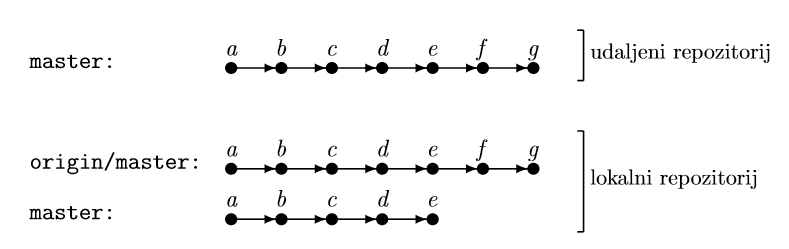
Nakon kloniranja udaljenog repozitorija imamo tri repozitorija kao na slici:



U master repo ćemo commitati svoje izmjene dok ćemo origin/master povremeno osvježavati kako bi imali ažurno stanje svog projekta...

Kloniranje preuzima cijelu povijest projekta, ako nas to ne zanima, moguće je preuzeti samo n zadnjih commitova: $git **clone** –**depth** 5 –**no-hardlinks** git://github.com/tkrajina/uvod-u-git.git

Ažuriranje grane origin/master: $git fetch – nakon čega je poravnato stanje u origin i origin/master grani, tj. U udaljenom repozitoriju i origin/master grani kao na donjoj slici:



$git diff origin/master – razlike između lokalne master i origin/master grane. Ako je prethodno napravljen fetch sa origin repozitorija, tada je to istovremeno i razlika između mastera i udaljenog mastera. Izmjene koje su napravljene na originu se nakon toga mogu vidjeti sa: $git log origin/master  
$git diff origin/master master

Da bi se izmjene povučene sa origina dodale u lokalni master repo, potrebno se je prebaciti u origin/master granu i pozvati merge naredbu:  
$git checkout origin/master  
$git merge origin/master – ovako prebacimo nove izmjene iz origin repozitorija koje smo fetch naredbom prebacili lokalno u njegovu kopiju: origin/master.

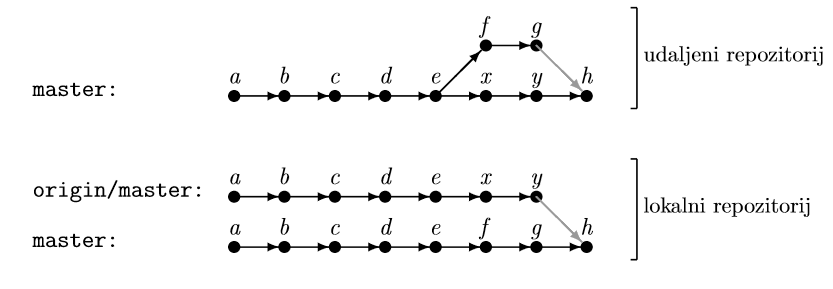
### Pull

Umjesto $git fetch - (master grane) i $git merge origin/master, može se koristiti jednistvena naredba:  
$git pull, koja radi isto samo odjednom.

### Push

Udaljeni repo mora biti konfiguriran da bismo mogli raditi push. Ako imamo ovlasit, onda je naredba: $git push origin master

Prije nego se napravi push, treba najprije „poravnati“ stanje na origin/master sa stanjem na origin, tako da prvo napravimo fetch, ako ima konflikata, onda ih riješimo i nakon toga se merga master i origin/master. Nakon što je to sve dobro prošlo, može se napraviti push: $git push origin master.



### Push tagova

Naredba $git push origin master – šalje na udaljeni repo samo izmjene u grani master. Ako postoje tagovi, njih treba pushati posebnom naredbom: $git push origin –tags

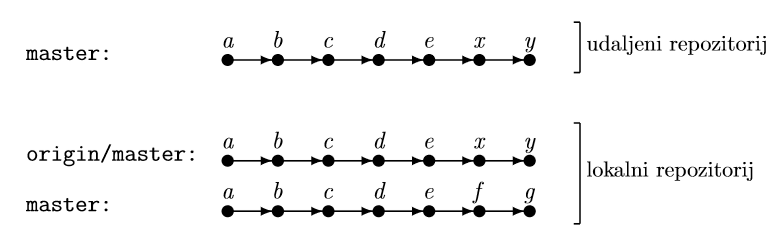
Tag se na remote repozitoriju briše naredbom: $git push origin :refs/tags/moj-tag – međutim treba poziti da netko drugi već nije preuzeo tag jer u tom slučaju nastaju problemi.

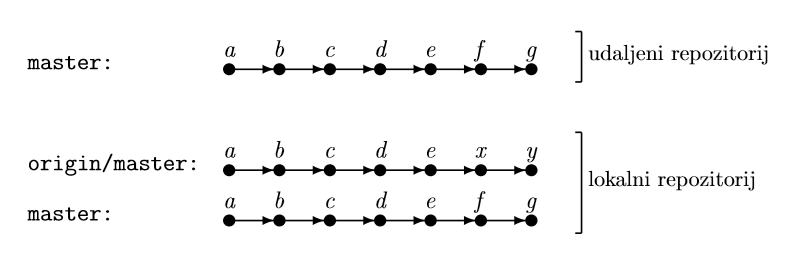
### Rebase origin/master

Koristi se u slučaju kada se želi da se na udaljenom repozitoriju vide svi čvorovi izmjenjene grane koje smo pushali:   
$git checkout master  
$git rebase origin/master  
$git push origin master

### Prisilni push

Slučaj kada želimo svojim izmjenama pregaziti izmjene na origin-u kao na slici.

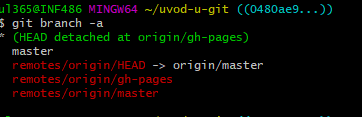


Dakle, x i y čvorove na originu želimo pregaziti svojim izmjenama kao da se nisu desile.   
$git push –f origin master. Nakon toga će rezultat biti:  


...pa zbog toga trebamo napraviti još jedan dodatni fetch kako bi i u origin/master repozitoriju bilo isto stanje kao i na origin repozitoriju: $git fetch

### Rad sa granama

Npr. Kada udaljeni repo ima više grana tada se kloniranjem tog repozitorija, kod nas lokalno kloniraju i sve njegove grane koje dobiju prefiks remote/. Sve grane koje nastanu kloniranjem mogu se izlistati sljedećom naredbom: $git branch –a. Rezultat je kao na donjoj slici:



Ako želimo raditi na nekoj od tih grana, moramo prvo prijeći na tu granu npr.: $git checkout origin/gh-pages, te s te grane napraviti kod sebe lokalno novi branch: $git branch gh-pages, te napokon: $git checkout gh-pages.

Zadnje dvije naredbe mogu se zamjeniti sljedećom naredbom: $git checkout –b gh-pages.

Naredba: $git fetch osvježava sve kopije remote grana sa origina, a ne samo origin/master, pa nakon toga sa: $git checkout gh-pages i $git merge origin/gh-pages, povučemo u lokalnu gh-pages granu nove izmjene ako ih ima. Na kraju ako nakon ovoga želimo poslati scoje izmjene u remote gh-pages, napravimo: $git push **origin** gh-pages, ako je gh-pages ime lokalne grane.

### Dodavanje i brisanje udaljenih repozitorija

Uobičajni pristup bi bio da prvo kloniramo projekt sa jednog udaljenog repozitorija kao npr

$git clone <https://github.com/korisnik/projekt.git>, te dodamo drugi udaljeni repo naredbom:   
$git **remote** **add** bojanov-repo [git@domena.com:projekt](mailto:git@domena.com:projekt)

Na taj način imamo dva udaljena repozitorija: origin i bojanov-repo

Origin repo se može i preimenovati sa: $git remote rename origin karlin-repo

Popis svih udaljenih repozitorija sa kojima radimo dobijemo sljedećom naredbom: $git remote show

Budući da radimo sa više udaljenih repozitorija, moramo sa svakog posebno povlačiti nove izmjene:  
$git fetch bojanov-repo – ova naredba će osvježiti lokalnu granu: remote/bojanov-repo  
$git fetch karlin-repo – ova naredba eć osvježiti lokalnu granu remote/karlin-repo

Ako želimo provjeriti te izmjene, odemo u tu granu i učitamo projekt. Dakle, najprije: $git checkout karlin-repo/master i kad se uvjerimo da je sve OK, vratimo se u master granu sa $git checkout master i naredbom: $git merge karlin-repo/master, preuzmemo karline izmjene.

### Fetch, merge, pull i push s udaljenim repozitorijima(gdje imamo i više grana u svakom repozitoriju)

Kada imamo samo jedan udaljeni repo tada se fetch vrši jednostavno sa: $git fetch. Međutim uz više udaljenih repozitorija mora se navesti ime repozitorija s kojeg se vrši fetch: $git fetch ivanov-repo

Nakon toga ide merge master grane sa: $git merge ivanov-repo/master ili $git merge ivanov-repo/grana... sve to kraće: $git pull ivanov-repo master ili $git pull ivanov-repo grana

Push... $git push ivanov-repo master ili $git push ivanov-repo grana u svakom slučaju kad imamo više udaljenih repozitorija moraju se navest i repo i grana za pull i push

### Pull request

GitHub ima opciju pull requesta

### Bare repozitorij

To je repo koji nema radnu verziju projekta, tj nisu nikad checkout-ani i sadrže samo .git direktorij.

Svaki programer radi na svom lokalnom repozitoriju gdje samo on ima pristup, ali napravi jedan remote bare repo gdje pusha svoje izmjene i gdje i ostali imaju pristup.

Konverzija repozitorija u bare repo se napravi sljedećom naredbom: $git config –bool core.bare true

Drugi način da kreiramo udaljeni bare repo je pomoću **remote** naredbe: $git remote add origin login@server:staza/do/repozitorija, međutim prije nego se može pushati na tako stvoren origin repo, mora se taj repo napraviti da buded Bare kako je prethodno opisano. Nakon toga biti će moguće pushati, pulla ti ostalo u taj repo.

## Higijen repozitorija

### Grane

$git branch –merged master – daje popis svih grana čije su izmjene u potpunosti mergane u master  
$git branch –no-merged naziv\_grane - daje popis svih grana gdje postoje izmjene koje još nisu u potpunosti mergane u granu naziv\_grane.

### Garbag collection

Primjenjuje se kad se žele npr izbrisati čvorovi koji su ostali visiti nakon radnji kao što su: amend ili rebase: $git gc

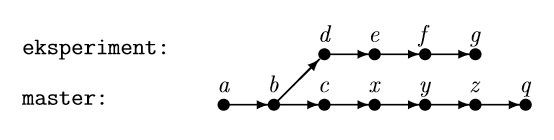
### Povijes i brisanje grana

Čvorovi pokazuju na prethodni čvor. Zadnji čvor(najnoviji) je čvor na koji pokazuje sama grana. Kad obrišemo granu koja nije mergana, tada svi njezini čvorovi ostanu „visiti“ i gube se iz povijesti projekta. Ako je grana mergana, tada se čvorovi ne brišu jer na njih pokazuje neki čvor grane u koju je merge izvršen.

Bezuvjetno brisanje grane: $git branch –**D** ime\_grane  
brisanje grane samo ako je ona potpuno mergana: $git branch –**d** ime\_grane

Merge ber da se u novom čvoru napravi referenca na granu iz koje su izmjene preuzete(kao posljedicu to ima da se nakon brisanja te grane, istovremeno brišu i njezini čvorovi iz povijesti, tj. Više nisu vidljivi): $git merge – squash

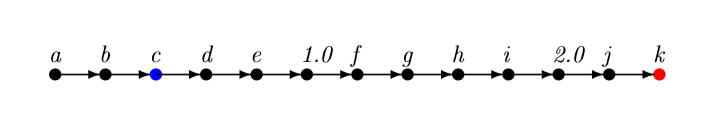
Nakon merge –squash naredbe, stanje je sljedeće:



## Bisect

Ti je git naredba koja koristi kada se traži izmjena u kodu koja je uzrokovala neki bug.

Važno je pritom da postoji način da se utvrdi da li se bug manifestira, te da u povijesti projekta znamo za neki čvor gdje tog buga sigurno nije bilo kao na donjoj slici gdje je c čvor kada smo sigurni da bug-a nije bilo, a k čvor u kojem se je bug manifestirao:



## Česta pitanja

### Jesmo li pushali svoj izmjene na udaljeni repo?

Prvo fetchamo sa udaljenog repozitorija. Nakon toga imamo lokalno stanje sa udaljenog repozitorija u nekoj origin/grani. Nakon toga se sa gitk pogleda koje su razlike u te dvije grane. Npr.  
$git fetch  
$gitk master origin/master

### Commitali smo u krivu granu

Npr dva commita želimo prebaciti iz jedne grane u drugu.

Riješenje:

Prebacimo se u granu u koju želimo preuzeti izmjene: $git checkout ime\_grane\_za\_preuzimanje

Zatim se jedan po jedan preuzmu ta dva commita u rtenutnu granu:  
$git cherry-pick master ~1  
$git cherry-pick master

Nakon toga moramo te commitove maknuti iz pogrešne grane:  
$git checkout ime\_pogresne\_grane  
$git reset –hard master ~2

### Commitali smo u granu X, ali te commitove želimo prebaciti u novu granu(a masete resetirati na isto stanje kao i u udaljenom repozitoriju)

$git branch nova\_grana  
$git fetch  
$git reset –hard origin/master

### Imamo necomitane izmjene i git nam ne da prebacivanje u drugu granu

Jedno riješenje je $git stash

Drugo riješenje je da se dio izmjena(ono što je već napravljeno) commita, a onda da naknadno dovršimo izmjene i pridodamo ih napravljenom commitu pomoću naredbe –amend npr.  
$git commit –amend – m „komentar o djelomičnom commitu“

### Pushali smo u remote repo izmjene koje nismo htjeli

Najprije lokalno vratimo na stanje prije tih izmjena:

$git revert SHA1\_prije\_izmjena  
$git push origin master – na ovaj način će čvor u kojem se nalaze izmjene ostati u remote repozitoriju, ali će se one pregaziti revertiranim izmjenama i novom čvoru.

Isti problem može se riješiti i na drugi način:  
$git reset –hard SHA1\_commita  
$git push-f origin grana

### Mergali smo, a nismo htjeli

Riješenje je da vratimo stanje prije merga sa:  
$git reset –hard HEAD^1

### Ne znamo gdje smo commitali

Treba koristiti naredbu reflog koja će nam izlistati sve čvorove u svim granama