

Integracija softverskih tehnologija

Osnovne oblasti:

- ▶ SVC - git

- ▶ Formati

- ▶ JSON, XML
 - XSD, DTD
 - XPath
 - (XSLT...)

- ▶ Node.Js

- ▶ Osnove, moduli
- ▶ Npm
- ▶ Express, servisi

- ▶ WS servisi

- ▶ SOAP
- ▶ REST

Alati koji će biti korišćeni

- ▶ Git for Windows
- ▶ XML editori
 - ▶ VisualStudio poslednja licencirana verzija
 - ▶ Pišemo (programiramo) XSLT transformacije
- ▶ Node.Js - Visual Studio Code
- ▶ Web Servrasi
 - ▶ VisualStudio (+ NetBeans)
 - ▶ (WS servisi i klijenti za te servise)

Poželjno predznanje

- ▶ Poznavanje rada na računaru
- ▶ Poznavanje osnovnih pojmova pri radu u Internet okruženju
- ▶ Osnove HTML

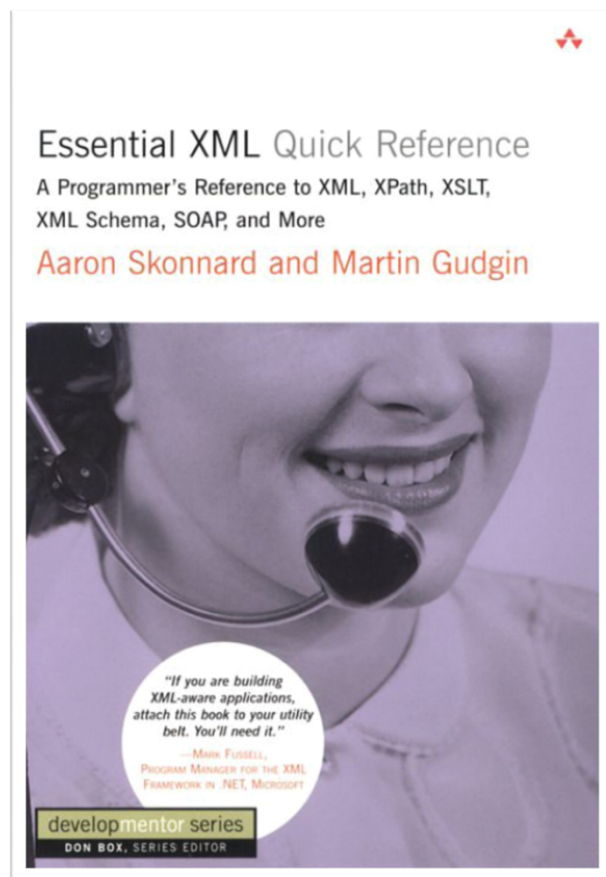
- ▶ Osnove JavaScript-a, Jave ili C#

- ▶ *Bez obzira na stečeno predznanje, biće prezentovano gradivo potrebno za kurs u celini!!!*

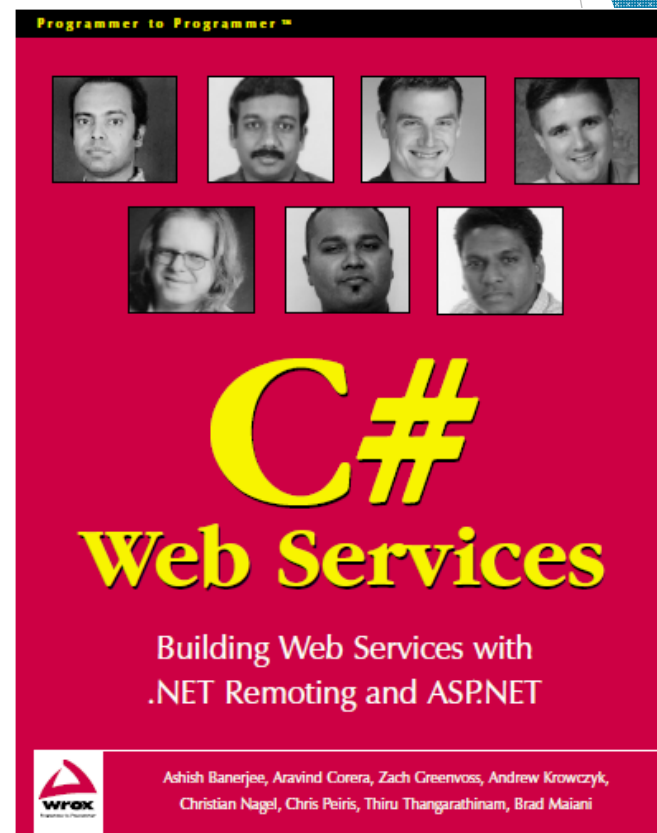
Literatura

- ▶ Materijali koji će biti postavljeni na stranici predmeta biće dovoljni za polaganje ispita sa najvećom ocenom...uz preduslove (predavanja, vezbe, ...)
- ▶ Materijali će biti u vidu pdf skripti/prezentacija, a sadržaćće i dodatne fajlove koji će pratiti odgovarajuće primere.
- ▶ Poželjno je koristiti i dodatnu literaturu:

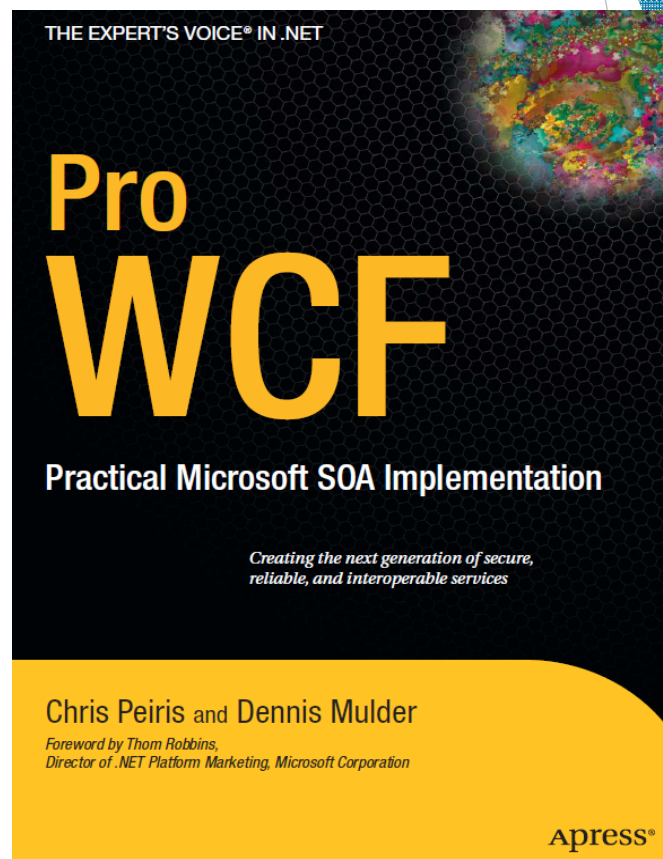
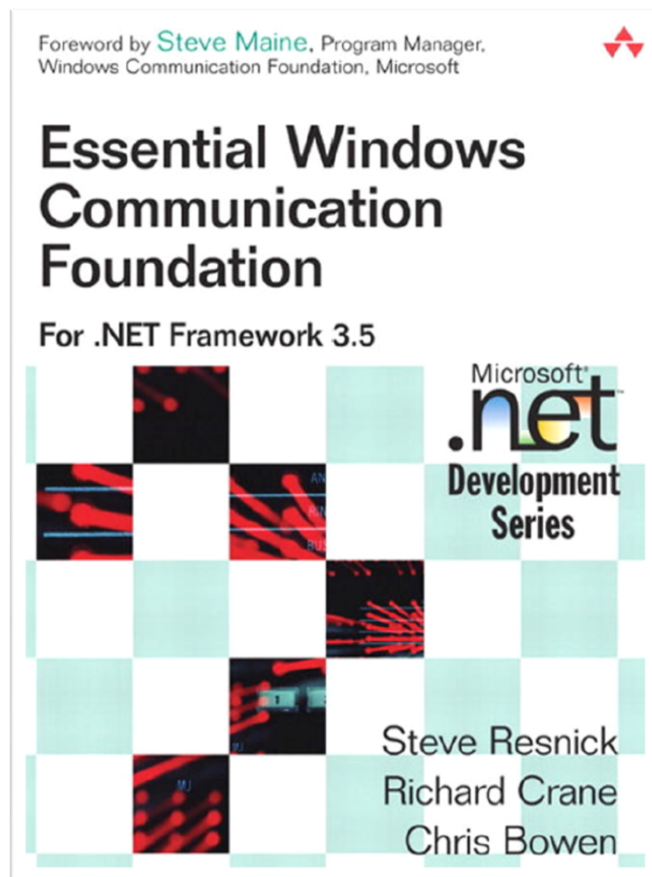
Dodatna literatura: I deo



Dodatna literatura: II deo



Dodatna literatura: III deo



Polaganje ispita

- ▶ Opcija 1. Ispit u celosti
- ▶ Opcija 2. Ispit iz delova.
- ▶ Parcijalno mogu polagati samo studenti koji su redovni na predavanjima i vežbama (na više od 80%)
 - ▶ Zadaci
 - ▶ Projekat
- ▶ Vežbe se evidentiraju
 - ▶ Vežbe su obavezne u 80%

Organizacija kursa

- ▶ Predavanja
 - ▶ 3 čas nedeljno
- ▶ Vežbe
 - ▶ 2 časa nedeljno
 - ▶ Vežbe su laboratorijske tj. za računarima

Organizatori kursa

► Predavanja

- Prof. dr Zoran Ćirović
- Kabinet: 514
- Konsultacije se definišu terminom koji je na Web stranicam Škole

► Vežbe

- Nemanja Cvijan

Pitanja?

Sistemi za verzioniranje

Osnove verzioniranja - 1

- ▶ Sistem za kontrolu verzija - engl. Version Control System (VCS)
- ▶ Upravljanje promenama dokumenata, koda, velikih sajtova i drugih kolekcija informacija naziva se **verzioniranje**. A sistemi koji omogućavaju rad sa verzijama **VCS sistemi**.
- ▶ Promene tj. nova verzija se obično identifikuje 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 pridružena odgovarajućim vremenskim pečatom - engl. timestamp, kao i osobom koja izvodi izmene.
- ▶ **Linus Torvalds** 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.

Osnove verzioniranja - 2

- ▶ VCS omogućavaju da se izmene mogu uporediti, sačuvati ili vratiti na prethodne, ali i spajati.
- ▶ Uloga VCS je višestruka. To su:
 1. Čuvanje istorije,
 2. Rad u timu,
 3. Grananje,
 4. Rad sa spoljnim učesnicima,
 5. Skaliranje.
- ▶ Pogledajmo redom svaku od ovih uloga.

Čuvanje istorije

- ▶ Čuvanje istorije podrazumeva da se promene na dokumentima čuvaju se od samog početka. Drugo, jako važno je da je u svakom trenutku tj. uvek je moguće uraditi povratak 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.

Rad u timu

- ▶ Verzioniranje pomaže pri:
 - Deljenju kolekcije fajlova sa ostalim učesnicima tj. članovima tima. Mogu se definisati verzije za pojedine timove, takođe moguće je izostaviti fajlove koje ne treba menjati i slično.
 - Spajanje promena koje su nastale od drugih učesnika ili timova.
 - Osiguravanje da se ništa ne može slučajno izgubiti ili preklopiti.

Grananje

- ▶ Grananje omogućava da postoji više verzija istog programa istovremeno. To se ostvaruje preko grananja. Na primer, neke grane pri razvoju jednog programa su:
 - 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)

Rad sa spoljnjim učesnicima u razvoju koda

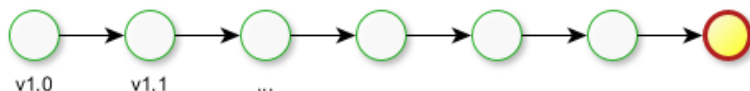
- ▶ Alati za verzioniranje pomažu u radu tj. razvoju koda u kome učestvuju spoljni sadradnici tzv. saradnici trećih strana - engl. *Third-party contributors*. Ovi alati omogućavaju:
- ▶ uvid u ono što se događa u razvoju tj. u projektu,
- ▶ pomaže im da urade i integrišu izmene (zacrpe),
- ▶ grananje razvoja softvera i njegovo spajanje u glavnu liniju.

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
- ▶ Dakle, primenom verzioniranja omogućava se razvoj i skaliranje projekata čak i u slučaju tako velikog broja učesnika kao što je slučaj sa Linux razvojem.

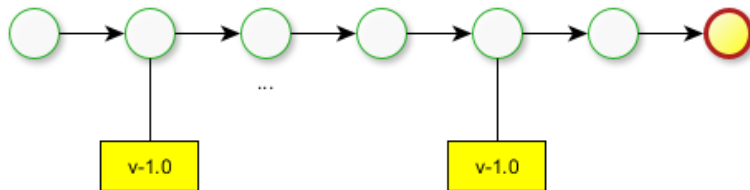
Grafički prikaz i tipični slučajevi

- Verzioniranje se često prikazuje grafički. Razlog je što grafički prikaz daje jasno postupak promene tj. prelaz iz verzije u verzije, grananje i spajanje. Grane su tipično označene strelicama koje povezuju stanja koja su prikazana grafički kao kružići sa pripadajućom labelom koja označava naziv verzije.

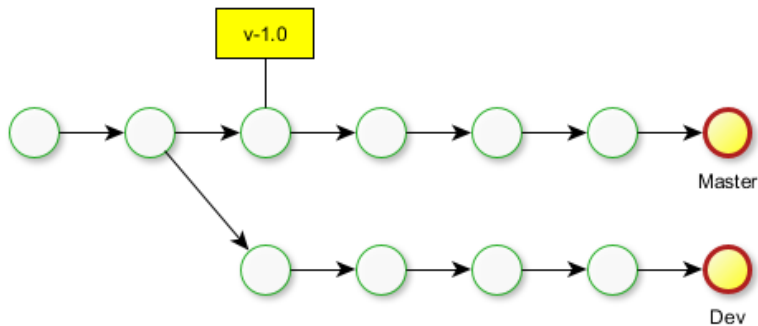


- Tekuća verzija na repozitorijumu se posebno grafički prikazuje i obično se označava kao „HEAD“.

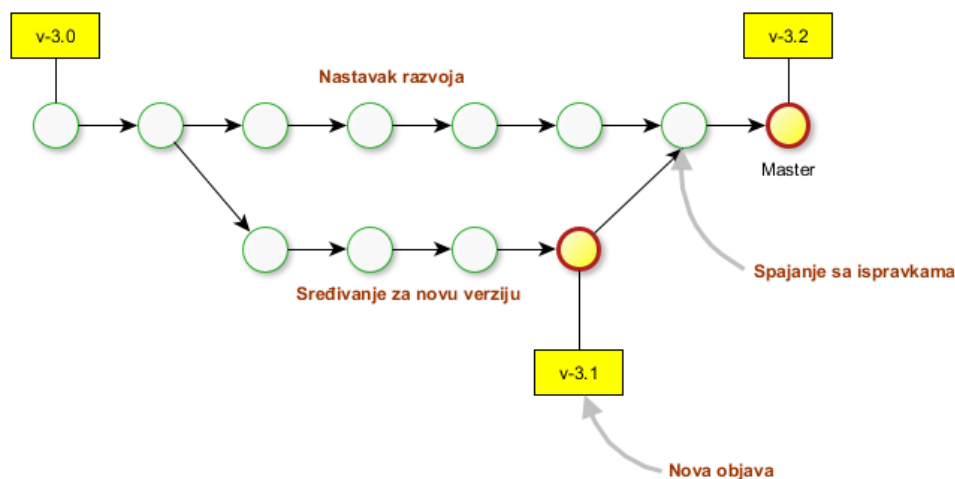
- ▶ Svaka verzija nosi prateći opis. Međutim, postoje posebne verzije koje su od posebnog značaja i koje se posebno označavaju. Obično su to verzije softvera koje se objavljuju (engl. *Release version*).



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



- ▶ 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.



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

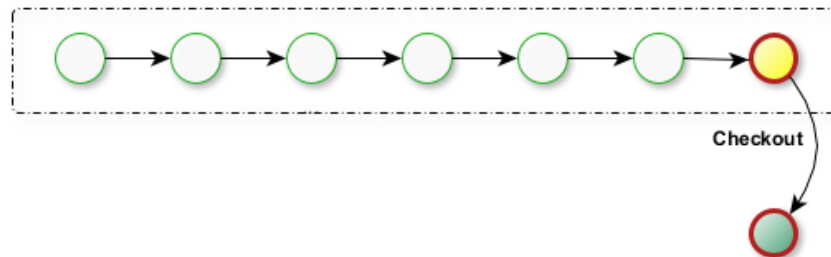
Organizacija verzioniranja

- ▶ U osnovi postoje dve vrste organizacije rada više korisnika 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 konfliktima. Git verzioniranje podrazumeva decentralizovan sistem sa spajanjem. Git omogućava pristup bilo kojoj grani tokom razvoja kao i praćenje svih izmena u svakoj fazi razvoja.

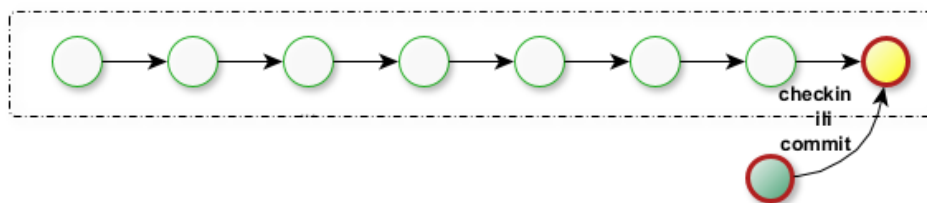
- ▶ Svaki repozitorijum mora imati na raspolaganju odgovarajući prostor za skladištenje kako bi bilo moguće skladištiti izmene u svakom fajlu projekta.
- ▶ Većina alata za verzioniranje koristi delta kompresiju kako bi optimizovao prostor za skladištenje, osim Git-a koji koristi tzv. objektno pakovanje.
- ▶ Svaki repozitorijum se identifikuje 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**.



- ▶ Korak 2. Sledi rad i izmene na preuzetim kopijama fajlova, tj. kreiranje verzija koje se oslanjaju na ovu preuzetu. Istovremeno radi se i na glavnoj grani.
- ▶ Korak 3. Kada je radna kopija fajlova spremna za postavljanje tj. za evidentiranje nove verzije izvodi se komanda **commit**.



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

Sadržaj repozitorijuma

- ▶ 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

Kratka istorija git-a

- ▶ 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 objašnjenja i sintakse. Bez navedene komande dobijate listu mogućih.

\$ git help

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--
info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-
objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--
namespace=<name>]
        <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone	Clone a repository into a new directory
init	Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add	Add file contents to the index
mv	Move or rename a file, a directory, or a symlink
reset	Reset current HEAD to the specified state
rm	Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect	Use binary search to find the commit that introduced a bug
--------	--

grep	Print lines matching a pattern
log	Show commit logs
show	Show various types of objects
status	Show the working tree status

grow, mark and tweak your common history

branch	List, create, or delete branches
checkout	Switch branches or restore working tree files
commit	Record changes to the repository
diff	Show changes between commits, commit and working tree, etc
merge	Join two or more development histories together
rebase	Reapply commits on top of another base tip

tag	Create, list, delete or verify a tag object signed with GPG
-----	---

collaborate (see also: git help workflows)

fetch	Download objects and refs from another repository
pull	Fetch from and integrate with another repository or a local branch
push	Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some concept guides. See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.

Ukoliko niste sigurni u sintasku ili opcije, svakako je možete iskoristiti.
Na primer: **git help status**, **git help commit**

Konfigurisanje

- ▶ Git poseduje mogućnost za podešavanje tj. konfigurisanje. Konfigurisanje se vrši postavljanjem određenih vrednosti za specifične parametre Git-a. Osnovna komanda je:
- ▶ **git config** [*parametri*]
- ▶ Postoje 3 različita nivoa konfiguracionih parametara. To su:
 1. sistemski,
 2. globalni,
 3. lokalni.
- ▶ Vrednosti parametara jednog nivoa preklapa vrednosti iz prthodnog nivoa.

Sistemiški/globalni/lokalni nivo

- ▶ Sistemiški parametri 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 koristi 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 nivoa 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

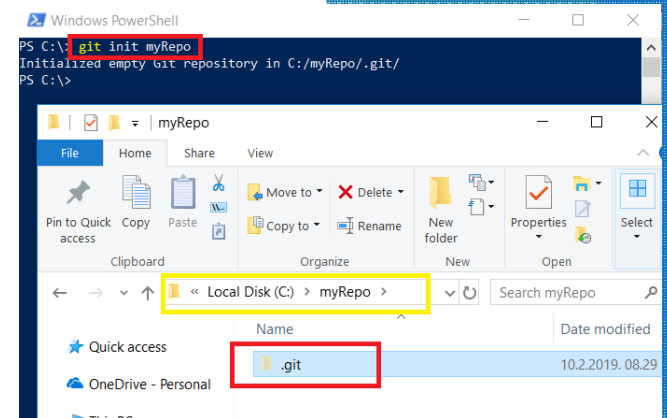
- ▶ U realnim slučajevima obično se radi verzioniranje velikog broja fajlova u okviru više projekata. Pri tome veliki broj fajlova nastaje primenom kompajlera odnosno korišćenjem određenog IDE okruženja (.class, .pyc, .o). Takvi fajlovi se obično ne koriste pri formiranju nove Git verzije jer se kreiraju automatizovano. Zato je važno da objasnimo mehanizam koji Git koristi u ovom slučaju.
- ▶ Da bi se odvajanje fajlova, koji se postavljaju na repozitorijum od onih drugih, izvršilo automatizovano, a ne ručno, Git predviđa upotrebu fajla **.gitignore** čiji sadržaj predstavljaju ekstenzije fajlova koje Git treba da ignoriše pri kreiranju verzija.

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

- ▶ #Visual Studio files
- ▶ *. [Oo]bj
- ▶ *.user
- ▶ *.aps
- ▶ *.pch
- ▶ *.vspssc
- ▶ *.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:



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)

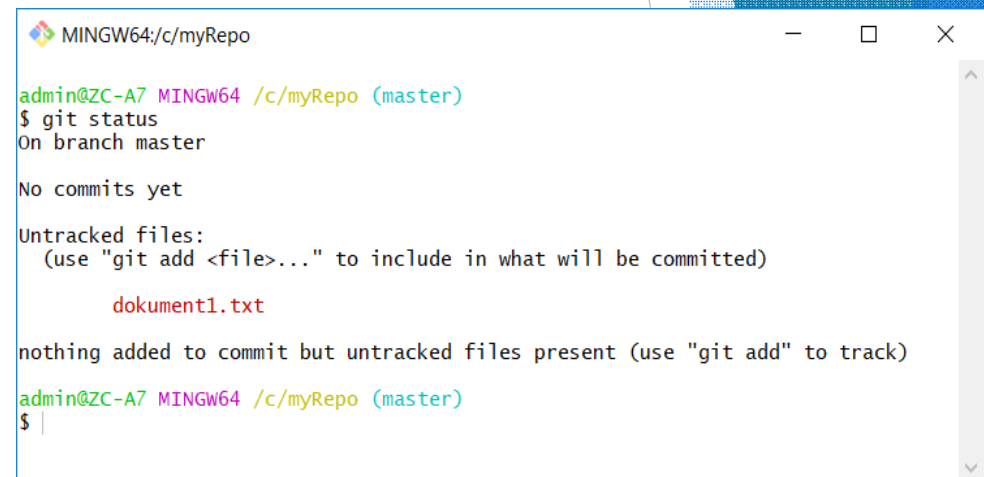
```
    New Text Document.txt
    dokument5.txt
```

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*

A screenshot of a terminal window titled 'MINGW64:/c/myRepo'. The prompt is 'admin@ZC-A7 MINGW64 /c/myRepo (master)'. The user enters '\$ git status'. The output shows 'On branch master', 'No commits yet', and 'Untracked files: (use "git add <file>..." to include in what will be committed)' followed by 'dokument1.txt' in red. A final line says 'nothing added to commit but untracked files present (use "git add" to track)'. The prompt returns to '\$ |' with the user's cursor.

```
MINGW64:/c/myRepo
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git status
On branch master

No commits yet

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

        dokument1.txt

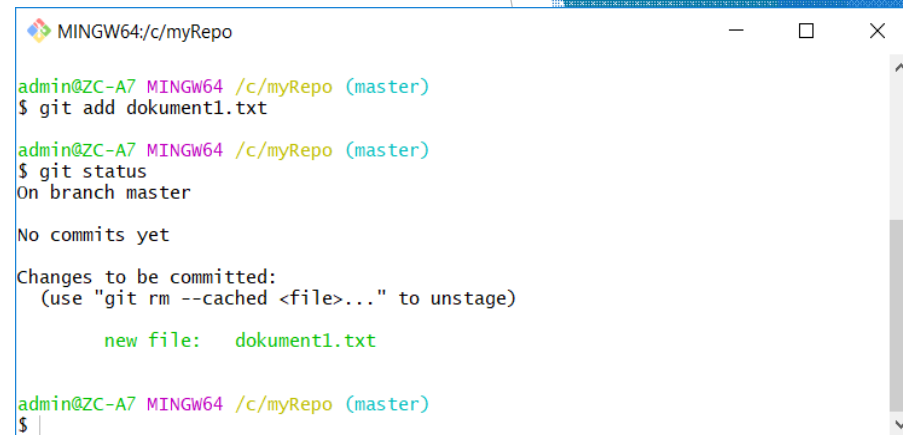
nothing added to commit but untracked files present (use "git add" to track)
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ |
```

- ▶ Skup fajlova koji Git prati naziva se **scena** ili indeks tj. a zbog dokumentacije često se koriste i engleski termini **index** ili **staging area**. Obratite pažnju da se poruke odnose na granu **master**. Ovo je osnovna tj. glavna grana repozitorijuma.

- ▶ Neke varijante ove komande su:

- ▶ **git add**

`--all (-A)` // dodavanje svih fajlova u folderu
`--force (-f)` // dodavanje fajlova koji su definisani za ignorisanje
`--dry-run (-n)` // pokazuje sve poruke ali bez dodavanja fajlova



```
MINGW64:/c/myRepo
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git add dokument1.txt
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git status
On branch master

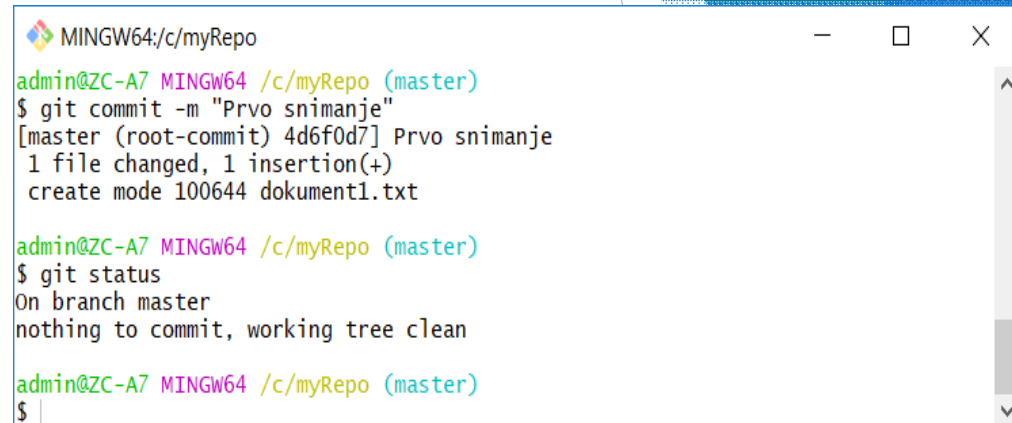
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   dokument1.txt
admin@ZC-A7 MINGW64 /c/myRepo (master)
$
```

Snimanje promena

- ▶ Snimanje promena u repozitorijum, tj. formiranje nove verzije, obavlja se komandom **commit** na sledeći način:
- ▶ **git commit -m "poruka"**
- ▶ Uz snimanje obavezno se navodi poruka koja se kasnije koristi u prikazu istorije. Poruka je obeležje verzije i treba je birati tako da daje neki podatak. Novi status bio bi:
- ▶ Opcija **-e --edit** otvara fajl za editovanje poruke.

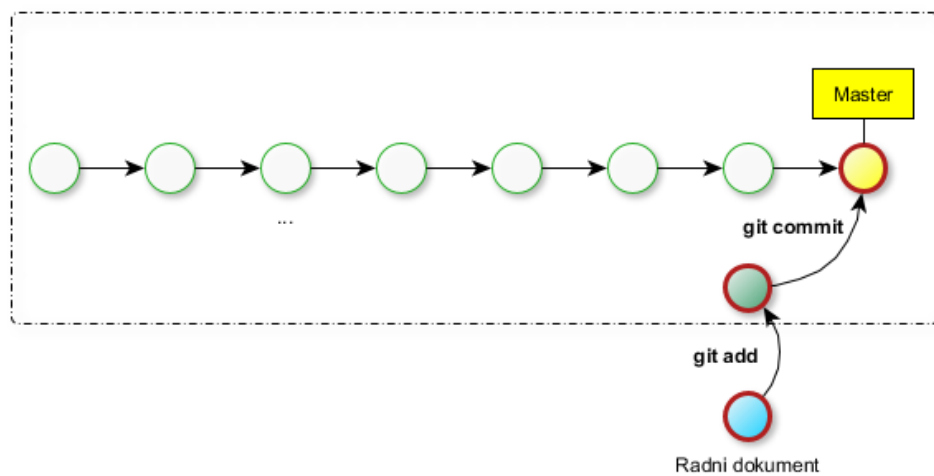


```
MINGW64:/c/myRepo
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git commit -m "Prvo snimanje"
[master (root-commit) 4d6f0d7] Prvo snimanje
1 file changed, 1 insertion(+)
create mode 100644 dokument1.txt

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)
$
```

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



- ▶ 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 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.

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

        modified:   dokument1.txt

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

        dokument2.txt

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

        modified:   dokument1.txt
        new file:   dokument2.txt

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

- Zatim se može izvesti novo snimanje.

```
MINGW64:/c/myRepo
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git commit -m "Novi_dokument"
[master 6d069de] Novi_dokument
2 files changed, 3 insertions(+), 1 deletion(-)
create mode 100644 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)
$
```

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.
- ▶ 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.

```
MINGW64:/c/myRepo
$ git status
On branch master
nothing to commit, working tree clean

admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git rm dokument2.txt
rm 'dokument2.txt'

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

        deleted:    dokument2.txt

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


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 obavlja rekurzivno na svim fajlovima koji nisu uključeni u reviziju počev od tekućeg foldera.

- ▶ 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 biti 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 brisanje 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_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 komandom prikazuje se razlike radnih kopija svih indeksiranih fajlova. Razlika se odnosi na promenjene fajlove u odnosu na poslednje snimanje (commit).

```
MINGW64:/c/myRepo
$ git status
On branch master
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:   dokument3.txt

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

        dokument4.txt

no changes added to commit (use "git add" and/or "git commit -a")
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git diff
diff --git a/dokument3.txt b/dokument3.txt
index 08ff392..a5c886d 100644
--- a/dokument3.txt
+++ b/dokument3.txt
@@ -1,2 @@
-Tekst dokumenta3
\ No newline at end of file
+Tekst dokumenta3
+Dodatak
\ No newline at end of file
admin@ZC-A7 MINGW64 /c/myRepo (master)
$
```

- ▶ 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`**

MINGW64:/c/myRepo

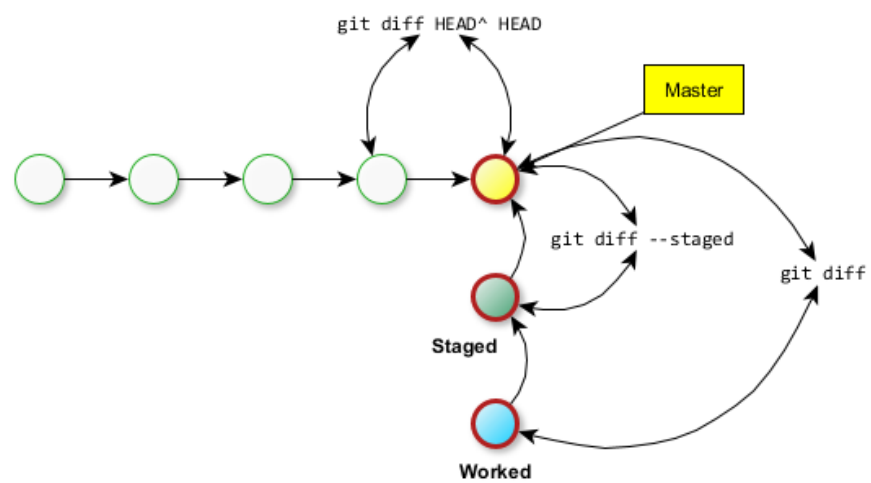
```
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git diff a11fc7:dokument2.txt 284e05:dokument2.txt
diff --git a/dokument2.txt b/dokument2.txt
index 770f438..ae0afc0 100644
--- a/dokument2.txt
+++ b/dokument2.txt
@@ -1,2 @@
-Ovo je radni tekst ali u dokumentu2.
\ No newline at end of file
+Ovo je radni tekst ali u dokumentu2.
+Izmena u dokumentu2
\ No newline at end of file

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

- ▶ 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:

```
MINGW64:/c/myRepo
admin@ZC-A7 MINGW64 /c/myRepo (master)
$ git diff HEAD^:dokument2.txt HEAD:dokument2.txt
diff --git a/dokument2.txt b/dokument2.txt
index 770f438..ae0afc0 100644
--- a/dokument2.txt
+++ b/dokument2.txt
@@ -1,2 @@
-Ovo je radni tekst ali u dokumentu2.
\ No newline at end of file
+Ovo je radni tekst ali u dokumentu2.
+Izmena u dokumentu2
\ No newline at end of file
admin@ZC-A7 MINGW64 /c/myRepo (master)
$
```


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



Referenca HEAD

- ▶ U radu sa Git-om često se barata sa terminimu HEAD.
- ▶ **HEAD:** Ukazuje na poslednje tj. važeće snimanje na repozitorijum. U standardnim 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

