Uvod

Sistem za kontrolu verzija – engl. Version Control System (VCS)

Upravljanje promenama dokumenata, koda, velikih sajtova i drugih kolekcija informacijama naziva se verzioniranje. A sistemi koji omogućavaju rad sa verzijama VCS sistemi.

Promene se obično identifikuju određenim brojem ili slovom označenim kao broj izmene – engl. Revision number. Na primer, početni skup fajlova je „revision 1“. Kada se urade prve izmene, rezultujući skup promena daje „revision 2“ itd.

Svaka izmena je pridružene odgovarajućim vremenskim pečatom – engl. timestamp, kao i osobom koja izvodi izmene.

Linus Torvards je tvorac Git-a. Razvijen je sa ciljem lakšeg razvoja i vođenja projekta Linux. Projekat verzioniranja je razvijen kao projekat otvorenog koda pa je ubrzo postao veoma aktuelan. Takođe, vremenom se razvijao sa potrebama korisnika.

VCS omogućavaju da se izmene mogu uporediti, sačuvati ili vratiti na prethodne, ali i spajati.

Uloga VCS je višestruka.

1. Čuvanje istorije

Promene na kodu ili na dokumentima čuvaju se od samog početka.

U svakom trenutku tj. uvek je moguće vratiti se na neku prethodnu verziju.

Istorija izmena je vidljiva i moguće je ispitati svaku izmenu, na primer:

- Kada je verzija urađena?
- Ko je uradio izmene?
- Šta je izmenjeno?
- Zašto je menjano?
- U kom kontekstu se izmene događaju, tj šta je bilo ispred i šta se događalo nakon toga.

Sav izbrisan sadržaj ostaje dostupan kroz istoriju.

2. Rad u timu

Verzioniranje pomaže pri:

- Deljenju kolekcije fajlova sa ostalim učesnicima tj. članovima tima.
- Spajanje promena koje su nastale od drugih učesnika.
- Osiguravanje da se ništa ne može slučajno izgubiti ili preklopiti.

3. Grananje

Grananje omogućava da postoji više verzija istog programa istovremeno. To se ostvaruje preko grananja. Na primer:

- Glavna grana

- Grana za održavanje (grana koja omogućava ispravke grešaka u starijim izdanjima)
- Grana za razvoj programa
- Grana za novo izdanje (gde se vrši zamrzavanje koda pre novog izdanja)

4. Rad sa spoljnjim učesnicima u razvoju koda

Alati za verzioniranje pomažu u radu tj. razvoju koda u kome učestvuju spoljnji sadradnici tzv. saradnici trećih strana – engl. *third-party contributors*. Ovi alati omogućavaju:

- uvid u ono što se događa u razvoju tj. u projektu
- pomaže im da urade i integrišu izmene (zakrpe)
- grananje razvoja softvera i njegovo spajanje u glavnu liniju 3

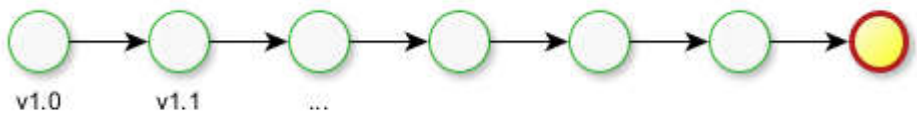
5. Skaliranje

Neki podaci (izvor: Linux Foundation) kazuju da Linux kernel, razvijan primenom GITa:

• ima oko 10000 promena u svakoj novoj verziji, na svaka 2-3 meseca

• 1000+ saradnika

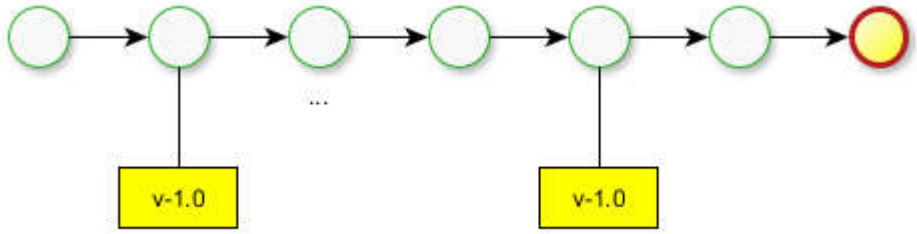
Grafički prikaz nekih pojmova



Slika 1 Repozitorijum koji sadrži istoriju počev od verzije v1.0

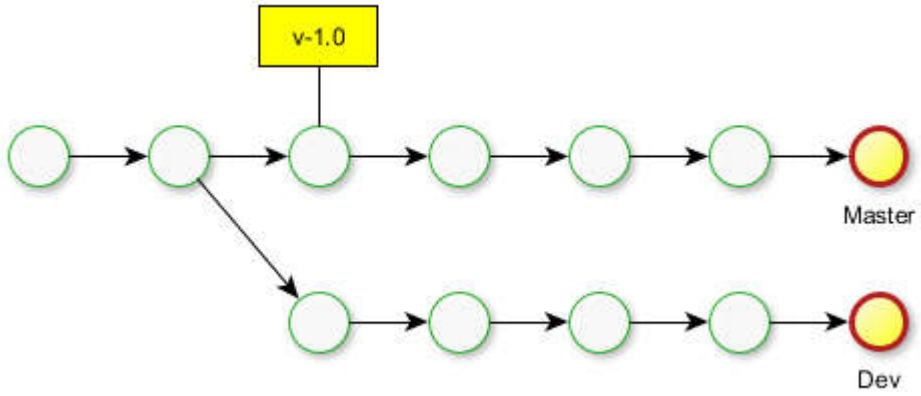
Tekuća verzija se posebno grafički prikazuje i obično se označava kao „HEAD“.

Svaka verzija nosi prateći opis. Međutim, postoje posebne verzije koje su od posebnog značaja i koje se posebno označavaju. Obično su to verzije softvera koje se objavljuju (engl. release version).



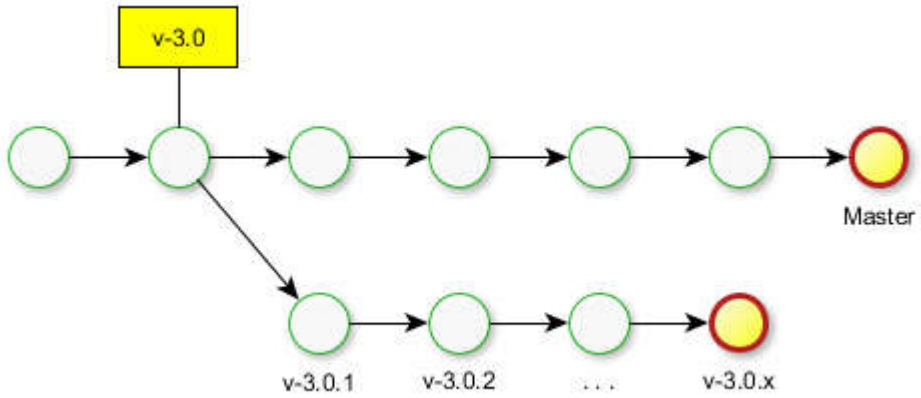
Slika 2. Niz verzija sa posebno označenim verzijama

Grananje predstavlja razdvajanje procesa razvoja projekta u više pravaca. U ovom slučaju, svaki od pravaca dobija sopstvene verzije.



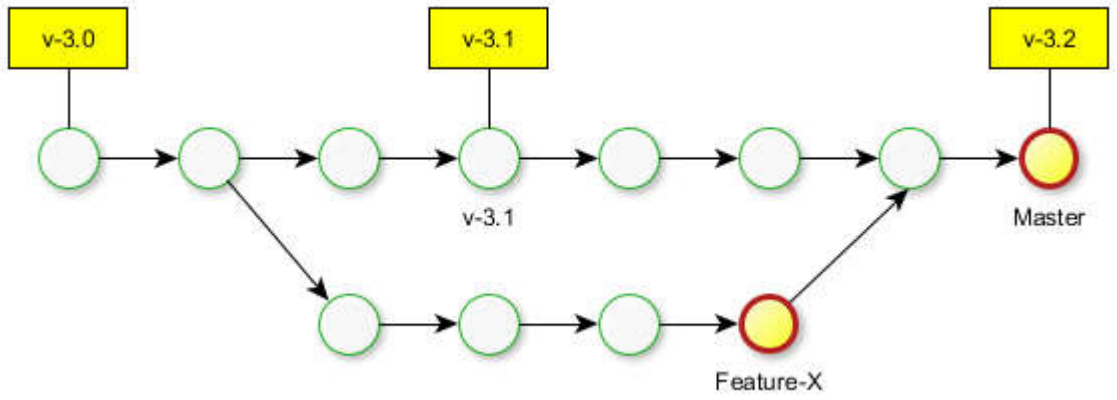
Slika 3 Grananje razvoja u dve grane: *Master* i *Dev*

Jedna od grana koja se u praksi koristi je grana koja se vezuje za objavljenu verziju softvera za koju se vrši ispravka grešaka na toj verziji.



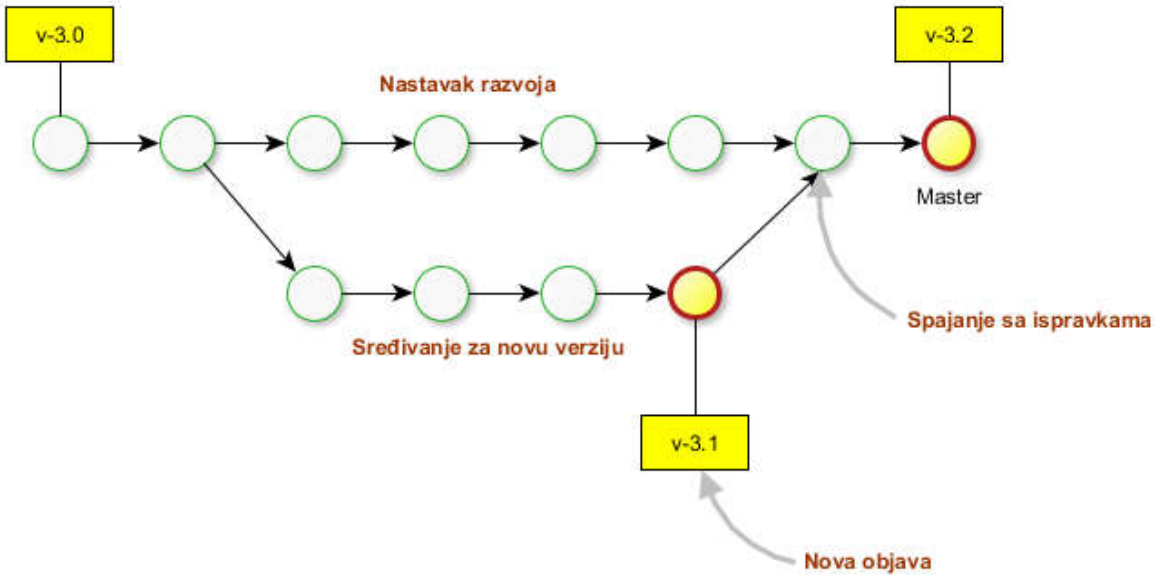
Slika 4 Grana za ispravljanje grešaka za objavljenu verziju v-3.0

Takođe, pri generisanju novih karakteristika za neku postojeću verziju vrši se izdvajanje posebne grane. U toj grani se vrši razvoj novih karakteristika, kao i eventualno ispravljanje grešaka u nekoj podgrani, a na kraju kada se karakteristika uradi, vrši se spajanje (engl. merging) sa glavnom granom.



Slika 5 Grana izdvojena za razvoj nove karakteristike *Feature-X* a zatim spajanje sa glavnom granom.

Kada se ostvari razvoj sa dovoljno karakteristika za novu objavu, nakon toga su prihvatljive jedino izmene koje se tiču ispravki grešaka. Zato se nakon dostizanja željenih karakteristika a pre objave tj pre ispravke grešaka vrši odvajanje u granu za objavu – engl. Release branch. Istovremeno nastavlja se razvoj na glavnoj grani.



Slika 6. Grananje i spajanje grana

Kada se razvoj završi, vrši se spajanje sa glavnom granom na koju se prabacuju sve ispravke tj. promene koje su nastale u grani za objavu. Takođe, grana za objavu opet postaje grana za održavanje tj. ispravku grešaka.

Organizacija verzioniranja

U osnovi postoje dve vrste organizacije rada više korisinika sa repozitorijumom.

Prva podrazumeva da svaki korisnik radi sa istim repozitorijumom. Ova organizacija se naziva **centralizovanom**.

Druga podrazumeva da svaki korisnik radi sa sopstvenim repozitorijumom – **decentralizovana**.

Takođe, razlikuju se dva načina rešavanja konflikta istovremenog rada na istom projektu. Jednostavniji, ali istovremeno manje efikasan metod, jeste zaključavanje pre izmena. Drugi način je spajanje promena. Ovo je efikasniji način ali se mora voditi računa o eventualnim konfiliktima. Git verzioniranje podrazumeva decentralizovan sistem sa spajanjem. Git omogućava pristup bilo kojoj grani tokom razvoja kao i praćenje svih izmena u svakoj fazi razvoja.

Svaki repozitorijum mora imati na raspolaganju odgovarajući prostor za skladištenje kako bi bilo moguće skladištiti izmene u svakom fajlu projekta.

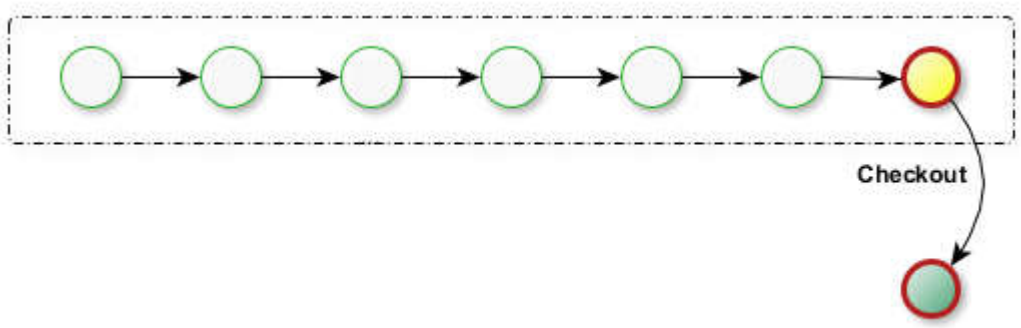
Većina alata za verzioniranje koristi delta kompresiju kako bi optimizovao prostor za skladištenje, osim Git-a koji koristi tzv. objektno pakovanje.

Svaki repozitorijum se identifijuje odgovarajućim URL-om. Alati za verzioniranje koriste više načina za interakciju sa udaljenim repozitorijumima. Obično se koriste standardni protokoli http ili https, ali i zaštićeni poput ssh. U nekim slučajevima moguća je upotreba specifičnih poput svn ili git protokola.

Postupak kreiranja nove verzije

Repozitorijum predstavlja jedinstvenu celinu koja se ne menja direktno. Promena, odnosno kreiranje nove verzije, vrši se u nekoliko koraka.

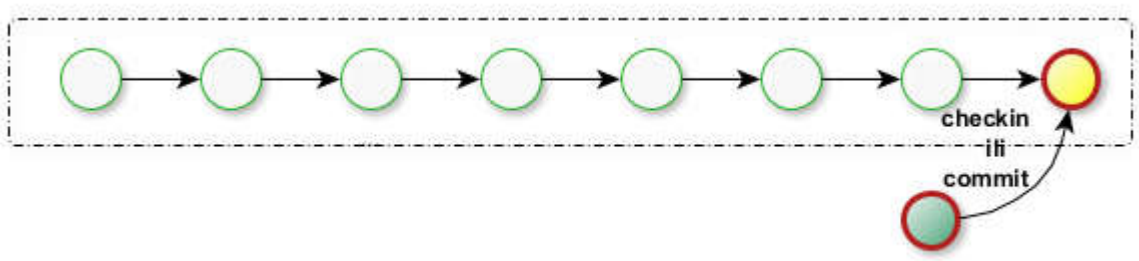
Korak 1. Preuzimanje (lokalne) kopije fajlova koji se koriste odnosno menjaju. Ova se postiže komandom **checkout**.



Slika 7a. Grafički prikaz preuzimanja poslednje verzije

Korak 2. Sledi rad i izmene na preuzetim kopijama fajlova.

Korak 3. Kada je radna kopija fajlova spremna za postavljanje tj. za evidentiranje nove verzije izvodi se komanda **commit**.



Slika-7c. Urađenje izmene se vraćaju na repozitorijum

Koraci od 1 do 3 se zatim više puta mogu ponoviti i tako formirati više verzija.

Šta se čuva na repozitorijumu

Na repozitorijumu se čuvaju svi fajlovi koji se ne generišu od strane alata koji se koriste u razvoju. Takvi fajlovi su:

- fajlovi izvornog koda (.c .cpp .java . . .)
- skripte za izradu projekta (project.sln makefile configure.in . . .)
- fajlovi koji su dokumenta koja prate projekat (.doc .txt . . .)
- fajlovi koji predstavljaju prateće resurse (ikone, slike, audio, . . .)

Fajlovi koje ne treba čuvati su fajlovi koji se generišu. Takvi su:

- .obj .exe .o .dll .class .jar
- fajlovi konda ili skripti ali koje generišu alati.

Čuvanje ovakvih fajlova izazvaće pojavu nepotrebnih konflikata pri spajanju različitih verzija.

GIT

Uvod

Pre 2005 Linux izvorni kod je bio upravljan preko Bitkeeper-a.

U aprilu 2005 usledio je opoziv od free-use licence.

Nijedan alat nije bio dovoljno napredan da bi zadovoljio Linux-ov razvoj sa ograničenjima (distribuiran rad, integritet, performanse). Linus Torvald je počeo da razvija Git.

Jun 2005: prvo izdanje Linux-a kojim je upravljao Git

Decembar 2005: Git 1.0 je objavljen

Napomena: Pre upotrebe potrebno je Git instalirati. Zvanična lokacija za preuzimanje Gita, za Windows operativni sistem, je na Gitovom sajtu: http://git-scm.com/download/win .

Rad u lokalu

Pomoć

Pre nego što praktično počnemo sa radom primenjujući i objašnjavajući razne komande koje možete koristiti, treba da znate prvu:

```
git help [komanda]
```

Ova komanda daje pomoć u vidu objašnjena i sintakse. Ukoliko niste sigurni u sintasku ili opcije, svakako je možete iskoristiti.

Na primer: `git help status`, `git help commit`

Konfigurisanje

Git poseduje parametre za podešavanje tj. konfigurisanje. Osnovna komanda je:

```
git config [parametri]
```

Postoje 3 različita nivoa kofigurancionih parametara. To su:

- Sistemski,
- Globalni ,
- Lokalni.

Vrednosti parametara jednog nivoa preklapa vrednosti iz prethodnog nivoa.

Sistemski/globalni/lokalni nivo

- Sistemski Važi za svakog korisnika na sistemu i za sve repozitorijume.
- Globalni se koristi za sve repozitorijume, ali jednog korisnika.
- Lokalni nivo se koristi za jedan repozitorijum.

Da bi se pogledali konfiguracioni parametri na određenom nivou korisi se komanda:

```
git config --list [--system][--global][--local]  // za pogled na listu parametara
git config --edit [--system][--global][--local]  // za editovanje config fajla
```

Napomene: Ako se ne navede nivo onda se prikazuju parametri za sva tri novoa istovremeno. Za editovanje fajla neophodno je da postoji već podešen editor.

Za postavljanje specifičnog parametra određenog nivoa:

```
git config --system color.ui true
git config --global user.name pera
git config --local core.ignorecase true
```

.gitignore

U realnim primenama u kojima se Git koristi obično se radi sa velikim brojem fajlova u okviru više projekata. Pri tome veliki broj fajlova nastaje primenom kompajlera odnosno korišćenjem određenog IDE okruženja (.class, .pyc, .o). Takvi fajlovi se često ne koriste pri formiranju nove Git verzije jer se kreiraju automatizovano.

Da se odvajanje fajlova koji se postavljaju na repozitorijum ne vrši ručno, postoji jedan fajl .gitignore čiji sadržaj predstavljaju ekstenzije fajlova koje Git treba a ignoriše pri pravljenju verzija.

Na primer, za jedan projekata u Visual Studio IDE možete kreirati fajl .gitignore sledećeg sadržaja:

```
#Visual Studio files
*.[Oo]bj
*.user
*.aps
*.pch
*.vspscc
*.vssscc
*_i.c
*_p.c
*.ncb
*.suo
*.tlb
*.tlh
*.bak
*.[Cc]ache
*.ilk
*.log
*.lib
*.sbr
*.sdf
*.pyc
*.xml
ipch/
obj/
[Bb]in
[Dd]ebug*/
[Rr]elease*/
Ankh.NoLoad
```

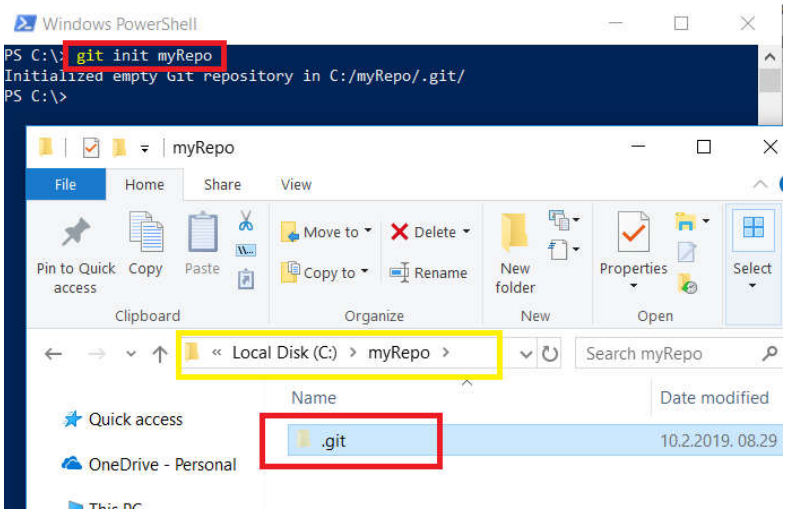
Kreiranje repozitorijuma

Komanda za kreiranje lokalnog repozitorijuma je:

```
git init repo
```

Ovom komandom se kreira direktorijum koji će biti repozitorijum. Ako se direktorijum ne navede repozitorijum se kreira u tekućem direktorijumu.

Kreiranje repozitorijuma prouzrokuje kreiranje skrivenog foldera `.git` u repozitorijumu. Ovaj folder sadrži sve podatke koji se tiču promena na repozitorijumu. Brisanje ovog foldera znači brisanje celokupne istorije, odnosno brisanje svih podataka o repozitorijumu. Pogledajte primer:



Slika 8. Kreiranje prvog repozitorijuma

Status repozitorijuma

Status repozitorijuma predstavlja stanje u repozitorijumu koje obuhvata podatke o radnim fajlovima kao i podatke o fajlovima koji su već indeksirani. Komanda je:

```
git status
```

Na primer:

1. Ako nema fajlova za snimanje:
```
$ git status
On branch master
nothing to commit, working tree clean
```
2. Ako postoje brisanje/dodavanje/izmena fajlova:
```
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   dokument3.txt
        deleted:    dokument4.txt

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

        New Text Document.txt
        dokument5.txt

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

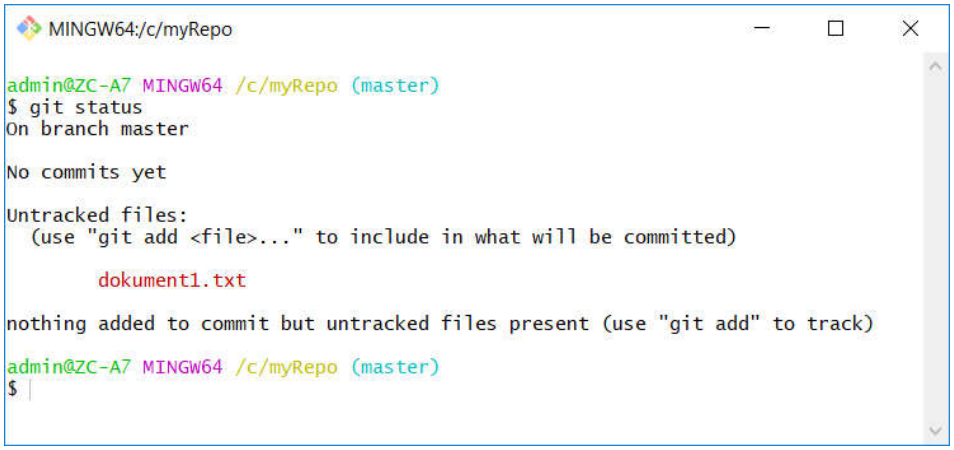
Sada pogledajmo kako se vrši dodavanje fajla u repozitorijum.

Dodavanje fajla

Ako je neki fajl u folderu koji je repozitorijum to ne znači automatski i praćenje tog fajla. Da bi Git čuvao verzije ovog fajla tj. pratio promene na tom fajlu najpre taj fajl treba uključiti u praćenje tj. kaže se da je potrebno postaviti ga na **scenu** (eng. stage) koju obuhvata Git. Za fajlove koji su na sceni kaže se da su **indeksirani** (eng. index). Ovo se postiže komandom:

```
git add imeFajla
```

Pogledajmo prikaz git statusa pre i posle dodavanja fajla dokument1.txt



Slika 9. Status pre i posle dodavanja

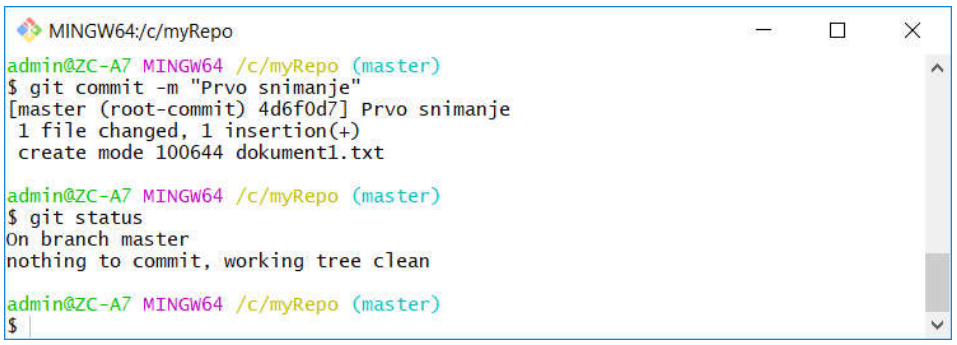
Skup fajlova koji Git prati naziva se scena ili indeks tj. u dokumentaciji engl. **index** ili **staging area**. Obratite pažnju da se poruke odnose na granu **master**. Ovo je osnovna tj. glavna grana repozitorijuma.

Snimanje promena

Snimanje promena u repozitorijum obavlja se komandom commit na sledeći način:

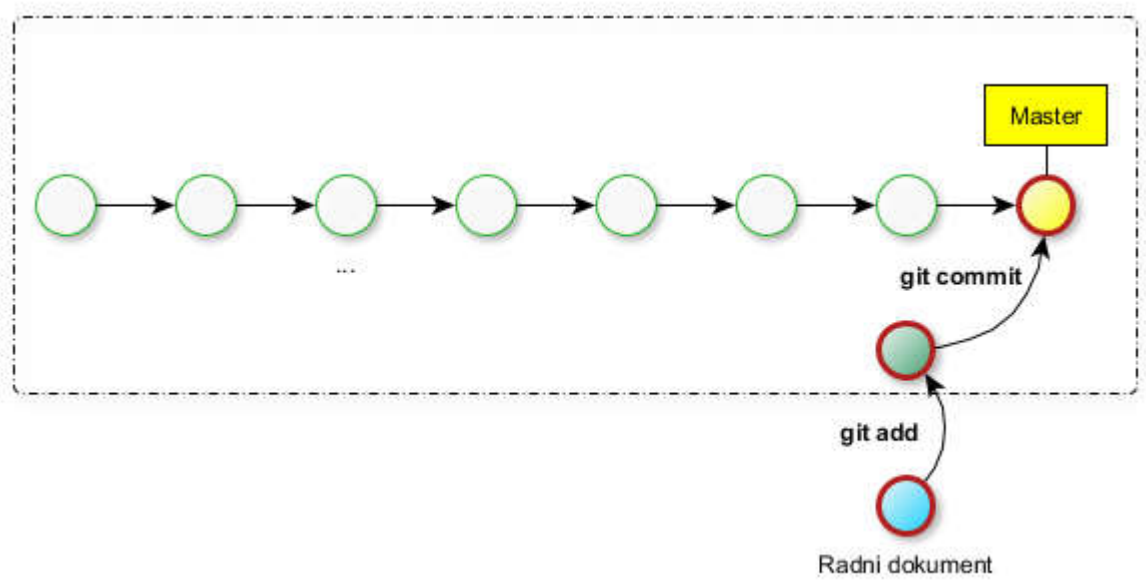
```
git commit -m "poruka"
```

Uz snimanje obavezno se navodi poruka koja se kasnije koristi u prikazu istorije. Novi status bio bi:



Slika 10. Snimanje promena u repozitorijumju

Postupak dodavanja fajla u repozitorijum kao i snimanje promene na repozitorijumu može se prikazati na slici 11.



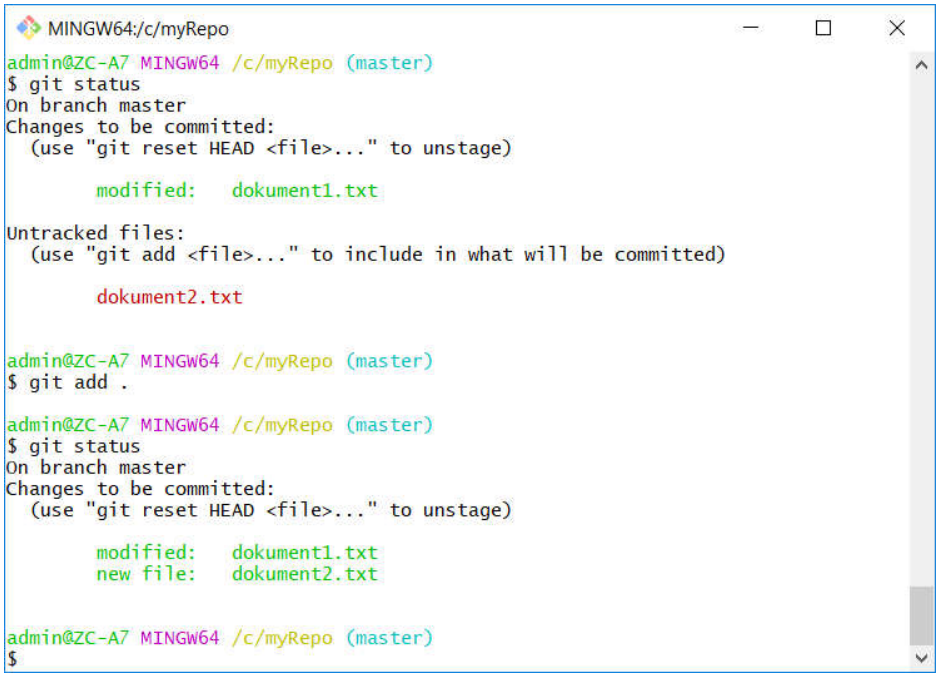
Slika-11. Grafički prikaz dodavanja i snimanja novog fajla

Naredba commit snima sve promene na fajlovima koji su indeksirani. Ako se u naredbi commit navede eksplicitno naziv fajla onda se taj fajl istovremeno indeksira (kažu nekada stejdžuje od engl. stage) i snima (kažu nekada komituje od engl. commit) u repozitorijum.

Moguće je izvesti delimično snimanje izmena. Ovo se izvodi eksplicitnim navođenjem fajlova koji treba snimiti. Češći slučaj je indeksiranje veće grupe fajlova. Ovo se postiže primenom specijalnih karaktera * ili . umesto eksplicitnog naziva fajla. Na primer:

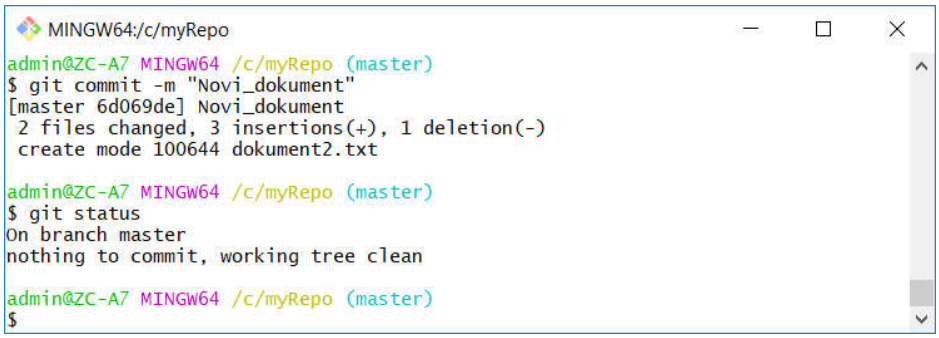
```
git add .
```

Ova komanda dodaje sav sadržaj tekućeg foldera u repozitorijum. Pod sadržajem se podrazumevaju fajlovi u ovom folderu kao i fajlovi u svim podfolderima.



Slika 12. Dodavanje svih dokumenata jednog foldera

Zatim se može izvesti novo snimanje.



Slika 13. Snimanje svih izmena

Tajno skladištenje

Ako se moramo prebaciti u drugu granu a pri tome izmene nismo završiti Git to neće dopustiti. Zato se promene moraju snimiti ili uraditi komanda git stach ili git stash push pomoću koje se mogu privremeno uraditi izmene. Kada se kasnije vratite na prvobitnu granu, prethodne izmene mogu se vratiti unazad. Na primer:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   dokumen7.txt
        modified:   dokument1.txt


admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git stash
Saved working directory and index state WIP on master: 3d75a7a v-1.6
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Pogled ajmo i listu

```
 $ git stash list
stash@{0}: WIP on master: 3d75a7a v-1.6
```

Nakon toga , varaćanje.... `git stash apply [stash@{x}]`

```
$  git stash apply
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   dokumen7.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   dokument1.txt
```

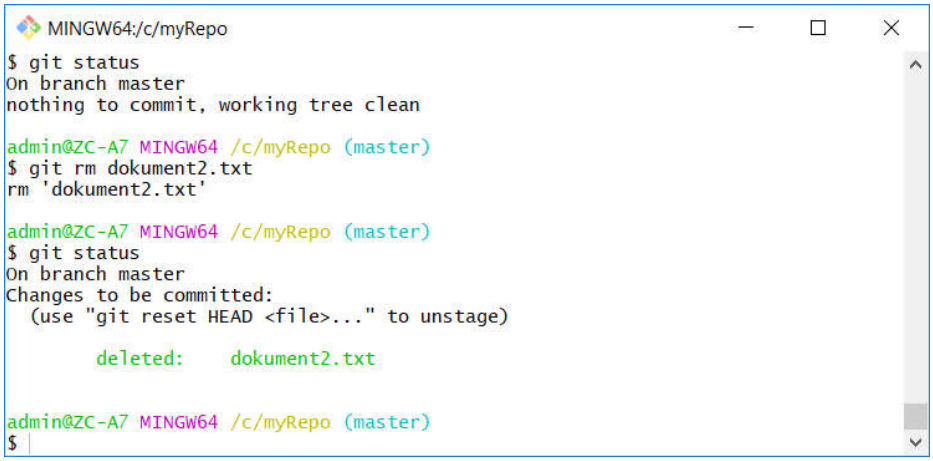
git stash → save/restore the state of the working copy and index (useful when in need to commit an urgent fix)

Brisanje fajlova

Fajlovi mogu da se obrišu iz repozitorijuma. Ovaj postupak znači uklanjanje fajlova sa scene tj. indeksa **ali istovremeno i njihovo brisanje u folderu**. Komanda kojom se briše neki fajl je:

```
git rm [--cached] file
```

Pogledajmo primer sa pratećim statusima.



Slika 14. Brisanje fajla dokument2.txt iz repozitorijuma

Ova komanda ima više opcija. Navedimo samo opciju **–cached.** Ovom opcijom se fajl izbacuje sa scene tj. ostavlja trag kao da je obrisana i na dalje neće biti deo sledećih snimanja, ali pri tome fajl se neće zaista obrisati.

Čišćenje radnog prostora

Git poseduje posebnu komandu za brisanje fajlova koje nisu deo scene. Brisanje ovih fajlova veoma je korisno ako koristimo neki IDE pa ne želimo da čuvamo fajlove i foldere koje generiše IDE ili koje su privremene. Ova komanda je neka vrsta „čišćenja“ foldera repozitorijuma.

```
git clean [-n][-i][-q][-x][-X][-d][-f]<path>
```

Brisanje se obalja rekurzivno na svim fajlovima koji nisu uključeni u reviziju počev od tekućeg foldera. Pošto Git može da ograniči rad odvajajući vrste fajlova sa kojima ne radi, moguće je preko opcije –x isključiti takvo ponašanje tj obezbediti brisanje na svim fajlovima. Ovo je od koristi ako se inače isključi praćenje fajlova koje generiše sistem, a pri brisanju se želi takve fajlove izbrisati.

Ako se navede **<*path*>** putanja onda se akcija primenjuje samo na te fajlove.
-n (--dry-run)
Samo pokazuje koji fajlovi/folderi će biti uklonjeni bez stvarnog uklanjanja. Zbog osetljivosti ove komande, obično je važno proveriti šta će zaista biti obrisano.
-i (--interactive)
Interaktivno pokazuje šta će birit obrisano.
-q (--quiet)
Tiho izvršavanje. Prikazaće samo greške ako postoje ali i ne fajlove koje obriše.
-x (malo slovo x)
Ignoriše pravila navedena u fajlu .gitignore (po folderu) odnosno $GIT_DIR/info/exclude. Omogućava brisanje svih fajlova koji nisu indeksirani.
-X (veliko slovo X)
Uklanja samo fajlove koje Git ignoriše. Ovo može biti korisno za *rebuild*.
-d
Uklanja foldere koji se prate kao i fajlove. Ukoliko je neki folder deo nekog drugog repozitorijuma onda se ipak ne briše.
-f (--force)
Ukoliko je Git konfigurisan tako da je promenljiva clean.requireForce postavljena na true, onda se brianje neće izvršiti bez da eksplicitno navedete ovu opciju.

Za prethodni primer:

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        dokument2_BACKUP_13472.txt
        dokument2_BASE_13472.txt
        dokument2_LOCAL_13472.txt
        dokument2_REMOTE_13472.txt
        sh.exe.stackdump

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

admin@DESKTOP-KUT132H MINGW64 /d/myRepo (master)
$ git clean -n
Would remove dokument2_BACKUP_13472.txt
Would remove dokument2_BASE_13472.txt
Would remove dokument2_LOCAL_13472.txt
Would remove dokument2_REMOTE_13472.txt
Would remove sh.exe.stackdump
```

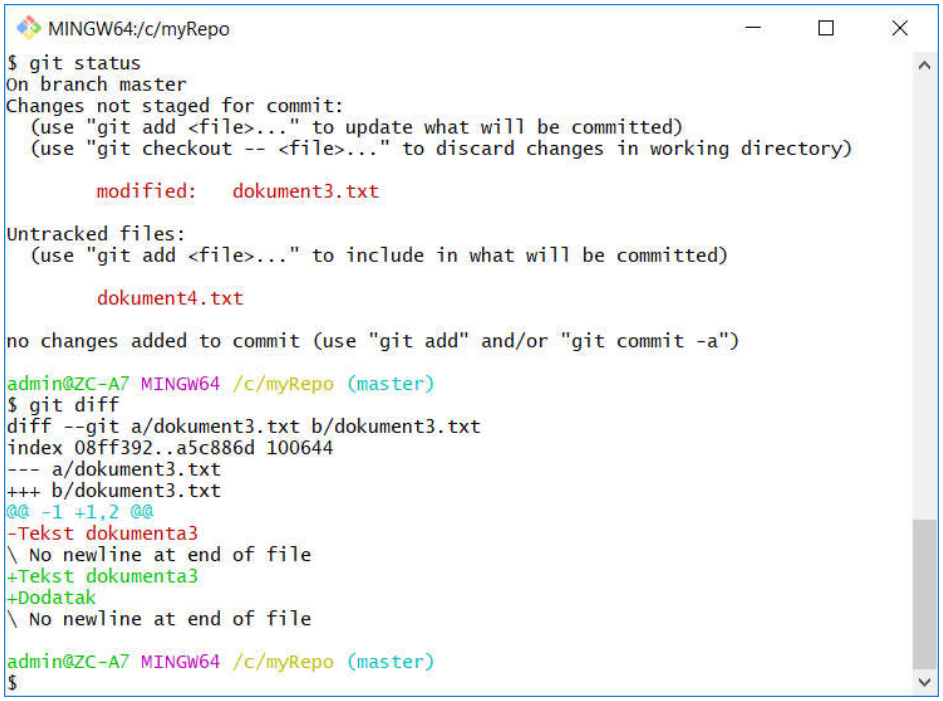
```
admin@DESKTOP-KUT132H MINGW64 /d/myRepo (master)
$ git clean -f
Removing dokument2_BACKUP_13472.txt
Removing dokument2_BASE_13472.txt
Removing dokument2_LOCAL_13472.txt
Removing dokument2_REMOTE_13472.txt
Removing sh.exe.stackdump

admin@DESKTOP-KUT132H MINGW64 /d/myRepo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Razlike

Git poseduje komande za prikaz razlika revizija. Standardna komanda je:

```
git diff
```

Ovom komadom prikazuje se razlike radnih kopija svih indeksiranih fajlova. Razlika se odnosi na promenjene fajlove u odnosu na poslednje snimanje (commit).



Slika 15. Prikaz komande: `git diff`

Obratite pažnju da ova komanda ne obuhvata fajlove koji nisu indeksirani bez obzira što pripadaju istom folderu.

Radni fajlovi se najpre postavljaju na scenu/indeks a zatim se vrši snimanje/komit. Prebacivanjem na scenu, razlike se mogu tražiti sada između fajlova na sceni i poslednjeg snimanja. Opcija `--staged` odnosi se na razilke u odnosu na fajlove koji su na scenu.

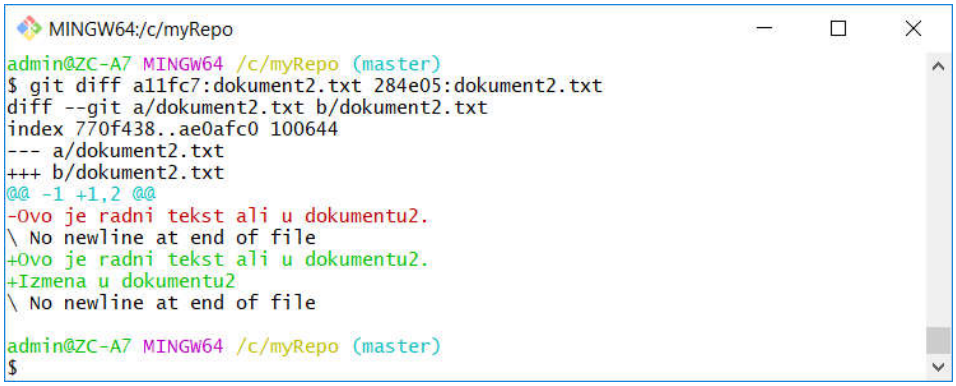
Primenom komande

```
git diff --staged r1
```

prikazuju se razlike tekućih fajlova na sceni i neke revizije r1. Ako se ne navede revizija onda se prikazuje razlika fajlova koji su na sceni od poslednjeg snimanja.

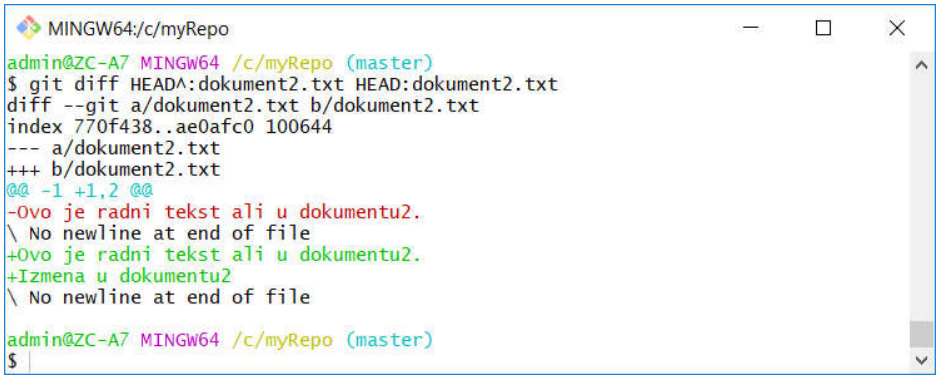
Komanda se može koristiti i za prikaz razlika između dve verzije. U tom slučaju komanda izgleda ovako:

```
git diff r1:file1 r2:file2
```



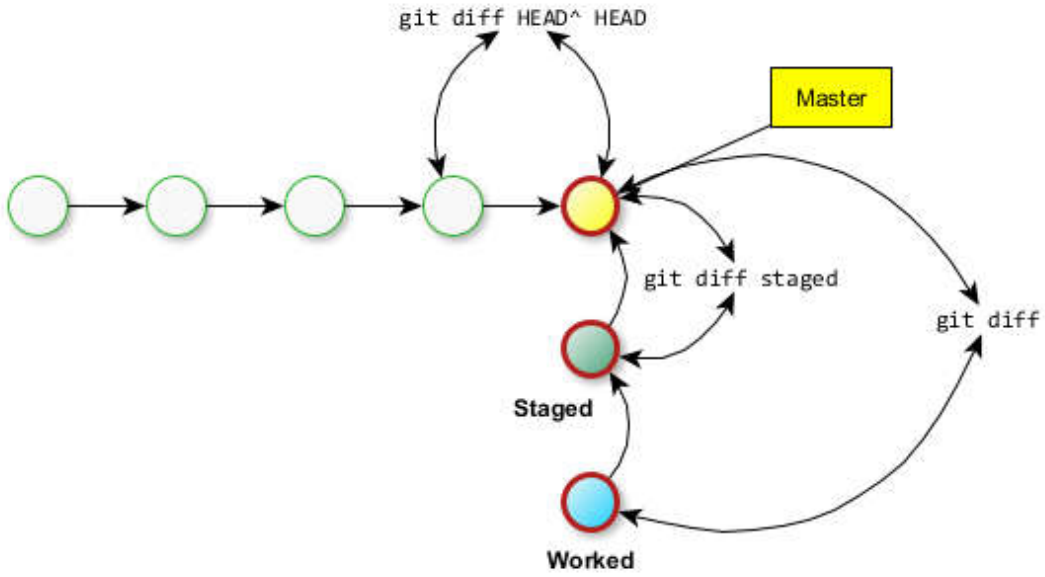
Slika 16. Prikaz razlika: `git diff a11fc7:dokument2.txt 284e05:dokument2.txt`

U primeru su za oznake revizija korišćene vrednosti a11fc7 odnosno 284e05 koje predstavljaju jedinstvene oznake sačuvanih revizija. Kasnije ćemo pogledati kako se mogu dobiti ove vrednosti iz istorije revizija. Sada pogledajmo još jedan način upoređivanja već postojećih revizija. Imajući u vidu da je HEAD univerzalna oznaka za tekuću verziju, onda se ova oznaka zajedno sa prefiksom ^ može koristiti za oznaku revizije. Tako gornji se primer može drugačije napisati kao:



Slika 17. Prikaz razlika: `git diff HEAD^:dokument2.txt HEAD:dokument2.txt`

Grafički prikaz nekih od prethodno objašnjenih komadi dat je na narednoj slici.

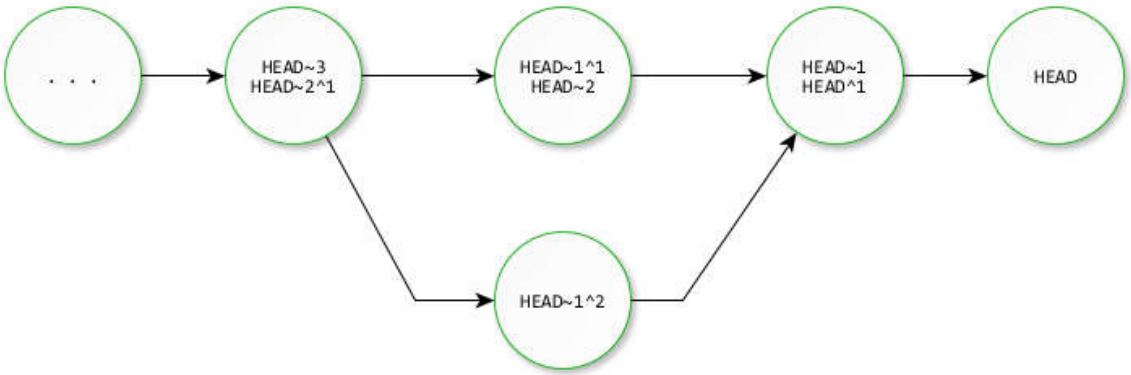


Slika-18. Prikaz više različitih diff komandi

Referenca HEAD

U radu sa Git-om često se barata sa terminimu HEAD.
HEAD: Ukazuje na poslednje tj. važeće snimanje na repozitorijum. U standarnim operacijama, kaže se najčešće, HEAD ukazuje na najnovije snimanje u trenutnoj grani, ali to ne mora biti slučaj. HEAD znači referencu na ono što je trenutni repozitorijum.
Ukoliko referenca HEAD ukazuje na snimanje koje nije vrh grane tj. nije poslednje u grani onda se to naziva "odvojena glava" (eng. detached head).

Postoji niz skraćenih oznaka počev pod poslednje. Njihove oznake su date na slici



Istorija promena

Istoriju snimanja dobijamo kratkom komandama:

`git log [--grap=regExp][-p]`

`git whatchanged [--grap=regExp]`

Druga komanda uz prikaz snimljenih verzija daje i istoriju na tim verzijama. Na primer:

```
commit 284e05d50978f23ee7357976befef27434b64ed9
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 18:45:04 2019 +0100

    v-1.2

    Izm: dokument2
    Nov: dokument3

commit a11fc79d92cc8a98f7cf859db23a3489098768b0
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 10:16:05 2019 +0100

    v-1.1

    Nov: dokument2
    Izm: dokument1

commit 4d6f0d78c030c2caccb5ad2cb0a004d42276767c
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 09:21:44 2019 +0100

    Prvo snimanje
```

```
commit 284e05d50978f23ee7357976befef27434b64ed9
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 18:45:04 2019 +0100

    v-1.2

    Izm: dokument2
    Nov: dokument3

:100644 100644 770f438 ae0afc0 M
dokument2.txt
:000000 100644 0000000 08ff392 A
dokument3.txt

commit a11fc79d92cc8a98f7cf859db23a3489098768b0
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 10:16:05 2019 +0100

    v-1.1

    Nov: dokument2
    Izm: dokument1

:100644 100644 760889e bd049fb M
dokument1.txt
:000000 100644 0000000 770f438 A
dokument2.txt

commit 4d6f0d78c030c2caccb5ad2cb0a004d42276767c
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 09:21:44 2019 +0100

    Prvo snimanje
```

Obe komande mogu imati parametar za pretragu. Vrednost za pretragu je regularni izraz, na primer:

```
$ git whatchanged --grep=$1.6
commit 3d75a7a0851b9a3663a2d79d9b37192e2db9946c (HEAD -> master, tag: ignore)
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sat Feb 16 14:27:22 2019 +0100

    v-1.6

:000000 100644 0000000 36045b4 A        .gitignore
:000000 100644 0000000 a31b7e9 A        dokument5.txt
```

Drugi slučaj u praktičnoj primeni je pretraga snimljenih verzija po nekoj vrednosti stringa u fajlu.

Sintaksa je:

`git log S“string koji se pretražuje“`

Na primer:

```
$ git log -STekst
commit 7f6fde2b899f92e9924fc5543af6062138a35d08 (tag: grananje)
Author: zoran <zoran.cirovic@gmail.com>
Date:   Mon Feb 11 19:20:55 2019 +0100

    v-1.3

commit 284e05d50978f23ee7357976befef27434b64ed9
```

```
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 10 18:45:04 2019 +0100

    v-1.2

    Izm: dokument2
    Nov: dokument3
```

Zapazite da ako string koji se pretražuje ima razmake onda ga moramo staviti između znaka navoda, u suprotnom ne.

Jedna korisnijih opcija je – p --patch, koja pokazuje razliku između svakog snimanja.

```
git log S“string koji se pretražuje“
```

Zbog velikog broja verzija postoji opcija koja pokazuje koliko poslednjih snimanja se koristi.

```
$ git log -p -1
commit 8f2da4dc6d656a2b896feb1eef310b24f4e871d3 (HEAD -> master)
Author: zoran <zoran.cirovic@gmail.com>
Date:   Sun Feb 17 21:35:09 2019 +0100

    v-1.8

diff --git a/dokument2.txt b/dokument2.txt
index 10d5de9..fc06fab 100644
--- a/dokument2.txt
+++ b/dokument2.txt
@@ -1 +1 @@
-glavne izmene...jos neke izmene
\ No newline at end of file
+glavne izmene
\ No newline at end of file

admin@ZC-A7 MINGW64 /c/myRepo (master)
```

Blame

Git omogućava i praćenje promena na određenoj datoteci primenom komande:

```
git blame [fileName]
```

Rezultat je prikaz kada i u kom snimanju je napravljena izmena navedenom fajlu.

```
$ git blame dokument1.txt
a11fc79d (zoran             2019-02-10 10:16:05 +0100 1) Ovo je radni tekst.
00000000 (Not Committed Yet 2019-02-17 20:52:32 +0100 2) Ovo smo izmenili.xx
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git blame dokument3.txt
7f6fde2b (zoran 2019-02-11 19:20:55 +0100 1) Tekst dokumenta3
7f6fde2b (zoran 2019-02-11 19:20:55 +0100 2) Dodatak
```

Tagovanje

Svako snimanje pomoću Gita jedinstveno je označeno jednom SHA vrednošću na koju korisnik ne utiče. Ove vrednosni nisu čitljive i teže ih je koristiti u radu. Umesto njih lakše je pratiti snimanja preko komentara koji se obavezno stavljaju pri svakom komitu.
Međutim, komentar ne može da posluži kao jedinstvena oznaka za rad sa određenim komitom. Osim komentara, korisnik može svakom komitu pridružiti jednistvenu oznaku koja se naziva **tag**. Čak je moguće dodati i više tagova za isti komit.

```
git tag [name][-d name]hard|soft|mixed]
```

Komanda **git tag** bez dodatnih argumenata daje listu tagova za tekući projekat. Ukoliko p
name : dodaje se novi tag na poslednji komit.
-d name : briše se postojeći tag *name*.
Kada postoji definisan tag može se koristiti umesto SHA vrednosti na mestima gde je definisan. Na primer:

`$ git tag`
`dev-oznaka-1`
`grananje`
`ignore`
	Prikaz tagova

`$ git checkout grananje`
`Note: checking out 'grananje'.`
 	
`You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.`
 	
`If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:`
 	
`git checkout -b <new-branch-name>`
 	
`HEAD is now at 7f6fde2 v-1.3`
`A dokumen7.txt`
`M dokument1.txt`
	Prebacivanje na komit za koji je postavljen tag „grananje“. Ova komanda je analogna:
	git checkout 7f6fde2…5d08

`$ git checkout master`
`Previous HEAD position was 7f6fde2 v-1.3`
`Switched to branch 'master'`
`A dokumen7.txt`
`M dokument1.txt`
	Vraćanje reference HEAD na kraj grane master

Uklanjanje sa scene

reset

Sistem za verzioniranje mora imati komande za lako uklanjanje sa scene repozitorijuma. Ovo se izvodi primenom komande:

```
git reset [--hard|soft|mixed]
```

Postoje tri osnovna oblika ove komande. Ovi oblici se definišu arugementima: --soft, --mixed, --hard. Ova tri argumenta odgovaraju redom internim stanjima Git mehanizmima.

Argumenti hard,soft,mixed.

Podrazumevano pozivanje git reset-a ima implicitne argumente --mixed i HEAD. To znači da je izvršavanje resetovanja git-a ekvivalentno izvršavanju git reset -mixed HEAD-a. U ovom obliku HEAD je specificirana naredba. Umesto HEAD-a može se koristiti bilo koji Git SHA-1 haš vrednost.

--hard

Ovo je najdirektnija i istovremeno opasnija opcija ali i često korišćena. Kada se uradi **--hard** komanda podaci se postavljaju na specificirano snimanje (komit). Zatim se sadržaj scene i radnih kopija resetuju tako da se poklapaju sa navedenim komitom. Sve promene koje su eventualno takođe

se resetuju da odgovaraju stanju tokom komita. To znači da će svaki rad koji je bio na čekanju bilo na sceni ili radnom delu biti izgubljen.

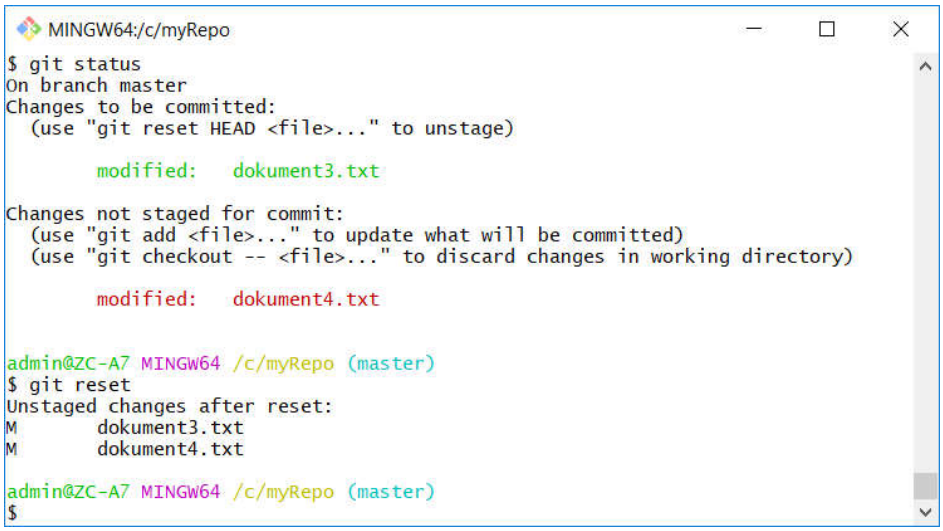
--mixed

Ovo je podrazumevani mod. Referenca se ažurira. Promene na sceni se resetuju na stanje nekog specifičnog snimanja. Promene na sceni se prebacuju u radne promene.

--soft

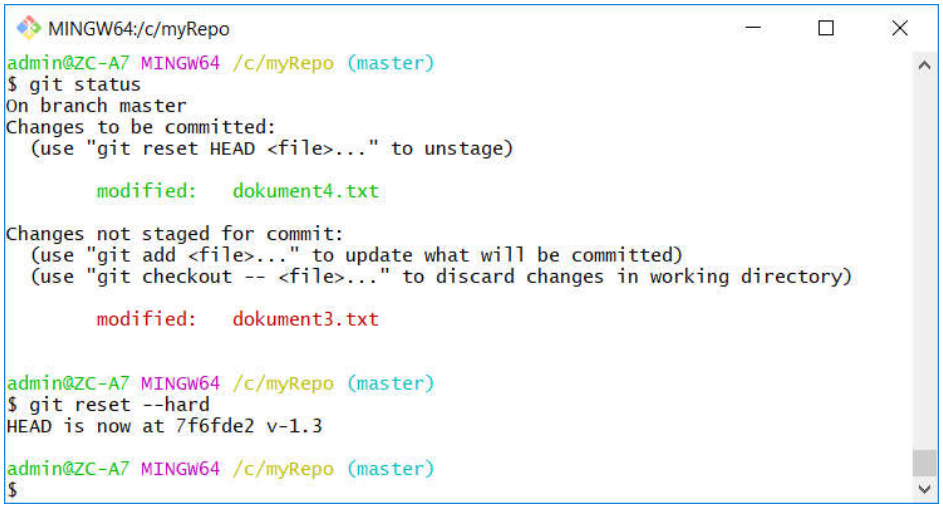
Primena ovog argumenta menjaju se samo reference. Sadržaj na sceni i u radnom folderu se ne menja.

Slede primeri:



Slika 19. Rezultat primene `git reset`

Ako je važno poništiti i izmene u radnim kopijama, koristi se opcija `--hard`. U ovom slučaju sve izmene u odnosu na prethodno snimanje se poništavaju. Dakle, pozicija liči na onu pri poslednjem snimanju.



Slika 20. Rezultat primene `git reset --hard`

Ukoliko želimo da uklonimo baš određeni fajl, određene revizije, onda se ova komanda proširuje oznakom revizije kao i nazivom fajla:

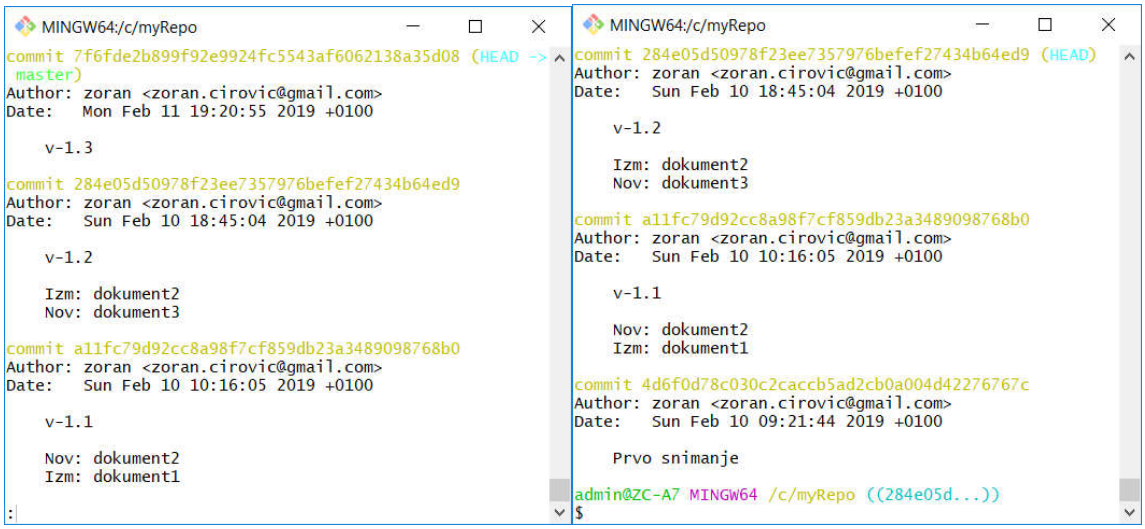
```
git reset HEAD fileA fileB ...
```

checkout

Promene je moguće poništiti pomoću komande **checkout** vraćanjem na određenu reviziju.

```
git checkout rev
```

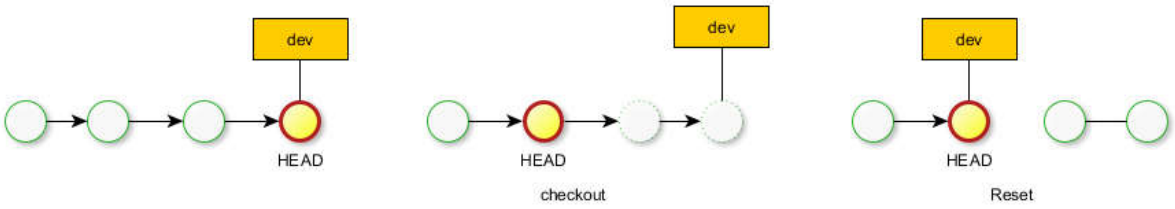
Ovom komandom vrši se poništavanje promena i preuzimanje neke prethodne revizije i njeno postavljanje na tekuću.



Slika 21. Prikaz istorije (git log) pre i posle primene komande git checkout r

Obratite pažnju da je **rev** jedinstvena hex brojčana SHA oznaka za reviziju. Jedinstvene oznake revizija mogu se pogledati komandom `git log`.

Napomena: Komande **git reset** odnosno **git checkout** su slične. Dok komanda git checkout samo vrši promenu na HEAD referenci, komanda **git reset** pomera HEAD referencu i tekuću referencu na granu. Ovo bolje ilustruje sledeći primer:



Slika 22-2. Prikaz razlike primene checkout i reset komandi

Revert

Svako novo snimanje utiče na istoriju verzija dodajući novi čvor sa svim podacima jedne verzije. Reset komanda menja istoriju. Ovo može negativno da utiče na timski rad i nekada nije dobra opcija da bi se ispravila nega greška. Ukoliko greška postoji u poslednjem komitu, a nije nam dozvoljeno da koristimo reset zbog određenih razloga, koristi se revert komanda.

```
git revert ref
```

Ova komanda dodaje novi komit na sam kraj grane koji je identičan referenci koja se navodi. Na primer:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   dokument2.txt


admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git status
On branch master
nothing to commit, working tree clean

admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git revert HEAD
hint: Waiting for your editor to close the file... 'C:/Program Files

admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    dokumen7.txt
        modified:   dokument1.txt
```

Ispravka poslednjeg snimanja

U praksi se događa da se nakon snimanja zaključi da je neophodno nešto izmeniti, ali ne u novoj verziju već na postojećem komitu tj. verziji koja je već snimljena.

Neka je trenutno stanje:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   dokument5.txt
```

Dakle imamo novi dokument na sceni koji hoćemo naknadno da snimimo u poslenji komit i naravno preklopimo poruku novom porukom. Komanda bi bila:

```
git commit --amend -m “v-1.6“
```

Za prethodni primer rezultat ove komande je:

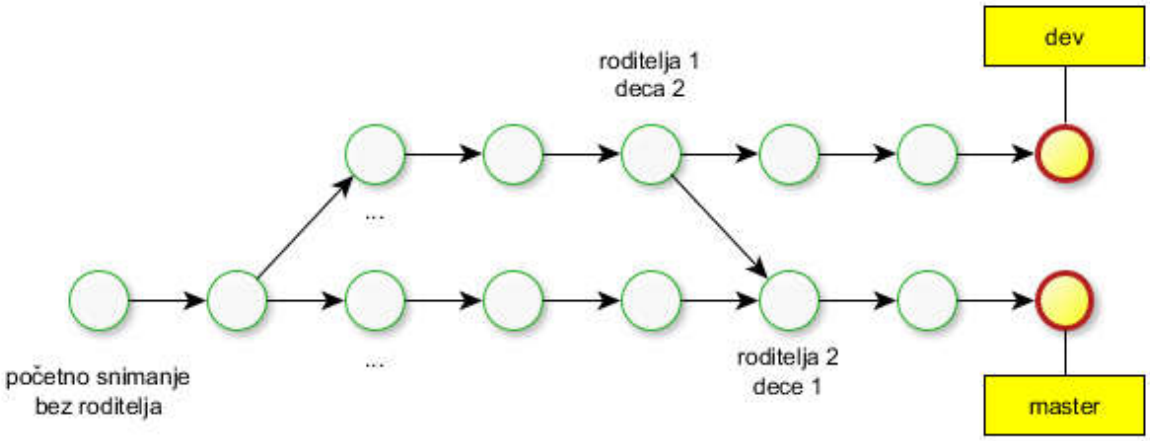
```
$ git commit --amend -m "v-1.6"
[master 3d75a7a] v-1.6
 Date: Sat Feb 16 14:27:22 2019 +0100
 2 files changed, 47 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 dokument5.txt
```

Vežbe

1. Kreirajte novi repozitorijum
2. Kreirajte novi fajl. Dodajte ga u repozitorijum. Snimite.
3. Pokrenite gitk.
4. Promenite fajl i uradite novo snimanje.
5. Promenite naziv fajla git mv
6. Proverite status
7. Obrišite fajl i snimite
8. Kreirajte dva nova fajla i snimite. Zatim modifikujte sadržaj a zatim pogledajte razliku radnih kopija
9. Dodajte jedan fajl na scenu, a drugi neka ostane u radnom delu. Prikažite promene:
    a. Između fajla na sceni i radne kopije
    b. Poslednjeg snimljenog i indeksa
    c. Poslednjeg komita i radne kopije
10. Resetujte indeks
11. Resetuje indeks i radne kopije.

Grananje i spajanje

Svaka snimljena verzija softvera ima svoje mesto u istoriji revizija. Sve revizije su povezane kao objekti jedne liste. Samo početna revizija nema roditeljsku i samo poslednja nema decu objekte. Ono što je na ovom mestu bitno da zapazite da revizija može imati dva deteta pri grananju, odnosno dva roditelja pri spajanju.



Slika 22. Primer istorije koja sadrži grananje i spajanje grana

Na prethodnoj slici su prikazane dve grane: dev i master. Svaka od grana ima svoju oznaku i poslednje snimanje.

Pri snimanju jedne revizije formira se jedinstveni identifikator dužine 160 bita koristeći SHA-1 šifrovanje. Pri generisanju ovog identifikatora učestvuju:

- Fajlovi koji učestvuju u jednoj reviziji

- Prateći (meta) podaci (poruka pri snimanju, ime autora,…)
- Heš vrednost od roditeljskog komit-a.

Ovako dobijena vrednost obezbeđuje sigurnost i pouzdanost sadržaja revizije kao i njenu povezanost sa prethodnim revizijama.

Grananje

Grananjem se od jedne grane formira još jedna. Nova grana se kreira primenom komande:

```
git checkout -b naziv [start]
git branch naziv [start]
```

gde je:

naziv – naziv nove grane

start – početna tačka tj. lokacija za novu granu. Može biti jedinstveni identifikator heš vrednost ili tag, ako postoji. Ako se ne navede koristi se tekuća lokacija.

Ovom komandama kreira sa nova grana. U slučaju prve komande tekuća grana se prebacuje na novu kreiranu granu.

Na primer:

```
admin@ZC-A7 MINGW64 /c/myRepo (dev)
$ git checkout master
Switched to branch 'master'

admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git status
On branch master
nothing to commit, working tree clean

admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git checkout -b feature-Mx
Switched to a new branch 'feature-Mx'

admin@ZC-A7 MINGW64 /c/myRepo (feature-Mx)
$
```

Obratite pažnju na prateće poruke. Vidi se da se generiše nova grana i istovremeno se prelazi na tu novu granu. Uvek postoji samo jedna grana koja je trenutno aktivna tj. na kojoj se izvode operacije.

Prelaz na drugu granu

Prelaz na postojeću granu obavlja se primenom gotovo iste komande.

```
git checkout naziv
```

naziv – ime grane na koju se prelazi.

```
admin@ZC-A7 MINGW64 /c/myRepo (feature-Mx)
$ git status
On branch feature-Mx
nothing to commit, working tree clean

admin@ZC-A7 MINGW64 /c/myRepo (feature-Mx)
```

```
$ git checkout dev
Switched to branch 'dev'

admin@ZC-A7 MINGW64 /c/myRepo (dev)
$
```

Preuzimanje nekog fajla

Naredba `git checkout` koristi se i za preuzimanje određenog fajla koji je na nekoj drugoj grani.

`git checkout` *`nazivgrane -- file1 file2`*

Na primer:

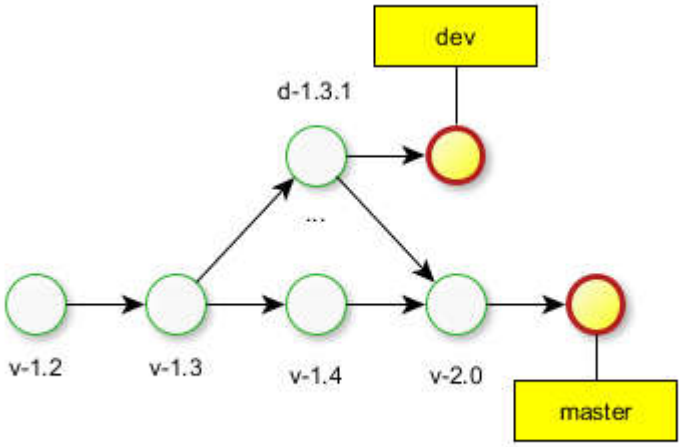
```
$ git status
On branch dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        aaa.xml

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

admin@ZC-A7 MINGW64 /c/myRepo (dev)
$ git checkout master -- dokument5.txt

admin@ZC-A7 MINGW64 /c/myRepo (dev)
$ git status
On branch dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   dokument5.txt

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

        aaa.xml


admin@ZC-A7 MINGW64 /c/myRepo (dev)
$
```

Spajanje grana

Neka u glavnoj grani postoje 4 fajla. Nakon grananja (git checkout –b dev) u novoj grani se nastavi rad i izvrše izmene na jednom fajlu (npr. dokument1.txt), zatim snimanje. U glavnoj grani na drugom fajlu (dokument2.txt) se izvrše izmene koje se zatim snime. Zatim se radi spajanje. Obično se vrši spajanje pomoćne grane sa glavnom. Dakle, ako smo na glavnoj grani, spajanje se postiže primenom komande

`git merge` *`name –m “message”`*

Gde je:
name - naziv grane sa koje se preuzimaju izmene pri spajanju, a
“*message*“ - poruka koja se postavlja pri snimanju koje se obavlja automatizovano pri spajanju.




```
// v-1.3
git checkout dev
// rad na nekim dok.;
git commit –m "d - 1.3.1"
git checkout master
// rad na nekim dok.;
git commit –m "v - 1.4"
git merge dev –m "v - 2.0"
```

Slika 25. Grananje i spajanje

Konflikti

Konflikt nastaje ako vrši izmena na istom fajlu u više grana koje se spajaju. Pri spajanju Git razdvaja dve varijane:

- Tekstualni fajlovi. Spajaju se po linijama.
  - Ako je jedna linija promenjena samo u jednoj grani onda se može uraditi automatski spajanje.
  - Ako je ista linija promenjena u više grana onda Git prijavljuje konflikt. Zone konflikta su unutar oznaka <<<<<<< >>>>>>>. U ovom slučaju Git zahteva spajanje fajlova eksplicitnim učešćem korisnika.
- Binarni fajlovi. Uvek se označava konflikt.

Rešavanje konflikata

Da bi Git uradio spajanje dve revizije potrebno je da se reše konflikti ukoliko postoje. Ako se neki fajlovi ne spoje ostaju u radnom delu i označeni su kao „unmerged“.
Ostali fajovi kod kojih ne postoji konflikt, kao i preteći metapodaci se automatski dodaju u indeks (na scenu).
Postoje dva načina za rešavanje konflikata.

1. Ručno rešavanje. Ovo rešavanje podrazumeva editovanje fajlova, a zatim eksplicitno dodavanje/brisanje nekih od njih primenom komandi: `git add file` ili `git rm file`
2. Pokretanje nekog od alata za rešavanje konflikata tj. spajanje (xxdiff, kdiff3, beyond compare,...)

Pošto se konflikti fajlova u indeksu reše vrši se snimanje: `git commit`

Primer

Neka je sledeći redosled aktivnosti u repozitorijumu.
Korak 1. Repozitorijum inicijalizovan sa jednim dokumentom: git init, git add .

Korak 2. Formiramo novu granu dev: git checkout –b dev
Korak 3. Dodamo dva nova fajla i snimimo ih u novoj granu: dokument2.txt i dokument-Y.txt. Ovo možemo uraditi u dva koraka: git add .+ git commit –m „***“
Korak 4. Prebacimo se u glavnu granu, dodamo dva nova fajla takođe, takođe u dva koraka, pri čemu se jedan od njih poklapa imenom sa jednim od fajlova u grani dev, ali sa svojim sadržajem.
Korak 5. Radimo spajanje. Dakle, u obe grane sve izmene su snimljene tj. Urađen je commit.
Pokrećemo komandu:

```
$ git merge dev
Auto-merging dokument2.txt
CONFLICT (add/add): Merge conflict in dokument2.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Detektovan je konflikt u fajlu dokument2.txt.
Kako rešiti konflikt? Najpre pogledajmo razliku:

```
$ git diff
diff --cc dokument2.txt
index 3a88407,115b8f6..0000000
--- a/dokument2.txt

+++ b/dokument2.txt
@@@ -1,1 -1,1 +1,5 @@@
- dokument2 u glavnoj grani
 -Neki tekst...
++<<<<<<< HEAD
++dokument2 u glavnoj grani
++=======
++Neki tekst...
++>>>>>>> dev
```

Master: dokument2.txt

dokument2 u glavnoj grani

dev: dokument2.txt

Neki tekst...

Sada ćemo pokrenuti alat za spajanje dokumenata:

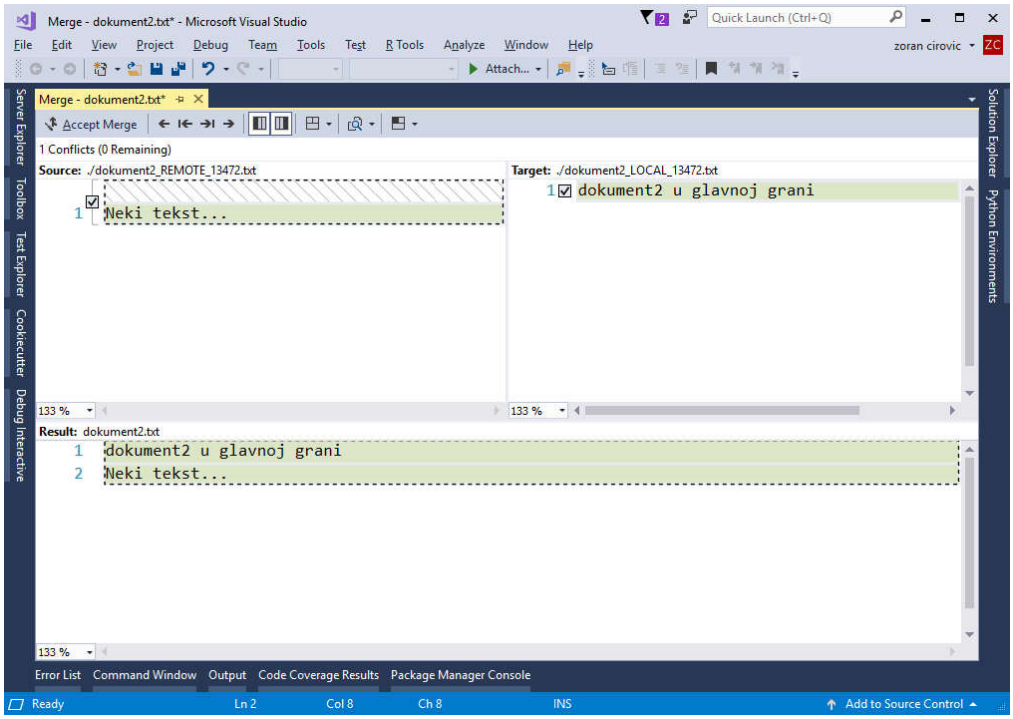
```
$ git mergetool dokument2.txt
Merging:
dokument2.txt

Normal merge conflict for 'dokument2.txt':
  {local}: created file
  {remote}: created file
Hit return to start merge resolution tool (vsdiffmerge):
```

Obratite pažnju da će pritiskom na ENTER da bude otvoren neki alat za spajanje.

Git nema sopstveni ugrađeni alat za vizualni prikaz i rešavanje konflikata. Kad nađete program koji vam odgovara možete ga postaviti kao merge.tool alat:

```
git config --global merge.tool /putanja/do/programa
```

Na slici je prikazan jedan takav alat koji dolazi uz VisualStudio - vsdiffmerge.



Slika 26. Alat u okviru Visual Studio radnog okruženja za rešavanje konflikata pri spajanju

Nakon snimanja promena imamo sledeći status:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:

        new file:   dokument-Y.txt

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

        both added:      dokument2.txt

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

        dokument2_BACKUP_13472.txt
        dokument2_BASE_13472.txt
        dokument2_LOCAL_13472.txt
        dokument2_REMOTE_13472.txt
        sh.exe.stackdump
```

Sada eksplicitno dodajemo dokument koji smo dobili spajanjem u glavnoj grani:

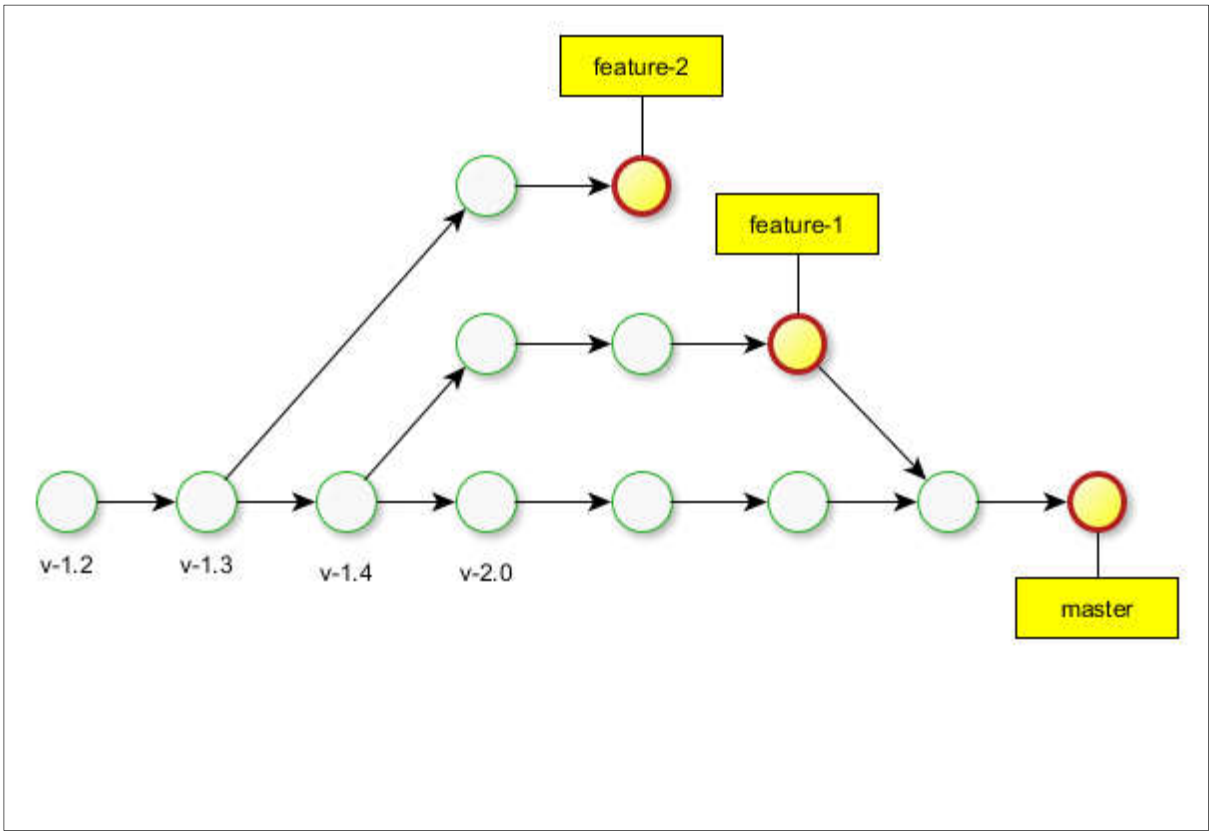
```
$ git add dokument2.txt

admin@DESKTOP-KUT132H MINGW32 /d/myRepo (master|MERGING)
$ git commit -m "v-2.0"
[master fa5eb98] v-2.0
```

Obratite pažnju da je u tekućem folderu generisano nekoliko fajlova koje nisu deo verzije i koje možete obrisati ili uključiti u praćenje. Ako ste sigurni da ove fajlove nikada nećete dodavati u reviziju i želite da ih nakon spajanja automatski alat za spajanje briše, onda je moguće to postaviti u podešavanja opcijom:

`git config --global mergetool.keepBackup false`

U suprotnom potrebno je da obrišete ove fajlove.

Brisanje grana



Slika-27. Istorija revizija

Komanda za brisanje grane:

```
git branch -d name
```

Gde je:

name - naziv grane koja se briše.

Pri tome treba imati na umu da brisanje ne može da se izvrši u slučaju ako pokušamo da brišemo:

1. tekuću granu – HEAD
2. granu koja još nije spojena na tekuću.

Za gornju sliku akcije i prateći rezultat bio bi:

```
$ git branch -d feature-1
Deleted branch feature-1 (was 54149ea).
$ git branch -d feature-2
error: The branch 'feature-2' is not fully merged.
If you are sure you want to delete it, run 'git branch -D feature-2'.
$ git branch -d master
error: Cannot delete the branch 'master' which you are currently on.
```

```
git branch [-d][-D][-q][-x][-X][-d][-f]<path>
```

Za dodatni rad pogledati: Fastforward, rebase, cherry-pick, git merge eksperimentalna-grana --no-commit

Vežbe

0. koristite "gitk - all" za prikaz svih grana
(i ne zaboravite da pritisnete F5 posle svake komande da biste vizualizovali promene)
1. napravite novu granu pod nazivom "develop"
2. napravite neke obaveze u ovoj grani
3. Vratite se na granu "master" i napravite nešto urezivanja
4. spajanje grana "razviti" u "majstor"
5. napravite novi urez u svakoj grani tako da generišete konflikt (uredite
isti deo filma)
6. spajanje grana "razviti" u "majstor", i fi konflikt
7. spojiti "majstora" u "razviti"

Udaljeni/mrežni repozitorijum

Mrežni repozitorijum se koristi kada se organizuje timski rad ili radi dostupnosti drugim korisnicima. Taj repozitorijum je dostupan članovima tima koji preko njega rade sa promenama:

- Preuzimaju se promene sa mreže koje su nastale akcijama drugih učesnika.
- Postavljanje promena koje jedan učesnik uradi.

Git hosting

Ukoliko se odlučimo za rad sa mrežnim repzitorijumom, bilo iz razloga timskog rada ili dostupnosti za druge korisnike moramo definisati da li taj repozitorijum formirano na sopstvenom serveru ili ćemo ga formirati na nekom od javno dostupnih servera. Većina ozbiljnih hostova namenjenih čuvanju verzija podržava Git. Neki od najpopularnijih su:

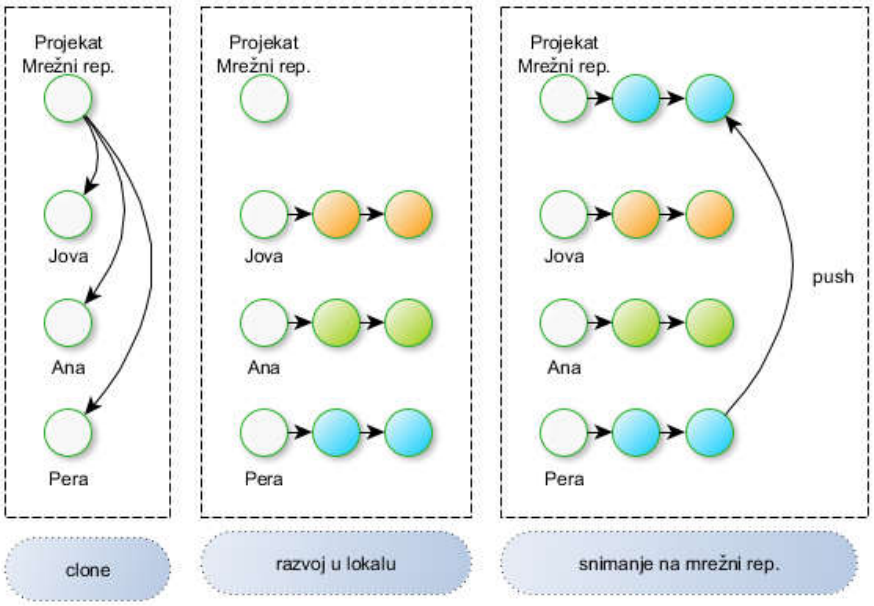
- http://github.com – Ukoliko želite da čuvate podatke privatno onda se mora platiti, inače je besplatan za čuvanje podataka koji bi bili dostupni drugim koristnicima.
- http://bitbucket.org – Takođe popularan, s tim što je besplatan i za privatne repozitorije.
- http://code.google.com – Za projekte otvorenog koda.
- http://sourceforge.net – Za projekte otvorenog koda.

Timski rad

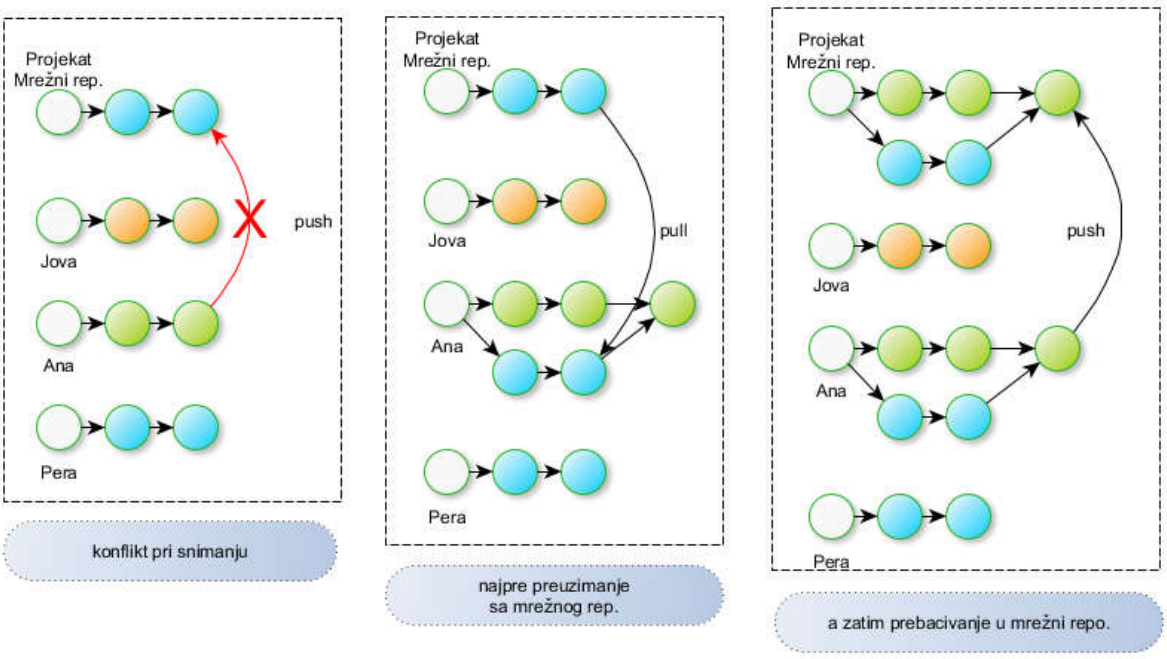
Tipičan timski rad obuhvata nekoliko koraka. To su:
Korak 1. Kloniranje repozitorijuma.
Korak 2. Razvoj lokalnih verzija.
Korak 3. Prebacivanje lokalne verzije na mrežnu.
Korak 4. Konflikt pri prebacivanju.
Korak 4. Rešavanje konflikta.

Pogledajmo sledeći primer. Ako u timskom radu učestvuje više korisnika njihov prvi korak je preuzimanje tekuće verzije sa zajedničke lokacije. Ta lokacija može biti mrežni folder ili folder negde na nekom od hosting sajtova. Ovo je prikazano na narednoj slici u prvom koraku.

Sledeći korak je lokalni razvoj. Svaki od korisnika kreira svoj razvoj, naravno zasnovan na zajedničkom projektu koji je preuzet sa mrežnog repozitorijuma.
Zatim neko od korisnika, neko ko prvi odluči da snimi primene na mrežni repozitorijum, prebacuje svoju verziju. Njegova verzija je nastala od mrežne i promene koje je on učinio lako se prebacuju na mrežu, tako da se na njoj postavlja verzija koja odgovara promenama kod tog korisnika.



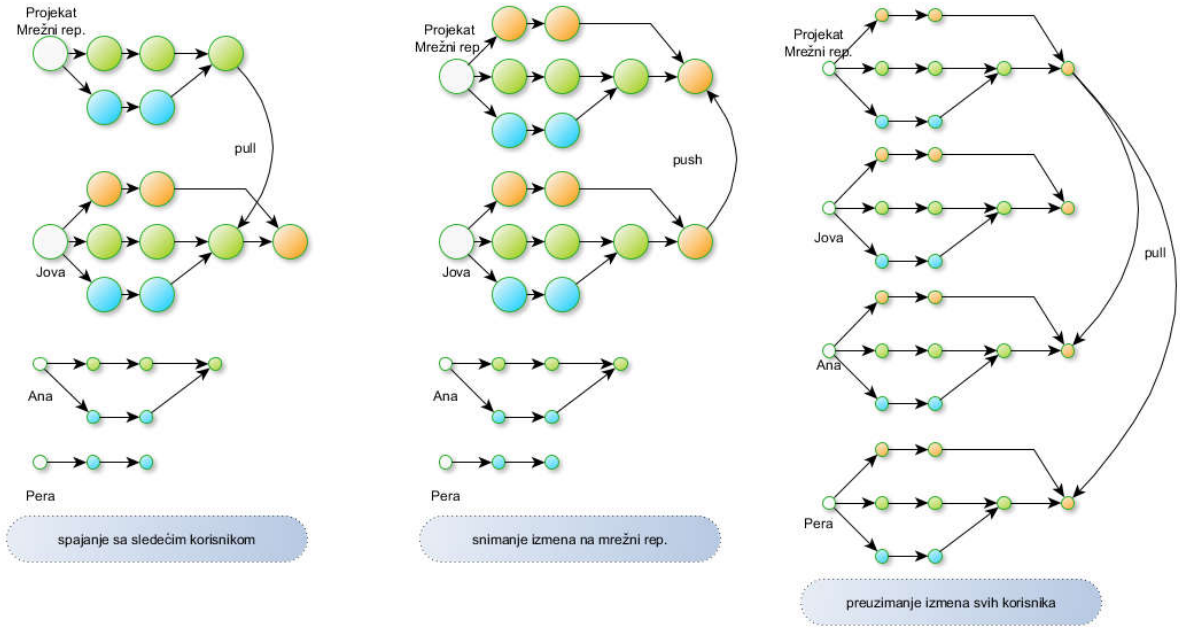
Слика-28 Preuzimanje zajedničke verzije, lokalne izmene i promene prvog korisnika



Slika-29. Postupak sinhronizacije

Sledi korak u kome sledeći korisnik treba da prebaci svoju verziju. Njegove izmene lako nailaze na konflikt sa verzijom na mreži. Razlog je promena koja je usledila u međuvremenu tj. novim snimanjem od strane prethodnog korisnika. U ovom slučaju drugi korisnik treba da preuzme promene sa mreže,

izvrši spajanje sa svojim izmenama, a zatim se tako spojena verzija se ponovo šalje na mrežu. Takva verzija sadrži podatke od oba korisnika.



Slika-30 Sinhronizacija ostalih korisnika

Postupak se nastavlja na isti način sa ostalim korisnicima. Svaki od njih prvo preuzima sve sa mreže, vrši spajanje sa svojim promenama, a novu verziju koja sadrži sve spojeno šalje ponovo na mrežu.

Rad sa udaljenim repozitorijumima

Git radi sa udaljenim repozitorijumima tako što kreira njihovu lokalnu sliku. Pri tome se može raditi sa više repozitorijuma istovremeno. Svaki od njih je identifikovan nekim lokalnim jedinstvenim imenom – alijasom.
Kada se radi sa jednim udaljenim repozitorijumom, obično se takav naziva **origin**.

Na sličan način se vrši imenovanje grana: remote/name/branch. Na primer

- remote/origin/master predstavlja glavnu granu udaljenog repozitorijuma origin.
- master predstavlja glavnu lokalnu granu

Dodavanje udaljenog repozitorijuma

Dodavanje udaljenog repozitorijuma u tekući zahteva poznavanje URL lokacije istog. U tom slučaju vrši se komandom:

```
git remote add name url
```

gde je:

- *name* – lokalni alijas koji identifikuje udaljeni repozitorijum, pošto ih može biti više,
- url – lokacija udaljenog repozitorijuma

Primer:

```
git remote add origin https://github.com/zcirovic/ist.primer1.git
```

```
git remote add origin https://zcirovic@bitbucket.org/zcirovic/ist.primer1.git
```

Moguće je raditi sa više udaljenih repozitorijuma, naravno koristeći različite nazive. Pregled korišćenih repozitorijuma i prateće *url* lokacije možete dobiti komandom: **`git remote -v`**. Na primer:

```
$ git remote -v
netOrigin       //ZC-A7/netRepo (fetch)
netOrigin       //ZC-A7/netRepo (push)
origin  //ZC-A7/wwwRepo (fetch)
origin  //ZC-A7/wwwRepo (push)
```

Ukoliko se predomislite i želite da uklonite udaljeni repozitorijum to se čini na sličan način:

```
git remote rm name
```

Dodavanje udaljenog repozitorijuma znači kreiranje posebne grane na lokalnom repozitorijumu koja je zadužena za rad sa udaljenim repozitorijumom i ima specifičan naziv. Nazi se sastoji od delova koji označavaju da se radi o grani za rad sa udaljenim repozitorijumom kao i naziv grane. Pogledajmo prikaz svih grana:

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

Snimanje lokalnih podataka na udaljeni

Snimanje lokalnih izmena na udaljeni repozitorijum vrši se komandom

```
git push [--tag]
```

Ova komanda snima tekuću granu

- ako promene u grani prate rast grane, onda se lokalne promene šalju na udaljenu granu
- ako ne, onda se ništa ne snima
- u slučaju konflikta prvo treba da se vrši izvršavanje git pull naredbe.

Snimanje nove grane na udaljeni repozitorijum

Za svaku granu koja je ažurna za dodavanje upstream referencu za mrežni repozitorijum koju koristi git-pull i druge naredbe bez argumenta. Dakle, nakon što ste snimili sa *push -u* vašu lokalnu granu ova će se automatski povezati sa udaljenom granom, pa možete koristiti git pull bez ikakvih argumenata.

```
git push -u [--tag]
```

Moguća je varijanta eksplicitnog zadavanja objekta za snimanje.
git push -u destRep ref [ref . . .]

gde je

Ref – lokalna referenca ili tag koji se snima udaljeno na destRep

Primer:

```
$ git push -u origin master
```

```
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (18/18), 1.46 KiB | 748.00 KiB/s, done.
Total 18 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/zcirovic/ist.primer1.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Preuzimanje izmena sa udeljanog repo

Preuzimanje verzija sa udaljenog repozitorijuma obavlja se pomoću komandi:

```
git fetch
git pull
```

U okviru ovog ažuriranja lokalnog repozitorija na osnovu udaljenog radi se:

- Preuzimanje novih snimanja (komita) sa udaljenog repozitorijuma
- Ažuriranje reference sa remote/name/* tako da odgovaraju novoj poziciji na korišćenoj grani.

Naredba fetch preuzima promene sa mrežnog skladišta i prebacije ih na lokalnu granu koja je zadužne za rad sa udaljenom (na primer: origin/master). Pogledajte sledeći primer:

```
$ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From //ZC-A7/wwwRepo
   d2ae608..b104682  master     -> origin/master

admin@ZC-A7 MINGW64 /c/myRepo4/wwwRepo (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

Ako pokušamo da se prebacimo na ovu granu, dobijamo sledeću poruku:

```
$ git checkout origin/master
Note: checking out 'origin/master'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at b104682 v4

admin@ZC-A7 MINGW64 /c/myRepo4/wwwRepo ((b104682...))
$
```

Očigledno da ova grana nije ista kao grane sa kojima inače radimo i da služi da pruži dodatnu funkcionalnost samom Git-u. Ono što ona ipak pruža je mogućnost spajanja.

```
admin@ZC-A7 MINGW64 /c/myRepo4/wwwRepo (master)
```

```
$ git merge origin/master
Updating d2ae608..b104682
Fast-forward
 dokument4.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 dokument4.txt
```

Spajanje sa udaljenim izmenama na lokalnoj grani

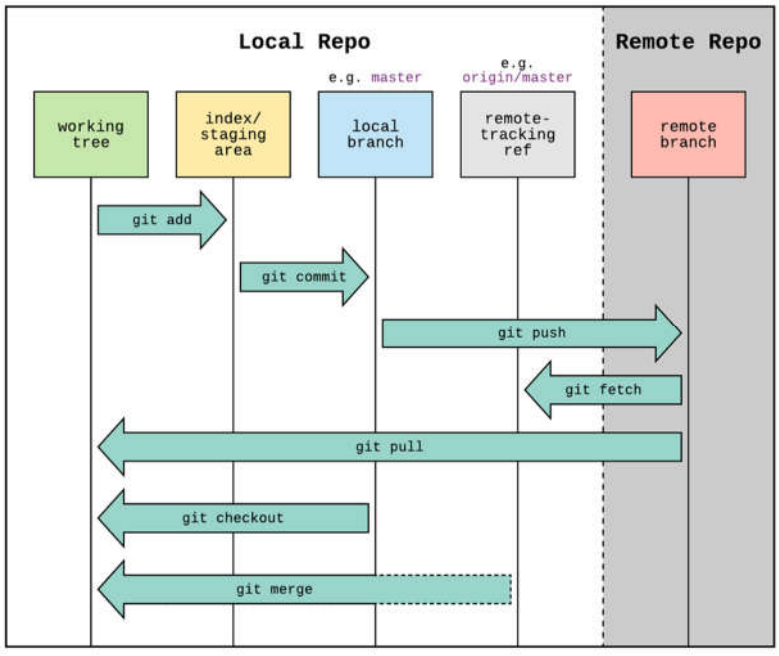
Spajanje udaljenih izmena sa tekućom lokalnom granom vrši se pomoću eksplicitne komande **git merge**. Na primer: git merge origin/master
U praksi obično se koristi `git pull`, koji je alijas za dve komande: **git fetch+git merge.**

`git pull`	`git fetch` `git merge origin/master`

Upotreba grana

lokalna pomoćna grana za rad sa udaljenim repozitorijumom, obično `origin/master`, je sinhronizovana sa mrežnom u nekom stanju.

Push - vrši se prebacivanje lokalnih kopija na mrežni repo. Ne utiče na stanje na pomoćnu granu.
Fetch - vrši se preuzimanje mrežnih verzija na lokalnu pomoćnu. Neophodno je uraditi spajanje da bi se te izmene koristile u lokalnoj verziji.
Pull - radi preuzimanje i spajanje mrežne verzije sa lokalnom.
Sve ovo je prikazano na narednoj slici.



Remote primer

- (Git init –bare –shared // remote repo)
- Git init // local repo
- Git commit
- Git remote add origin shared_url
- Git push //nista
- Git push –u origin master
- Git commit
- Git commit
- Git push
- Neki drugi korisnik snima svoje verzije
- Git commit
- Git push ---- konfilkt
- Git fetch – preuzima nove verzije na origin/master granu
- Radi se spajanje u lokalu. git merge origin/master i resavanje konflikata u lokalu
- Zatim se radi git pull, tj ponovo se preuzimaju izmene, a sada
- Git push – radi bez konflikta

Postavljanje udaljene grane kao tekuće

Standardnom komandom *checkout*, dakle

```
git checkout name
```

Gde je name ime grane koja nije u lokalu. Dakle, ako Git ne nađe *name* granu u lokalu traži je na udaljenom repozitorijumu. Ako je nađe, kreira se lokalna grana i istovremeno se konfiguriše da prati udaljenu granu.

```
$ git branch --all
* master
remotes/origin/master
remotes/origin/new-fancy-feature
$ git checkout new-fancy-feature
Branch new-fancy-feature set up to track remote branch new-fancy-feature from
origin.
Switched to a new branch 'new-fancy-feature'
$ git branch --all
master
* new-fancy-feature
remotes/origin/master
remotes/origin/new-fancy-feature
```

Kloniranje repozitorijuma

Kloniranje repozitorijuma predstavlja formiranje lokalne kopije udaljenog repozitorijuma na lokalu. Kloniranje omogućava dalji rad sa lokalnom granom preko koje će se obavljati sinhronizacija podataka sa udaljenim repozitorijumom. Komanda je:

```
git clone url [dir]
```

Zapravo, git clone je skraćena komanda tj. sekvenca nekoliko komandi:

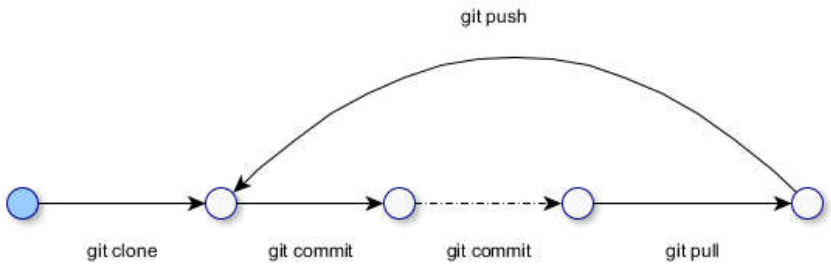
- git init dir
- cd dir
- git remote add origin url
- git fetch

- `git checkout master`

U praktičnoj upotrebi retko se koristi ova sekvenca komandi već jednostavno komande: `git clone` odnosno `git pull` .

Tipični sekvenca komandi

Najčešće korišćena sekvenca komandi u radu sa mrežnim repozitorijumom koja obezbeđuje sinhronizaciju data je na slici 33.



Slika-33. Tipična sekvenca