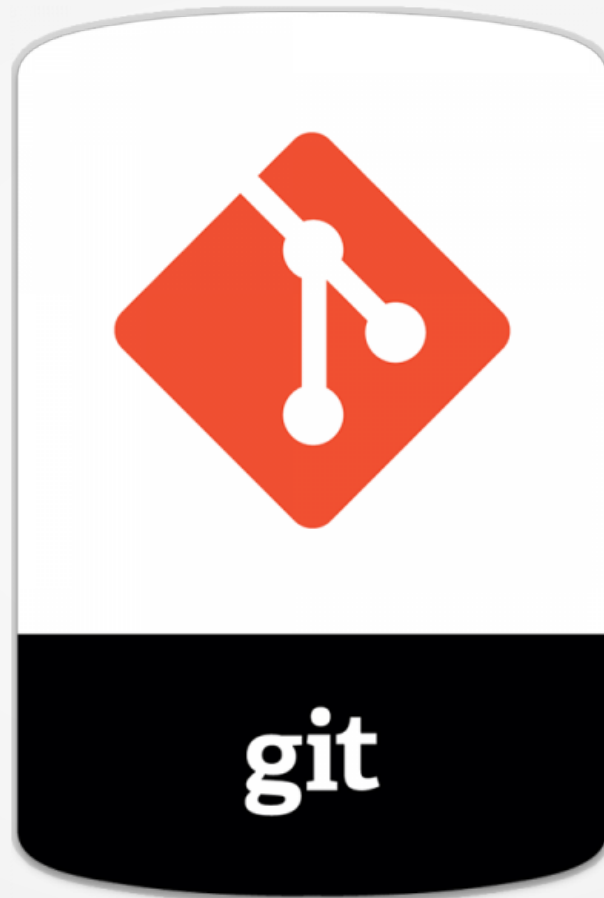


# Git



# Sistemi za pracenje koda

- Postoje dve vrste sistema za pracenje I kontrolu verzija
  - **CVCS** – Sistem sa centralizovanom verzionom kontrolom
    - SVN, Perforce, CVS.
    - Svi podaci o verzijama se nalaze na centralnom serveru, dok korisnici imaju dostupnu samo trenutnu verziju na kojoj rade. Ovaj princip rada omogucava lakse odrzavanje I administriranje, no sigurnost podataka moze biti problem ukoliko dodje do otkaza sistema (gube se sve informacije).
  - **DVCS** – Sistem sa decentralizovanom verzionom kontrolom
    - Mercurial, Bazaar, Darcs, **Git**
    - Korisnici pored poslednje verzije, preuzimaju I kompletnu bazu o svim verzijama na tom repozitorijumu. Sigurnost podataka nije problem, jer u slucaju otkaza sistema, dovoljno je da samo jedan od klijenata postavi podatke na server.

# Git

- Kloniranjem nekog repozitorijuma putem Git-a, kopira se celokupna baza o projektu do tog trenutka, sto nam omogucava da lokalno koristimo sve verzije fajlova (koje postoje do trenutka kad smo izvrшили kloniranje).
- Podaci se sad nalaze u lokalni, I odziv je brzi nego kad se podaci nalaze na udaljenom serveru.
- Git princip cuvanja podataka se zasniva na cuvanju celokupnog stanje sistema, u tom trenutku se snima tzv. snapshot svakog fajla projekta.
  - Ukoliko fajl nije promenjen, onda se cuva pokazivac na njegovo prethodno stanje
- Prostor koji zauzima projekat na Git-u, je mnogo manji od prostora koji isti taj projekat zauzima na SVN-u.

# Git – Osnovni pojmovi

- **Working directory** – radni, lokalni folder u kome menjamo i kreiramo fajlove koji su obuhvaceni verzionisanjem
- **Index – Prostor pripreme (Staging area)** – svi fajlovi, pre nego sto se komituju, moraju da budu pripremljeni za to, i zato se smestaju u ovaj pripremljeni deo. Samo oni fajlovi koji se nalaze ovde, bice komitovani. Kako Git pravi snapshot svih fajlova prilikom komita, korisceni ovaj prostor pripreme, sistem nece morati da pravi snapshot svaki put kad god imamo neki fajl spreman.
- **Commit – Komitovanje** – je naredba sa kojom pravimo snimak promena stanja fajlova u odnosu na predhodno stanje. U Git-u nikad ne commit-ujemo na udaljeni repozitorijum, nego snimamo u lokalni repozitorijum na nasem racunaru.
- **Repository (Repo) – Repozitorijum ili Skladiste** je prostor gde se smestaju tzv. Snapshot-ovi stanja fajlova prilikom commit-ovanja. Ovde se nalaze svi metapodaci i baza podataka o nasem projektu. Kada neko preuzima (klonira) projekat, kopira mu se ovaj direktorijum.
- **Clone – Kloniranje** – kopiranje udaljenog repozitorijuma, ali tako da novi (lokalni) repozitorijum ostaje svestan da je on kopija nekog udaljenog repozitorijuma.

# Git – Osnovni pojmovi

- **Push** – naredba sa kojom saljemo svoje izmene na lokalnom repozitorijumu, nazad na originalnu lokaciju (npr. GitHub).
  - Za koriscenje ove naredbe je potrebno da imamo ovlascenja, ukoliko ih nemamo koristimo naredbu **Pull Request** – kratka poruka vlasniku udaljenog repozitorijuma da imamo izmene koji bi on mogao da preuzme u svoj repozitorijum. Poruka sadrzi opis izmena koje smo napravili, kao i adresu naseg repozitorijuma.
- **Branche** – grana projekta.
- **Bare git repository** – repozitorijum koji nema radnu verziju projekta (radni direktorijum), sadrzi samo .git repozitorijum. Takav repozitorijum se uglavnom nalazi na nekom serveru, i na njega se ne **commit**-uje, nego se izvrsi **push** sa lokalnih repozitorijuma
- **Non-Bare git repository** – repozitorijum koji sadrzi i .git direktorijum, kao i **working tree**
- **Head** – zamisljeni pokazivac na trenutni commit unutar git-a. Uglavnom je to poslednji commit u grani u kojoj se nalazimo, mada moze biti i bilo koji drugi. Ako pokazivac ukazuje na commit koji nije poslednji u grani – repozitorijum je u **detached head** stanju
- **Git** – uglavnom se koristi putem komandne linije, dok ...
- **GitHub** – web based GUI, ali i GitHub Desktop App

# Git - Instalacija

- GNU/Linux, MacOS, Windows
- Download: [git-scm.com](https://git-scm.com) + nudi linkove ka GUI Git klijentima
- GNU/Linux Terminal:

```
vmm@workstation: ~  
File Edit View Search Terminal Help  
vmm@workstation:~$ sudo apt-get install git
```

- Nakon instalacije mozete proveriti da li je git instaliran kao I verziju istog:

```
vmm@workstation: ~  
File Edit View Search Terminal Help  
vmm@workstation:~$ git --version  
git version 2.7.4  
vmm@workstation:~$
```

# Git - dodatno

- Konfiguracija imena i adrese:

```
vmm@workstation: ~  
File Edit View Search Terminal Help  
vmm@workstation:~$ git config --global user.name IME  
vmm@workstation:~$ git config --global user.email eAdresa@email.com  
vmm@workstation:~$
```

- Konfiguracioni fajl se nalazi u /home/user direktorijumu: (/home/user/.gitconfig)

```
vmm@workstation: ~/Desktop  
File Edit View Search Terminal Help  
vmm@workstation:~/Desktop$ cat /home/vmm/.gitconfig  
[user]  
    name = IME  
    email = eAdresa@email.com  
vmm@workstation:~/Desktop$
```

- **.gitignore** – fajlove koje ne zelimo da pratimo, ukljucujemo u .gitignore fajl
  - Nakon sto izaberemo folder u okviru koga cemo pratiti promene na fajlovima (folder u kome ce biti smesten repozitorijum), potrebno je izbaciti sa liste pracenja promena neke nepotrebne fajlove, koji se neretko zatetku unutar foldera (fajlovi koje koristi text editor, ili ih kompajler stvara, ...)

# Git - dodatno

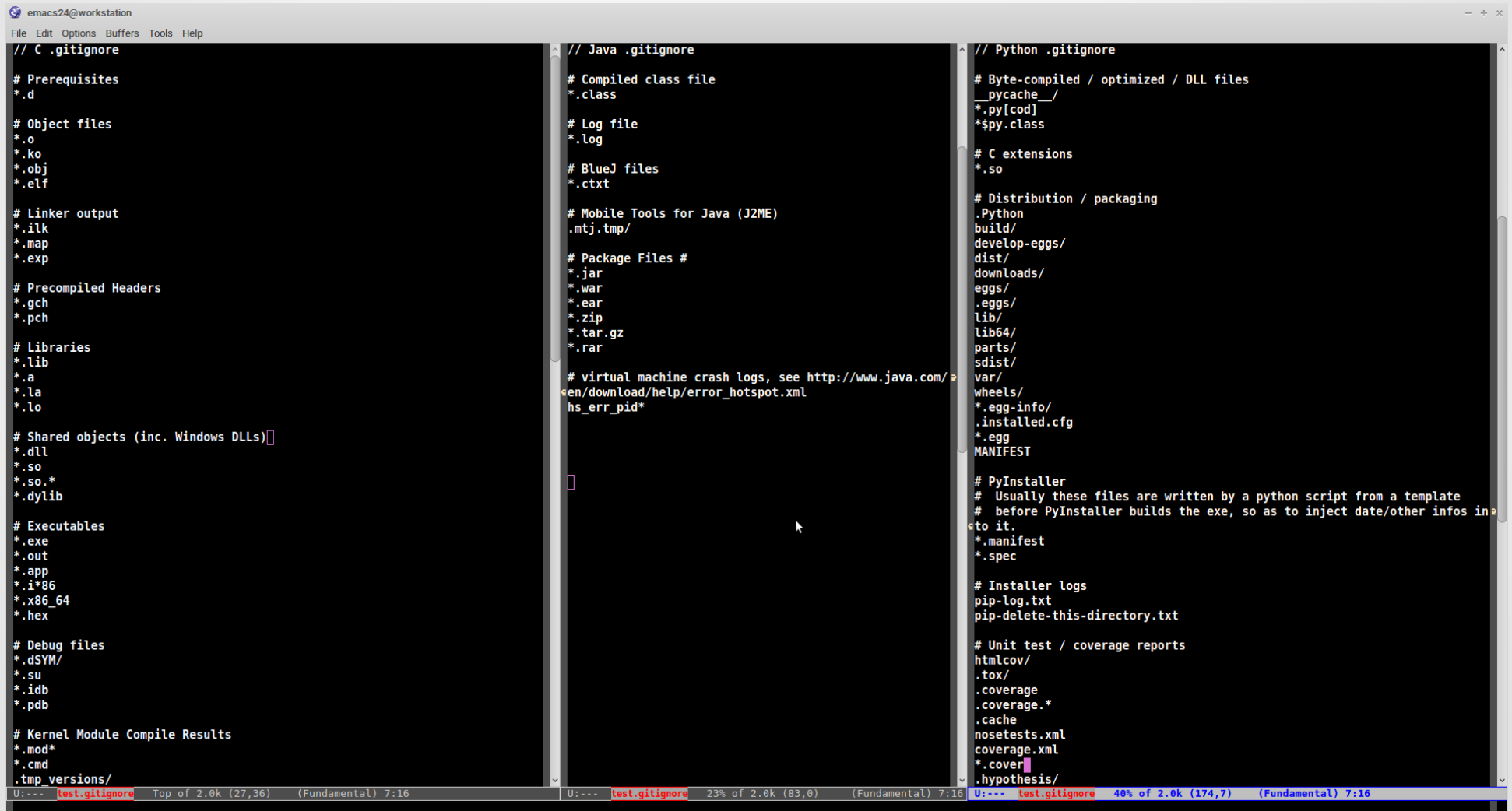
- Unutar .gitignore fajla, fajlove i foldere koje ne zelimo da pratimo obelezavamo:

```
// Upisemo ekstenzije fajlova koje ne zelimo da pratimo:  
// npr: .iso .jar .sql .log .class  
  
// Upisemo foldere koje ne zelimo da pratimo:  
// npr: ime_foldera  
  
// U slucaju da unutar foldera postoji fajl koji ipak treba da se prati:  
// npr: !ime_fajla.txt  
  
// Upisemo folder i njegove podfoldere koje ne zelimo da pratimo:  
// npr: ime_foldera/ime_podfoldera  
  
// Upisemo folder i sve njegove podfoldere koje ne zelimo da pratimo:  
// npr: ime_foldera/*  
  
// Upisemo sve objekte i arhive koje ne zelimo da pratimo:  
// npr: *.slo *.obj
```

- Ukoliko vec pratimo neki fajl, pa ga naknadno ubacimo u .gitignore, git nece prestati da ga prati, moramo pre ubacivanja u .gitignore narediti git-u da prestane da ga prati
  - git rm -cached ime\_fajla.extenzija***



# Git - .gitignore examples



The screenshot shows the Emacs editor interface with three .gitignore files open side-by-side. The top bar indicates the editor is 'emacs24@workstation'. The status bar at the bottom shows the current file is 'test.gitignore' and the cursor is at line 7, column 16.

```
// C .gitignore
# Prerequisites
*.d

# Object files
*.o
*.ko
*.obj
*.elf

# Linker output
*.ilk
*.map
*.exp

# Precompiled Headers
*.gch
*.pch

# Libraries
*.lib
*.a
*.la
*.lo

# Shared objects (inc. Windows DLLs)
*.dll
*.so
*.so.*
*.dylib

# Executables
*.exe
*.out
*.app
*.i*86
*.x86_64
*.hex

# Debug files
*.dSYM/
*.su
*.idb
*.pdb

# Kernel Module Compile Results
*.mod*
*.cmd
.tmp_versions/

// Java .gitignore
# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
*.mtj.tmp/

# Package Files #
*.jar
*.war
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*

// Python .gitignore
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[co]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
*.egg-info/
*.installed.cfg
*.egg
MANIFEST

# PyInstaller
# Usually these files are written by a python script from a template
# before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
.hypothesis/
```

# Git - naredbe

- **Instanciranje repozitorijuma**

- Kada zelimo da pratimo fajlove u nekom folderu, potrebno je, unutar njega, da iniciramo pravljenje git repozitorijuma komandom **git init**, koja unutar naseg foldera kreira novi **.git** folder (tj. Repozitorijum) u kome ce se nalaziti celokupna baza o nasem projektu
- Ukoliko vise ne zelimo da git prati nas projekat, dovoljno je da obirsemo ovaj folder

- **Git status**

- Kontrola da li imamo promene na fajlovima u odnosu na poslednji commit, vrsi se naredbom: **git status**
- Ako je stanje na radnoj verziji naseg projekta potpuno isto kao i u poslednjoj verziji git repozitorijuma, nemamo nista za commit-ovanje, u suprotnom git ce nas obavestiti koji su fajlovi izmenjeni

- **Pregled promene koda**

- Ispisivanja svih promena trenutnog stanja projekta na lokalu i poslednje verzije projekta snimljene u repozitorijumu, vrsi se naredbom: **git diff**
- Razlika izmedju commit-a na istoj grani: **git diff HEAD^ HEAD**
  - Ili **git diff @~..@** ili **git show** ili **git diff commit\_id HEAD** ili **git diff HEAD~2..HEAD** ili **HEAD~3..HEAD** (prethodna 2 ili prethodna 3 commit-a)
- Razlika izmedju commit-a na dve grane:
  - **git diff master sekundarna\_grana**

# Git - naredbe

- **Slanje promena u pripremni prostor (Index)**
  - Da bismo poslali snapshot promenjene verzije nekog fajla (e.g. args.c), u repozitorijum, moramo prvo da posaljemo snapshot u pripremni prostor (index). Samo fajlovi koji se nalaze u index-u, se mogu commit-ovati (poslati u repozitorijum).
  - **Slanje odredjenog fajla:**
    - *git add args.c*
  - **Slanje odredjenog direktorijuma:**
    - *git add ime\_direktorijuma/\**
  - **Slanje svih promenjenih fajlova odredjene ekstenzije iz root-a radnog foldera:**
    - *git add \*.c*
  - **Slanje svih promenjenih fajlova odredjene ekstenzije ukljucujuci i podfoldere:**
    - *git add '\*.c'*
  - **Slanje svih promenjenih fajlova:**
    - *git add .*

# Git - naredbe

- **Git commit**

- Pravimo snimak promene stanja fajlova, koje smo prethodno izabrali i prebacili u pripremni prostor
  - ***git commit -m 'Obavezan komentar'***
- Nikad se ne vrši commit u udaljeni repozitorijum, nego snimke promena pravimo samo u svoj lokalni repozitorijum (repozitorijum na racunaru)
- **Prepravljanje poslednjeg commit-a:**
  - Ukoliko nakog poslednjeg commit-a shvatimo da postoji jos nesto sto smo trebali da izmenimo, a nismo, to mozemo uraditi na dva nacina:
    - **Prvi losiji nacin:** je da nakon prepravke fajla, ponovimo citav postupak (dodamo u index, pa commit), sto dovodi do toga da nam ostaje prethodni commit, kao i snapshot koji nije validan i ne sluzi nicemu.
    - **Drugi nacin:** (izbegavamo pravljenje nepotrebnog commit-a) jeste da nakon izmene fajla u radnom direktorijumu, prebacimo u pripremni prostor samo taj fajl, kao da se spremamo za standardni commit, ali umesto commit naredbe, koristimo:
      - ***git commit - - amend -m 'Obavezan komentar'***
      - *amend govori git-u da popravi, zameni (replace) commit sa novim, koji je u svemu isti kao prethodni, osim u delu datog (navedenog) fajla, takodje komentar koji smo stavili menja prethodni, zapravo kao da prosli commit nije ni postojao*

# Git - naredbe

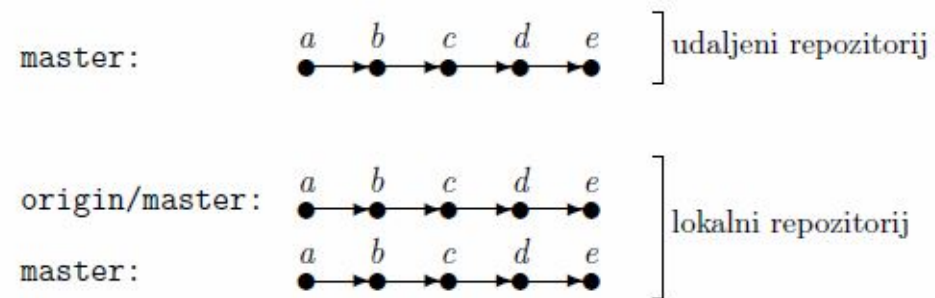
- **Prekid pracenja promena**
  - **Bez fizickog brisanja:**
    - Ukoliko smo greskom commit-ovali I neke fajlove nepotrebne za projekat, cije promene ne zelimo da pratimo (cuvamo), ali ne zelimo ni fizicki da ih obrisemo iz radnog direktorijuma, onda naredimo git-u da prestane da ih prati I da ih izbaciz repozitorijuma:
      - ***git rm - - cached args.c*** ili
      - ***git rm -r - - ime\_direktorijuma*** (-r opcija kaze git-u I sve poddirektorijume I fajlove unutar njih)
      - I nakon ove naredbe fajl/ovi se I dalje nalazi unutar Index-a, I da bismo sprecili da budu prosledjeni repozitorijumu tokom sledeceg commit-a, potrebno je da ih pre sledeceg commit-a ubacimo u .gitignore fajl
    - **Sa fizickim brisanjem:**
      - Ukoliko postoji potreba da se fajl pored izbacivanja iz repozitorijuma obrisemo I iz radnog direktorijuma, vrsi se naredba:
        - ***git rm args.c*** ili
        - ***git rm -r ime\_direktorijuma***
      - Nakon cega izvorsimo commit bez ikakvog slanja u index

# Git - naredbe

- **RAD SA UDALJENIM REPOZITORIJUMOM**

- Udaljeni repozitorijum je specijalna vrsta repozitorijuma koji sadrzi samo .git direktorijum i naziva se 'bare repozitorijum'. Takav repozitorijum nema radnu verziju projekta tj. radni direktorijum. Ovaj repozitorijum je zamisljen da se na njemu **ne radi direktan commit, vec** da se sa lokalnih repozitorijuma odradi **push**. Na udaljeni repozitorijum vlasnik (i svi sa ovlascenjima) mogu '**push**-ovati' svoje izmene dok 'obicni' saradnici mogu samo slati **pull request**.
- Lokalni repozitorijum je u specifichnoj vezi sa udaljenim repozitorijumom, on je 'svestan' postojanja udaljenog repozitorijuma. Sa tacke gledista lokalnog repozitorijuma, '**udaljeni bare repozitorijum**' sa kojim je povezan zove se **origin**, a njegova **master grana** se zove **remote/origin/master**.
- U okviru lokalnog repozitorijuma koji je povezan na udaljeni bare repozitorijum ima uvek bar dve grane:
  - **origin/master** - lokalna verzija udaljenog repozitorijuma koja mora 'rucno' da se azurira, a služi kao lokalna kopija udaljenog repozitorijuma i na nju se ne može commit-ovati ali su dostupni svi njeni fajlovi
  - **master** - nasa lokalna grana u kojoj radimo i pravimo nove commit-e
  - Razlozi za postojanje udaljenog repozitorijuma jesu: **pravljenje sigurnosne kopije na udaljeni racunar** | **saradnja vise ucesnika na istom projektu**

# Git - naredbe



# Git - naredbe

- **POVEZIVANJE LOKALNOG I UDALJENOG REPOZITORIJUMA**

- I. **Imamo lokalni – pravimo udaljeni repozitorijum**

- I. Pravljenje novog udaljenog repozitorijuma (koji ce da postane bare repozitorijum) se izvodi naredbom:

- I. ***git config - - bool core.bare true*** (ovo nije potrebno ukoliko koristimo servise kao GitHub I slicne, jer to uradi servis pri kreiranju praznog repozitorijuma)

- II. Nakon toga je potrebno lokalnom repozitorijumu dati do znanja koji udaljeni repozitorijum je njegov, naredbom:

- I. ***git remote add origin link\_do\_udalj\_repozitorijuma***

- `git remote add origin https://github.com/user_name/repo_name.git`

- III. Kontrolu dobro obavljenog posla mozemo proveriti naredbom:

- I. ***git remote -v***

- IV. Ukoliko smo dobro povezani, mozemo vrsiti slanje fajlova na udaljeni repozitorijum:

- I. ***git push -u origin master***

- V. Ukoliko zelimo da prekinemo vezu sa udaljenim repozitorijumom:

- I. ***git remote remove origin***



# Git - naredbe

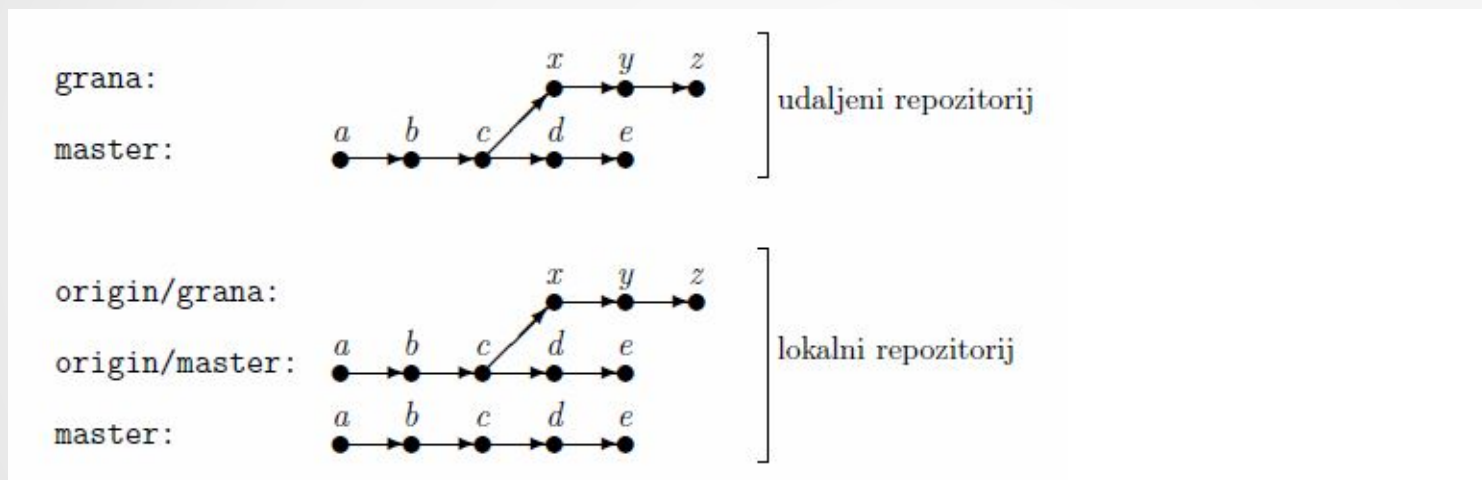
- II. Kloniranje udaljenog repozitorijuma

- Kloniranje repozitorijuma sa jednom granom

- Ako zelimo da napravimo lokalnu verziju nekog projekta koji vec ima aktiviranu kontrolu verzija tj. ima bare repozitorijum, potrebno je da kloniramo udaljeni repozitorijum. Kloniranje je kopiranje udaljenog repozitorijuma, ali tako da novi (lokalni) repozitorijum ostaje 'svestan' da je on kopija nekog udaljenog repozitorijuma. Kopirani repozitorijum cuva informaciju o repozitorijumu iz kojeg je nastao. Kloniranje se vrsi naredbom:
      - **git clone link** (git clone https://github.com/user\_name/repo.git )
      - Ovim ce se u folder u kome se nalazimo, napraviti folder koji nosi ime repozitorijuma i u njemu ce biti svi fajlovi tog projekta
      - Ukoliko zelimo da promenimo originalni naziv repozitorijuma pri kloniranju, potrebno je samo nadovezati novi naziv foldera na naredbu kloniranja
        - **git clone https://github.com/user\_name/repo.git novi\_naziv\_foldera**
    - Master grana udaljenog repozitorijuma koju smo klonirali sa tacke gledista novog lokalnog repozitorijuma zove **remote/origin/master**. A nove grane u lokalnom repozitorijumu nakon kloniranja su:
      - **origin/master** – kopija udaljenog repozitorijuma u trenutku kloniranja
      - **master** – grana na kojoj nastavljamo rad i na koju pravimo nove commit-e

# Git - naredbe

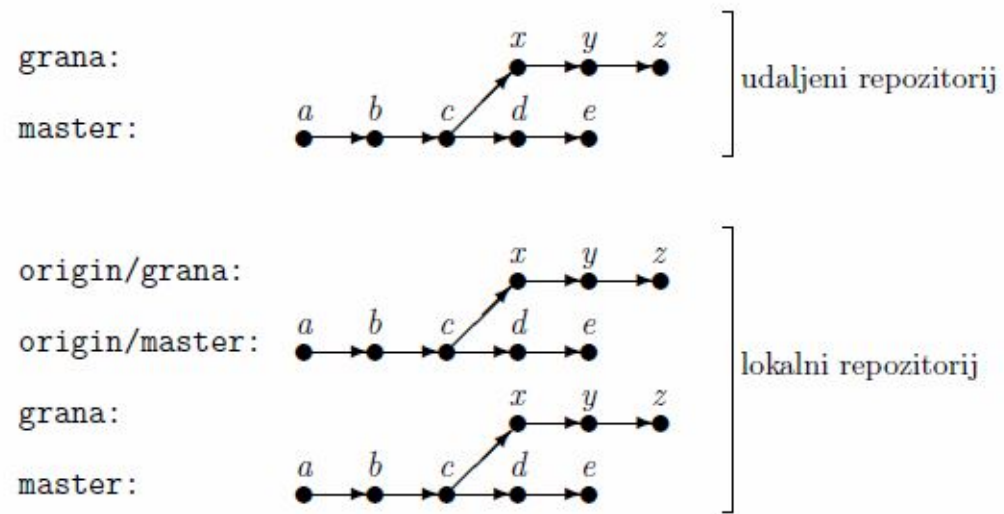
- Kloniranje repozitorijuma sa vise grana
  - Ovo je tezi slucaj za kloniranje jer se nakon standardnog kloniranja dobija lokalna master grana koja se ne racva (kao na ovim dijagramima)



- Da bi se dobila racva I na master grani u okviru naseg lokalnog repozitorijuma, potrebno je izvršiti nekoliko naredbi:
  - Prebaciti HEAD na kraj origin/grane: ***git checkout origin/grana***
  - Napraviti novu granu na masteru pod nazivom 'grana': ***git branch grana***
  - Prbaciti glavu na kraj nove grane: ***git checkout grana***

# Git - naredbe

- Nakon cega je sve klonirano u potpunosti:



# Git - naredbe

- **Delimicno kloniranje**

- Git repozitorijum sadrzi celu istoriju projekta, I ukoliko projekat dugo traze moze biti prilicno veliki. Zato ako zelimo skinuti projekat samo da bismo pogledali njegov kod, a da nas ne zanima cela istorija, moguće je klonirati samo nekoliko poslednjih commit-a:

- ***git clone - - depth 5 - - no-hardlinks git://github.com/adresa\_repo.git***

# Git - naredbe

- **AZURIRANJE LOKALNIH FAJLOVA**

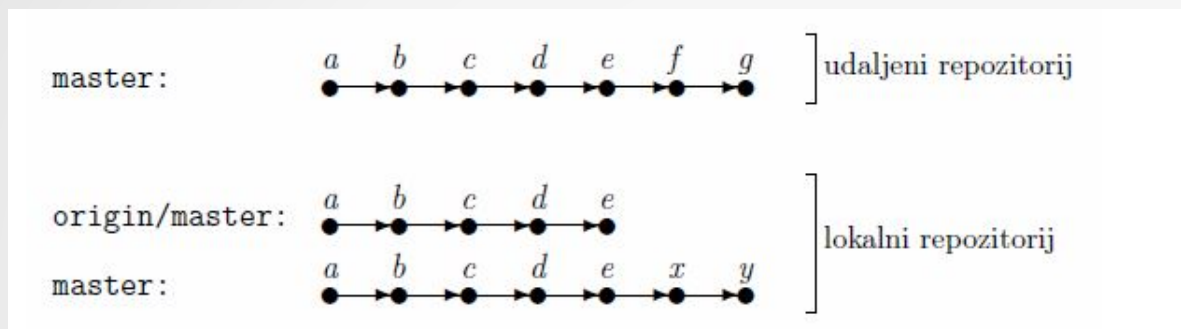
- **Fetch – azuriranje lokalne origin/master grane**

- Dok mi radimo u lokalu, paralelno nase kolege rade u svojim lokalnim repozitorijumima. Ukoliko **push**-uju svoje promene na udaljeni repozitorijum, pojavice se razlika izmedju remote origin/. Grana origin/master je kopija master grane sa udaljenog repozitorijuma I trebala bi da se azurira da bi pratila promene koje cini vlasnik na udaljenom repozitorijumu

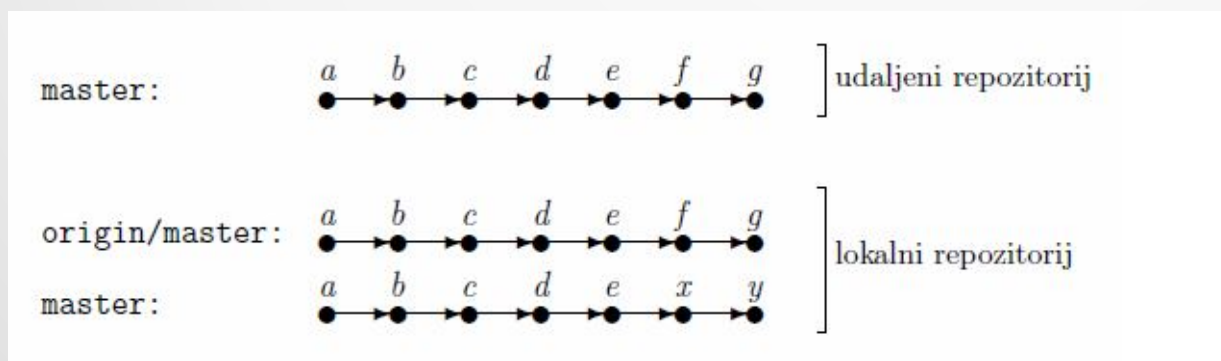
- **git fetch** – azuriranje origin/master grane

# Git - naredbe

- Pre fetch-a:

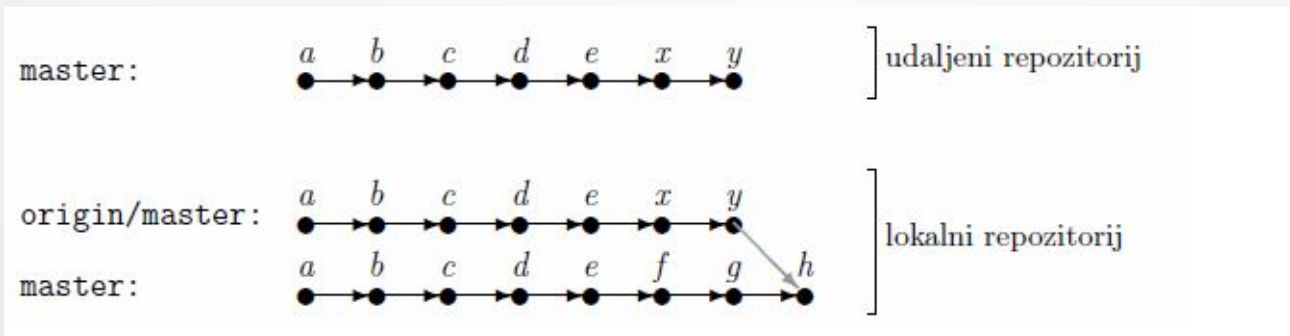


- Nakon fetch-a:



# Git - naredbe

- Merge – azuriranje lokalne master grane
  - Zatim koristimo naredbu koja ce u nasu master granu dodati sve nove commit-e sa udaljenog repozitorijuma, preko novo azurirane grane origin/master
    - *git merge origin/master*
  - Nakon cega grafik izgleda ovako:



# Git - naredbe

- Pull
  - Zbog cestog korištenja naredbi fetch I merge, uvek u istom redosledu (*git fetch* and *git merge origin master*), zamenjuje ih naredba *git pull*



# Git - naredbe

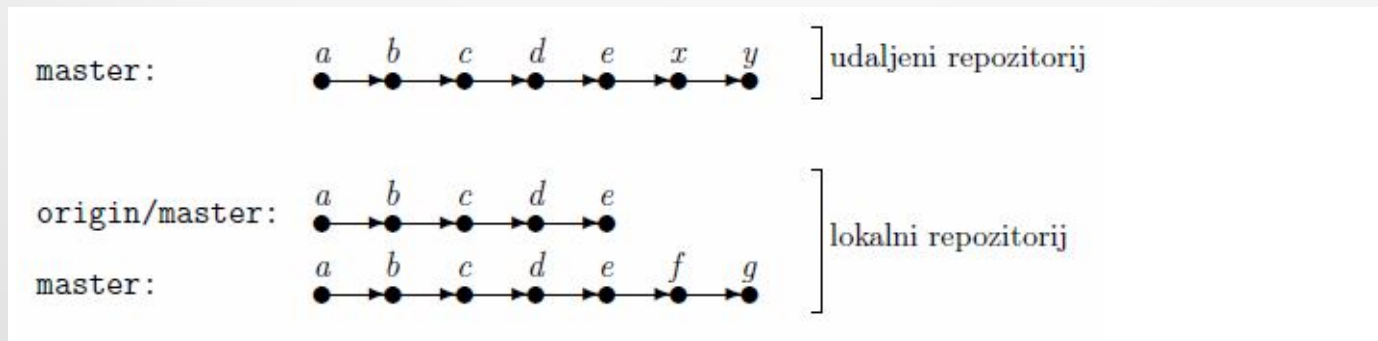
- Slanje izmena na udaljeni repozitorijum

- Obicnim commit-om se promene ne salju u udaljeni repozitorijum kao kad je slucaj sa lokalnim repozitorijumom, nego se koristi posebna naredba za to, **push**. *Praksa je da u lokalu napravimo vise commit-a, pa tek onda kad zavrismo neku celino posaljemo promene na udaljeni repozitorijum. Za slanje postoje dve opcije:*

- **push** – kada imamo ovlascenja
- **pull request** – kada nemamo ovlascenja

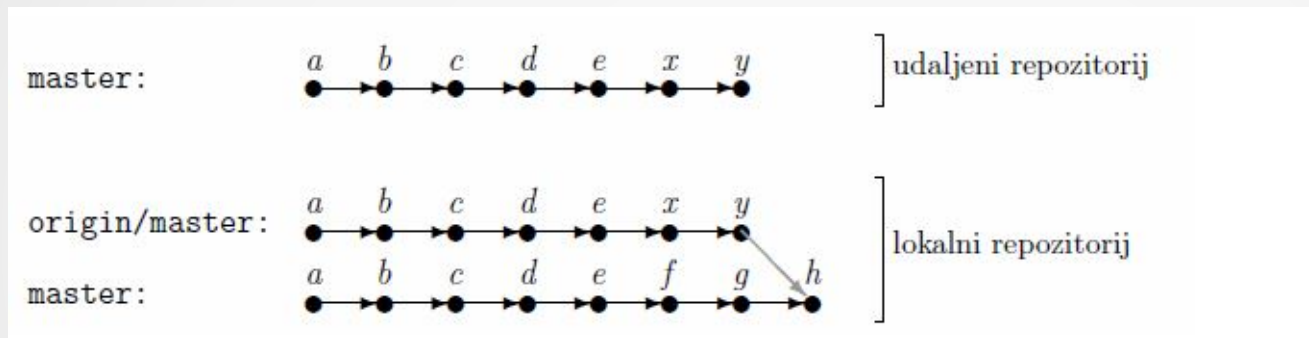
- push

- Kada na jednom projektu radi vise saradnika na razlicitim problemima, situacija tokom vremena postaje kompikovanija. U slucaju da **vlasnik** repozitorijuma radi na svom repozitorijumu i commit-ova je svoja dva commit-a, **x** i **y**, dok istovremeno **saradnik** radi u lokalu na drugom problemu i commit-ova je **f** i **g** cvorove, dijagram bi izgledao ovako:

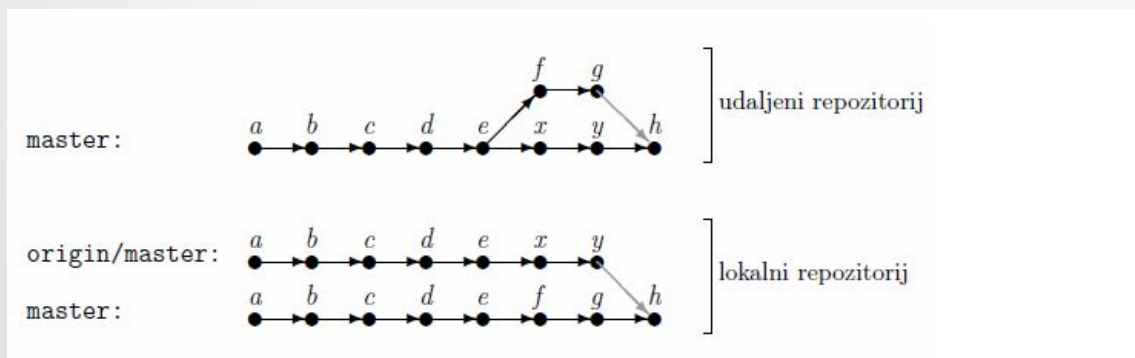


# Git - naredbe

- Ukoliko koristimo **push**, git **nece prihvatiti**, jer moramo prethodno da sredimo nas lokalni repozitorijum I azuriramo nasu lokalnu *origin/master* granu sa naredbom **git pull**



- Nakon sredjivanja lokalnog repozitorijuma, mozemo poslati nase commit-e na udaljeni repozitorijum sa **push** naredbom: **git push origin master**
- Nakon toga grafik izgleda ovako:



# Git - naredbe

- pull request
  - Pull request nije nista drugo nego kratka poruka vlasniku nekog udaljenog repozitorijuma koja sadrzi adresu naseg repozitorijuma, opis izmena koje smo napravili kao I predlog da on te izmene preuzme u svoj repozitorijum
    - *git request-pull*

# Git - naredbe

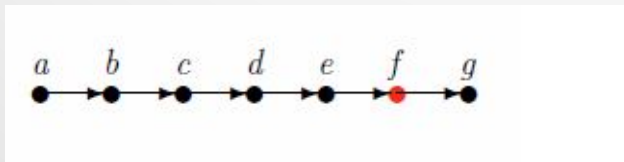
- Checkout
  - Pregled commit-a u istoj grani
    - Sa naredbom **checkout** se mozemo prebaciti na drugu granu ili na stariji commit u istoj grani. Takvo stanje se zove **detached HEAD**, stanje kada HEAD pointer ne pokazuje na poslednji commit grane
    - e.g. Ukoliko zelimo da se vratimo za 10 commit-a (koraka) unazad:
      - **git checkout HEAD~10**
    - Nakon prebacivanja na drugi commit, u radnom direktorijumu vidimo fajlove vezane za taj commit. A ukoliko pravimo izmene nad tim fajlovima, potrebno je kreirati novu granu i izvršiti commit, kako bismo sacuvali izmene. Nakon ovoga ce HEAD pokazivati na poslednji commit (commit na kraju nove grane) i repozitorijum tada vise nece biti u **detached HEAD** stanju
  - Pregled commit-a na razlicitim granama
    - Naredbu **checkout** takodje mozemo koristiti i za kretanje po granama
      - **git checkout naziv\_grane\_na\_koju\_se\_prebacujemo**
    - Kada smo promenili granu, unutar naseg radnog direktorijuma ce se nalaziti fajlovi sacuvani prilikom poslednjeg commit-a u toj grani. Kod prebacivanja sa grane na granu mogu nastati manji problemi ukoliko imamo nekomitovanih izmena. Takodje postoji nekoliko situacija u kojima nam git nece dozvoliti prebacivanje. Najcesci problem se javlja kada u dve grane imamo dve razlicite verzije istog fajla, pri cemu na jednom mestu nisu komitovane poslednje izmene. Tako da je najbolje prvi izvršiti commit, pa se onda prebacivati sa grane na granu.
    - Da bismo se ponovo vratili na poslednji commit u glavnoj grani:
      - **git checkout master**

# Git - naredbe

- Povratak poslednjeg stanja
  - Ako se dogodilo da smo izmenili fajl, a nismo izvršili commit, mozemo da vratimo verziju tog fajla pomocu: **git checkout - - file.extenzija**
  - Naredba vraca stanje datog fajla iz poslednjeg commit-a, dok HEAD pokazivac nije pomeren (pokazuje na poslednji commit)

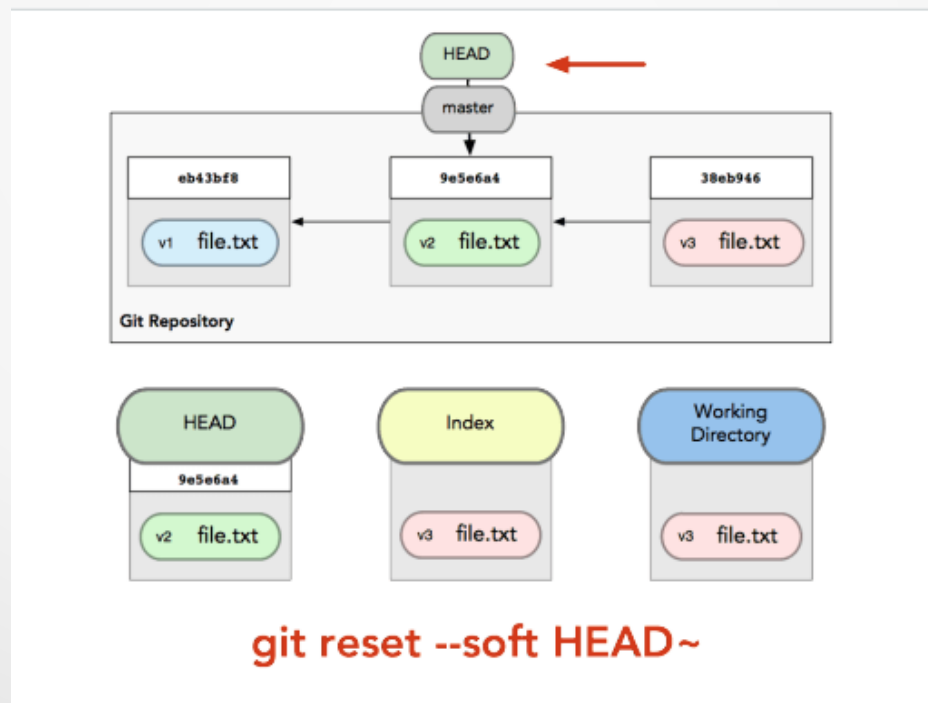
# Git - naredbe

- Vracanje stanja fajla iz istorije
  - Ukoliko jos nismo izvršili commit izmena, a zelimo da vratimo prethodnu verziju, dovoljno je koristiti **checkout** naredbu, medjutim ukoliko imamo komitovane izmene, moramo potraziti drugi nacin
  - **Revert**
    - Prilicno siguran mehanizam, sigurniji od naredbe **reset**, jer ne brise commit, nego dodaje novi commit cvor u kome su sklonjene sve izmene naprevljene u commit-u koji zelimo obrisati. **Revert** naredba pravi novi commit koji ponistava izmene snimljene u nekom prethodnom commit-u. Veoma je korisna prilikom rada vise ljudi na istom projektu.
    - Ukoliko je poslednji commit u cvoru f, I zelimo da izbacimo promene nastale u njemu, **revert** neće obrisati cvor f, nego će napraviti novi commit u cvoru, npr g, koji će biti isti kao u cvoru e.
      - **git revert HEAD**



# Git - naredbe

- Resetovanje
  - Soft reset
    - Ukoliko zelimo da stanje nasih fajlova u radnom direktorijumu ostane nepromenjeno, ali samo da 'zaboravimo' sve commit-e od tog trenutka
      - ***git reset --soft HEAD~1***

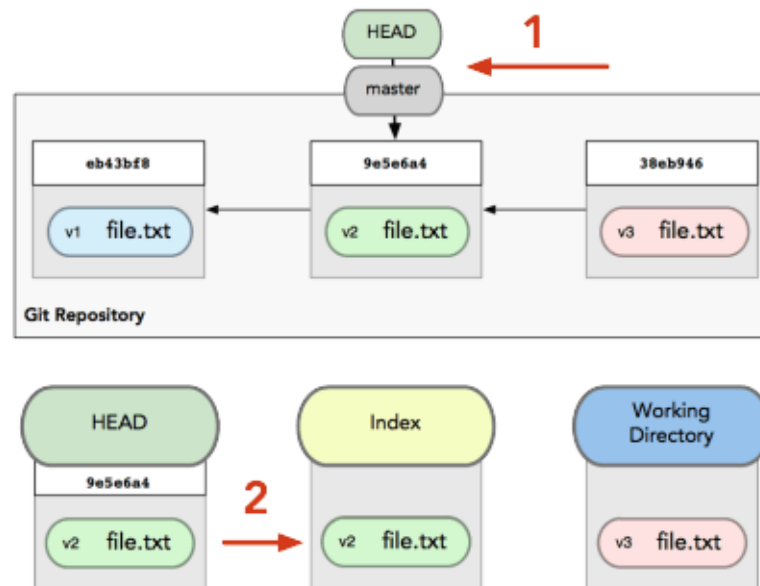


# Git - naredbe

- Mixed reset

- Vraca HEAD na staru verziju, ali takodje vraca fajl u pripremni prostor (index) bez promene fajla u radnom direktorijumu

- *git reset --mixed HEAD~1*



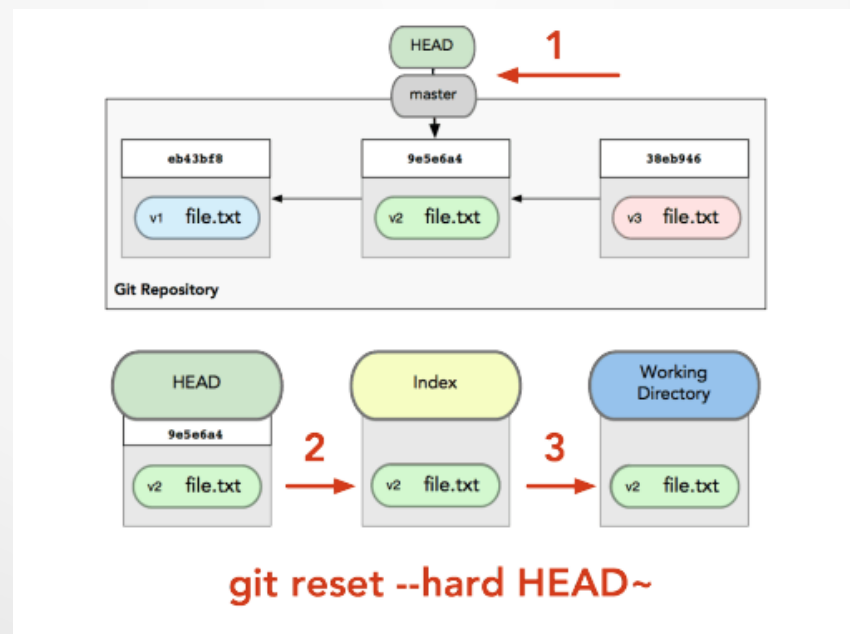
*git reset [--mixed] HEAD~*



# Git - naredbe

- Hard reset

- Ukoliko zelimo da vratimo stanje nasih fajlova u radnom direktorijumu, od par commit-a unazad, ali I da se 'zaborave' svi commit-i koji su uradjeni od tog trenutka, radimo:
  - ***git reset --hard HEAD~1***
- I npr ukoliko smo greskom poslali fajlove u pripremni prostor (index), mozemo ukloniti sve fajlove iz indexa (staging area) naredbom:
  - ***git reset HEAD .***



# Git - naredbe

- Rad sa granama
  - Popis grana
    - Ukoliko zelimo spisak svih grana koje se koriste u repozitorijumu
      - *git branch*
    - Prilikom izlistavanja, sa zvezdicom je obelezena trenutna grana
  - Pravljenje nove grane
    - *git branch naziv\_nove\_grane*
    - Kada se napravi nova grana, unutar nema cvorova jer nismo uradili niti jedan commit, a mi se i dalje nalazimo u glavnoj (master) grani, sto mozemo proveriti *git branch* naredbom, i da bismo u novoj grani napravili prvi cvor, neophodno je da izvorsimo *commit*
  - Prebacivanje sa grane na granu
    - *git checkout grana\_na\_koju\_zelimo\_da\_se\_prebacimo*
    - Kada smo se prebacili na zelenu granu, unutar naseg radnog direktorijuma ce se nalaziti fajlovi sacuvani pri poslednjem commit-u te grane
  - Brisanje grane
    - *git branch -D naziv\_grane*

# Git - naredbe

- Preuzimanje fajla iz druge grane

- Neretko se desava da zelimo preuzeti samo jednu ili više datoteka iz druge grane, bez prelaska na drugu granu.

- *git checkout naziv\_grane - - file.ext*

- Moguce je preuzeti i više fajlova, istovremeno

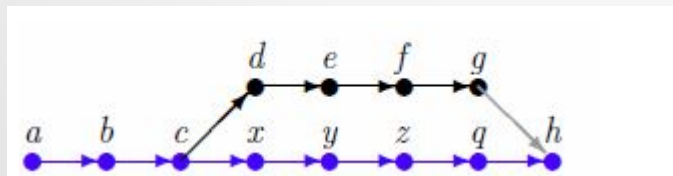
- *git checkout naziv\_grane - - file1.ext file2.ext*

- Spajanje grana

- Za preuzimanje izmena iz jedne grane u drugu, potrebno je biti u grani u kojoj zelimo dodati izmene

- *git merge naziv\_grane\_iz\_koje\_preuzimamo*

- Iako se **merge branch** prevodi kao 'spajanje grana', to nije ispravan smisao ove naredbe, jer standardno značenje 'spajanje grane' bi napravilo samo jednu granu, dok ovde grane nastavljaju svoj život odvojeno, jedino što se od tog trenutka **sve izmene nastale u jednoj grani preuzimaju u drugu granu.**



- Nakon izvršenog pripajanja nije potrebno raditi commit jer je sama naredba napravila novi čvor **h**. Za razliku od ostalih cvorova (commit-a) samo ovaj ima dva 'roditelja' tj. dve strelice vode do njega.

# Git - naredbe

- Uz sledece slucajeve cemo bolje razumeti kako funkcionise ova naredba

| Slučaj  | Rezultat naredbe   |
|---|--|
| Fajl je izmenjen u pripojenoj grani a u glavnoj nije                                    | Dodaće se izmene u fajlu   |
| Fajl je dodat u pripojenoj grani  | Biće dodat fajl  |
| Fajl je obrisani u pripojenoj grani a u glavnoj nije                                    | Fajl će biti obrisani  |
| U pripojenoj grani smo izmenili i preimenovali fajl, a u glavnoj smo samo izmenili fajl | Ako izmene na kodu nisu bile konfliktne, onda će se u glavnoj fajl <i>preimenovati</i> i <i>sadržati izmene iz obe grane</i> . |
| U pripojenoj grani smo obrisali fajl, a u glavnoj smo ga samo izmenili.                 | KONFLIKT   |

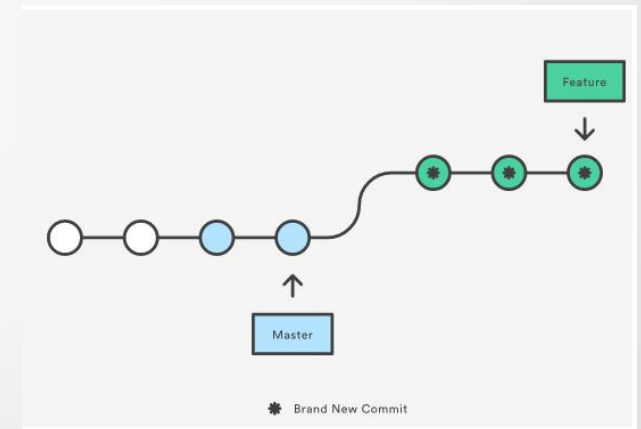
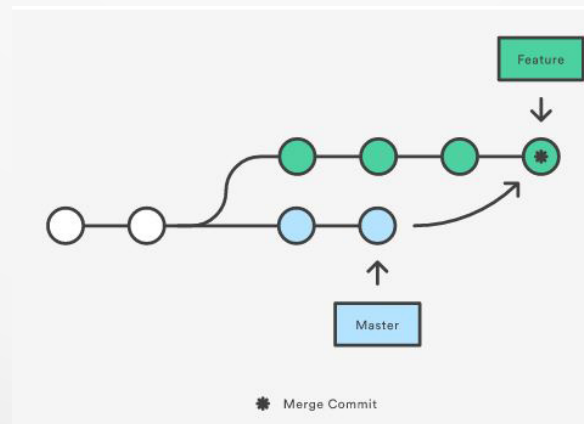
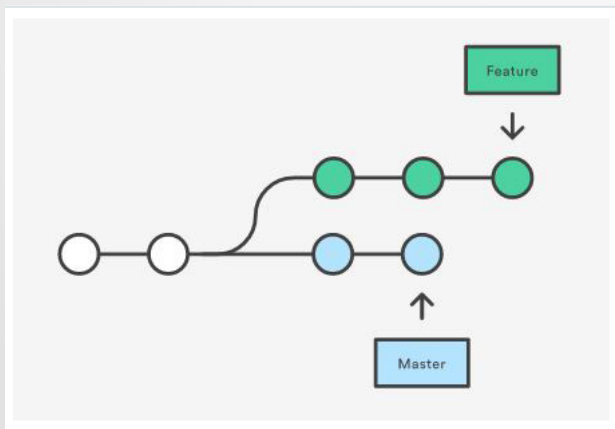
# Git - naredbe

- **Konflikti pri spajanju grana**
  - U slucaju konflikta, problematicni fajl se nece komitovati, vec je ostavljeno da korisnik sam edituje fajl I izabere verziju koja mu odgovara. Postoje programi (**merge tool**) za vizuelni prikaz konflikta I lakse resavanje istog
  - Neki od alata: **araxis, bc3, codecompare, deltawalker, diffuse, ecmerge, emerge, gvimdiff, gvimdiff2, kdiff3, meld, opendiff, p4merge, tkdiff, tortoisemerge, vimdiff, vimdiff2, xxdiff, ...** Mozemo ih aktivirati nakon instalacije naredbom: ***git config - - global merge.tool /path\_to\_program***
- **Cherry-pick**
  - Specijalna vrsta spajanja dve grane, kada zelimo da prebacimo samo jedan commit iz jedne grane u drugu, tada ne zelimo da koristimo merge jer bi prebacio sve izmene koje smo napravili u celoj grani.
  - Postupak: za prebacivanje poslednjeg commit-a sa sporedne grane u master granu
    - ***git log sporedna\_grana*** – proucimo istoriju sporedne grane
    - ***Izaberemo jedinstveni broj commit-a***
    - ***git cherry-pick jedinstveni\_broj\_commit-a*** – naredba koja aktivira cherry-pick

# Git - naredbe

- **Rebase**

- Naredba je slicna naredbi **merge**, ali ih ne spaja u jedan novi krajnji cvor (kao merge commit), nego dodaje celu istoriju svih commit-a
- Najveca prednost je jasnija istorija projekta ciji dijagram je linearan kao i eliminisanje dodatnih cvorova koji se pravi pri koriscenju merge naredbe. Najcesce se koristi za azuriranje origin/master grane koja je lokalna kopija master grane udaljenog repozitorijuma
- Primer:
  - Prikazano je stanje projekta, gde pored master grane postoji i sporedna grana, a u okviru nje par commit cvorova. Treba da spojimo podatke iz master grane (plavi krugovi) u sporednu granu.



# Git - naredbe

- Pretraga
  - Pretraga komentara
    - *git log - - grep = "rec\_za\_pretragu"*
  - Pretraga reci (string-a) unutar fajlova
    - *git log -S rec\_za\_pretragu*
    - *git log -S "rec sa razmacima"*
    - Pretraga nam nalazi commit u kome se nalazi trazena rec, I to u obliku **76cf802d23834bc74473370ca81993c5b07c2e35**. Naziv commit-a cine prvih pet znakova **76cf8**, I mozemo commit pregledati naredbom: *gitk 76cf8*
- Ciscenje radnog direktorijuma
  - Kada zelimo obrisati samo privremene fajlove koje su rezultat kompajliranja ili privremene direktorije koje nisu trenutne verzije repozitorijuma, git ih moze sam pronaci. Za ovu opciju koristimo naredbu **clean**
    - *git clean -n* – spisak stvari koje git nudi da obrisemo
    - *git clean -f* – brisanje svega sa spiska
    - *git clean -f - - naziv\_direktorijuma* – brisanje samo odredjenog direktorijuma
    - *git clean -x* – brisanje cak I svih fajlova iz .gitignore

# GitHub Fork

- Naredba **fork** ne postoji u okviru git-a, vec je vezana za servis GitHub. Omogucava nam da neki projekat drugog korisnika GitHub-a, kopiramo u nas GitHub, tako da nasa kopija bude 'svjesna' da postoji roditeljski repozitorijum.
- Nakon forkovanja projekta na nas GitHub, potrebno je da se napravi lokalna verzija repozitorijuma kloniranjem, I taj lokalni repozitorijum je u vezi sa oba repozitorijuma na GitHub-u.
- Iz naseg lokalnog repozitorijuma dajemo naredbu koja lokalnom repozitorijumu daje do znanja da postoji pocetni repozitorijum na osnovu koga je forkovan repozitorijum na nas GitHub.
  - ***git remote add upstream https://github.com/username/repo.git***
- Gledano sa lokala, roditeljski repozitorijum kod standardnog kloniranja se zove 'origin', a originalni GitHub repozitorijum koji je forkovan se naziva 'upstream'
- Veza izmedju lokalnog repozitorijuma i upstream repozitorijuma je bitna jer preko nje azuriramo lokalnu granu upstream/master koja je kopija upstream grane sa naredbom:
  - ***git fetch upstream***
  - ***git merge upstream/master*** – spajanje promena
  - ***git rebase upstream/master*** – ili ako upstream ima jednostavan set commit
  - Po zavrsetku rada na fajlovima u lokalu, mozemo standardno poslati sve promene na nas udaljeni repozitorijum **origin** sa naredbom **push**, nakon čega možemo na više načina da pošaljemo na **upstream pull request**.