Git za matematike

Abstract

Git je program, ki omogoča vodenje zgodovine različic datotek v nekem direktoriju. V glavnem se uporablja za upravljanje z izvorno kodo pri razvoju računalniških programov. Mnogi med nami pa ga uporabljajo tudi pri pisanju besedil v *LaTeX*-u. Poleg tega, da Git hrani zgodovino sprememb, tudi omogoča da več ljudi hkrati sodeluje pri urejanju istih datotek. Ogledali si bomo, kako Git deluje. Opisali bomo, kako Git uporabi *zgoščevalne funkcije*, *Merklejeva drevesa* in *usmerjene aciklične grafe*, da shrani zgodovino različic in omogoči hkratno urejanje vsebine. Matematični model, ki ga Git uporablja je v resnici zelo preprost in njegovo razumevanje nas lahko reši marsikatere zagate, ki nastane med njegovo uporabo.

1. Kaj je Git?

Git je kot **časovni stroj** za datoteke. Uporabniku omogoča, da vidi **pretekle različice** datotek, sprememinja datoteke, **brez skrbi, da bi kaj pokvaril** in datoteke **deli z drugimi**. Poleg časovnega stroja je Git **razpršeno skladišče datotek**. Omogoča, da datoteke **hkrati ureja več uporabnikov** na različnih računalnikih in kasneje spremembe **združi**.

Git hrani vsebino direktorija z datotekami in celotno zgodovino različic datotek iz preteklosti. Za vsako različico hrani Git zapis o avtorju, datumu in opis sprememb, ki so nastale v primerjavi s predhodno različico. Vse te imformacije dajejo podroben pregled nad zgodovino sprememb.

Opomba
Git in GitHub nista eno in isto. Ljudje pogosto mešajo Git in GitHub. Git je program, ki si ga lahko vsakdo namesti in poganja na svojem računalniku. Program Git je ustvaril Linus Torvalds, da bi lažje upravljal z izvorno kodo za jedro operacijskega sistema Linux. GitHub je javno spletišče, ki je namenjeno skladiščenju Git repozitorijev.

Opomba
Sisteme, ki omogočajo hranjenje preteklih različic datotek, imenujemo sistemi za nadzor različic (angl. version control system (VCS)) ali sistemi za upravljanje z izvorno kodo *(angl. Source Code Management (SCM)).*

Poleg nadzora različic Git omogoča hkratno spreminjanje datotek več uporabnikov na različnih računalnikih. Zato je Git distribuiran sistem za nadzor različic (angl. Distributed Version Control System (DVCS)).

V nadaljevanju bomo obravnavali naslednje teme:

- *Podatkovno skladišče:* Kako Git uporablja zgoščevalno funkcijo in Merklejeva drevesa za hranjenje posnetkov vsebine direktorija.
- *Zgodovina sprememb:* Kako zgodovino predstavimo z usmerjenim acikličnim grafom, v katerem so vozlišča različice in ki povezuje različice z njihovimi neposrednimi predhodniki.
- *Reference*: Kako preproste reference (kazalci) na vsebino omogočajo bliskovito preklaplanje med različicami in preprečijo popolno zmešnjavo, ko več ljudi hkrati spreminja iste datoteke.

2. Podatkovno skladišče

Ko ustvarimo nov Git repozitorij, Git ustvari direktorij z imenom `.git`, ki vsebuje vse podatke, ki jih Git potrebuje. Git v mapi `.git` hrani različne stvari:

- vsebino datotek, s ki smo jih dodali v repozitorij
- drevesno strukturo direktorija
- posnetke stanja v različnih trenutkih s podatki o avtoju, datumu in opisu sprememb
- kazalce na posamezne posnetke stanja

Git repozitorij je vsak direktorij, ki vsebuje poddirektorij `.git` z zgoraj navedenimi podatki. Podrobnosti o tem lahko preberete poglavju o shrambi objektov v knjigi o Gitu.

2.1. Zgoščevalna funkcija

Git ne shranjuje datotek z običajnimi imeni, ampak za ime uporabi vrednost zgoščevalne funkcije njene vsebine. Git uporablja zgoščevalno funkcijo *SHA-1*. Funkcija SHA1 je posebna implementacija zgoščevalne funkcije, ki se uporablja v kriptografiji.

Naj bo B množica vseh možnih podatkovnih nizov(besedil). Zgoščevalna funkcija *SHA1* je funkcija

$$H : B \to \{0, 1, ..., 2^{160} - 1\},$$

ki vsakemu besedilu b priredi 160-bitno *zgoščeno vrednost* besedila $H(b)$. Funkcija H, je izbrana tako, da sprememba enega samega bita v besedilu $b \in B$ spremeni vrednost $H(b)$. Poleg tega zahtevamo, da je porazdelitev vrednosti $H(b)$ čim bližje enakomerni porazdelitvi. To pomeni, da so vse vrednosti $H(b)$ približno enako verjetne. Kljub temu, da zgoščevalna funkcija H ni injektivna, je verjetnost, da bi imela dva podatkovna niza isto vrednost H zelo majhna($\approx 2^{-159}$). Zato lahko v praksi predpostavimo, da je z vrednostjo $H(b)$ niz b enolično določen.

Ko datoteko z vsebino b zabeležimo v Git repozitorij, git shrani vsebino v datoteko z imenom $H(b)$ v `git/objects`[1]. Vsebina b je tako vedno dostopna pod imenom $H(b)$. Tako dobimo vsebinsko naslovljivo shrambo objektov, ki je ena od bistvenih značilnosti Gita. Ta način shranjevanja omogoča, da lahko vedno preverimo, če ima shranjenjena vsebina isto vrednost zgoščevalne funkcije, kot je njeno ime. Lahko tudi shranimo več različic iste datoteke, saj ima vsaka različica drugačno zgoščevalno vrednost. Zgoščevalna vrednost služi tudi kot kontrola, če je prišlo do kvaritve podatkov, ki so shranjeni v Git repozitoriju.

2.2. Datotečna drevesa

V vsebinsko naslovljivo shrambo objektov lahko shranimo vsebino datotek in njihovih prejšnjih različic. A kako ohranimo informacijo o imenu datotek in drevesni strukturi direktorija? Git za to ustvari nov tip objekta *drevo* (angl. *tree*), ki hrani preprost seznam imen datotek in naslovov na vsebino datotek v direktoriju. Naslov na vsebino datoteke b je seveda zgoščena vrednost vsebine $H(b)$. Seznam imen datotek in zgoščenih vrednosti je preprosta tekstovna datoteka, za katero lahko izračunamo zgoščeno vrednost. Zgoščena vrednost datotečnega drevesa natanko določa tako imena datotek, kot tudi vsebino datotek, ki so vsebovane v direktoriju. Če se katerakoli datoteka ali ime datoteke v direktoriju spremeni, se bo spremnila tudi njena zgoščena vrednost in posledično zgoščena vrednost za drevo. Poleg posameznih datotek, lahko drevo vsebuje tudi poddrevesa. Tako lahko rekurzivno ustvarimo drevesno podatkovno strukturo, ki zajema direktorij z datotekami in poddirektoriji v poljubni globini.

[1] V resnici Git shrani vsebino v datoteko z imenom $h_3h_4...h_{40}$ v mapi h_1h_2, kjer je $h_1h_2h_3...h_{40}$ zapis $H(b)$ v 16-tišlkem sistemu. Datoteka, katere vsebina ima zgoščeno vrednost $H(b)$ enako `8dd6d4bdaeff93016bd49474b54a911131759648` bo shranjena v `.git/objects/8d/d6d4bdaeff93016bd49474b54a911131759648`

```
100644 blob bcc1382241e267cf790ca6b3afe9fde6dcf1072f    bla.txt
100644 blob 2ce22b4dc77442103f095503f1205937c1b0fcfc    blabla.txt
040000 tree 605f479464bebe4f7250ace49bab48e72855f84a    podmapa
```

Program 1: Vsebina direktorija v Gitu je preprost seznam datotek in poddirektorijev in zgoščenih vrednosti njihove vsebine

Poglejmo si primer. Denimo, da imamo v naslednjo strukturo datotek in poddirektorijev

```
├── bla.txt (vsebina: bla)
└── mapa
    ├── bla.txt (vsebina: bla)
    └── blabla.txt (vsebina: blabla)
```

Git bo shranil naslednje objekte v vsebinsko naslovljivo shrambo:

- vsebino datoteke `bla.txt`:

Zgoščena vrednost: `bcc1382241e267cf790ca6b3afe9fde6dcf1072f`
`bla`

- vsebino datoteke `blabal.txt`:

Zgoščena vrednost: `2ce22b4dc77442103f095503f1205937c1b0fcfc`
`blabla`

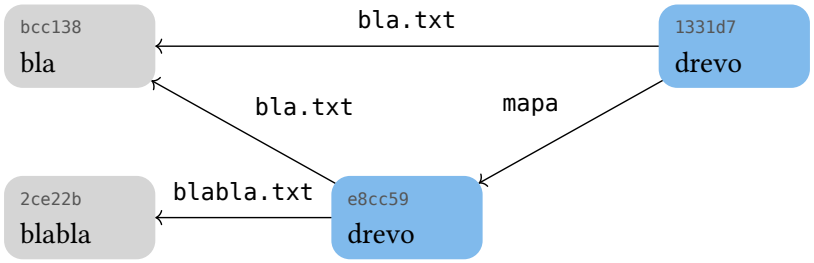
- seznam datotek v direktoriju `mapa`:

Zgoščena vrednost: `e8cc593eddfb9cfdafb4f9c46ab7f5a05ea00b2b`
`100644 blob bcc1382241e267cf790ca6b3afe9fde6dcf1072f    bla.txt` `100644 blob 2ce22b4dc77442103f095503f1205937c1b0fcfc    blabla.txt`

- seznam datotek v korenskem direktoriju:

Zgoščena vrednost: `1331d77b31e40d6b470706b195f21244bb32cf21`
`100644 blob bcc1382241e267cf790ca6b3afe9fde6dcf1072f    bla.txt` `040000 tree e8cc593eddfb9cfdafb4f9c46ab7f5a05ea00b2b    mapa`

Git z uporabo zgoščene vrednosti kot kazalca na vsebino, vsebino direktorija postavi v podatkovno strukturo, ki jo matematično lahko opišemo z *usmerjenim acikličnim grafom*. Ko je vsebina datotek enaka(npr. `bla.txt` in `mapa/bla.txt`), Git shrani le eno kopijo, ki je dostopna v datoteki `.git/objects/bc/c1382241e267cf790ca6b3afe9fde6dcf1072f`. Zato datotečno drevo v Gitu ni nujno drevo, ampak usmerjen aciklični graf.



Slika 1: Primer datotečnega grafa povezanega z zgoščenimi vrednostmi

Posledično lahko celotno vsebino direktorija opišemo z eno samo zgoščeno vrednostjo. Če spremenimo vsebino, ime ali lokacijo datoteke, bo sprememba vplivala na zgoščeno vrednost spremenjene vsebine in sprememba bo splavala na površje do zgoščene vrednosti za korenski direktorij. Zgoščena vrednost služi tako kot identifikator vsebine, kot tudi kot kontrolna vsota, ki omogoča detekcijo sprememb.

Opomba
Podatkovna struktura objektov v Gitu je podobna Merklejevim drevesom. Razlika je v tem, da Gita hrani le eno kopijo datotek z identično vsebino, zato dobimo usmerjen aciklični graf in ne drevesa. Postopek je podoben veriženju blokov, ki se uporablja v kriptovalutah.

Opomba
*Dostop do objekta je mogoč, če poznamo **zgoščeno vrednost** njegove vsebine. To pomeni, da je referenca na posamezen objekt v Gitu preprosto zgoščena vrednost(angl. hash) vsebine tega objekta. Po drugi strani je vsebina objekta določena z njegovo zgoščeno vrednostjo. To pomeni, da lahko enostavno preverimo verodostojnost vsebine, ki je shranjena v Gitu. Git hrani skladišče objektov v direktoriju `.git/objects`.*

3. Zgodovinski graf sprememb

V prejšnjem poglavju smo videli, kako Git hrani vsebino direktorija in kako je mogoče do vsebine dostopati če poznamo zgoščeno vrednost korenskega direktorija. Zgodovinsko drevo sprememb je preprosta razširitev omenjene podatkovne strukture.

3.1. Posnetki stanja

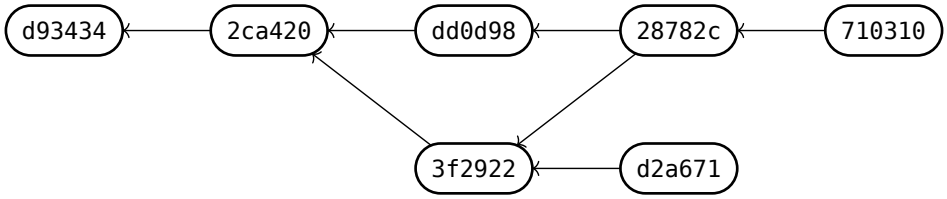
Osnovna enota v Gitu je **Vnos** (angl. **commit**). Vnos je posnetek stanja zabeleženih datotek v trenutku, ko je bil ustvarjen. Poleg vsebine datotek vsak vnos vsebuje še metapodatke o avtorju, datumu vnosa in opisom sprememb. Podobno kot objekt tipa *drevo*, je tudi vnos objekt v vsebinsko naslovljivi shrambi, ki je ima določeno **zgoščeno vrednost vnosa**. Zgoščena vrednost vnosa je natanko določena z vsebino shranjenih datotek in metapodatkov vnosa.

Zgoščena vrednost: `8dd6d4bdaeff93016bd49474b54a911131759648`

`tree 65c47feec7465e80492620a48206793e078702e0`
`parent 16f2994757f1213935b8edb9ae7fee3a8e9ec98d`
`author MV <mv@example.com> 1765235698 +0100`
`committer MV <mv@example.com> 1765235698 +0100`
 	
`Dodaj bla`

Tabela 1: Primer vnosa v Gitu. Vnos vsebuje zgoščeno vrednost posnetka direktorija(`tree`), zgoščeno vrednost starševskega vnosa (`parent`) in metapodatke. Tudi sam vnos je natančno določen z zgoščeno vrednostjo.

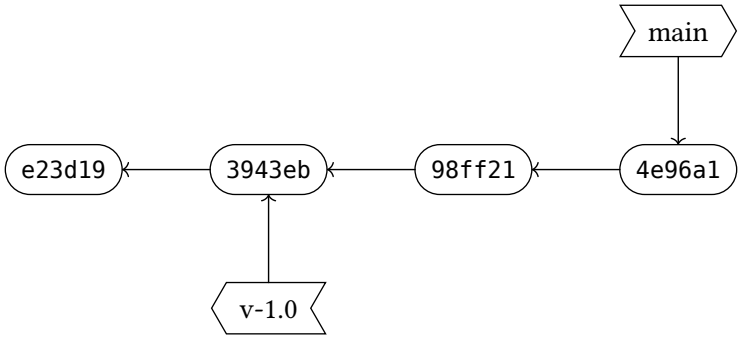
Vsak vnos je povezan s točno določenim posnetekom vsebine korenskega datotečnega drevesa, ki ga identificira zgoščena vrednost. Poleg tega so posamezni vnosi so povezani v **usmerjen acikličen graf (DAG)**, ki predstavlja zgodovino sprememb. Vsak **vnos** je **vozlišče** v grafu. Vsak vnos izhaja iz enega ali več starševskih vnosov. Izjema je prvi vnos. **Povezave** v grafu povezujejo vnose z njihovimi starši.



Slika 2: Vnosi v Gitu kot usmerjen graf. Vsak vnos(razen prvega) ima povezavo na vnose iz katerih izhaja.

4. Kazalci: veje in značke

Poleg objektov kot so *vnosi*, *posnetki direktorijev* in *posnetki datotek* pozna git še reference. Reference so kazalci z določenim imenom na posamezen vnos.

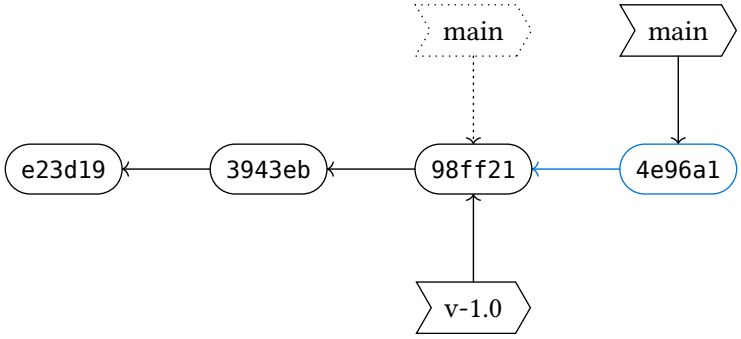


Slika 3: Veja (angl. branch) ali značka(angl. tag) je preprost kazalec na posamezen vnos(angl. commit).

Referenc git ne hrani v skladišču objektov, temveč posebej v direktoriju `.git/refs`. Zato so reference vezane na posamezen repozitorij in se lahko razlikujejo med različnimi kloni določenega repozitorija.

Veja (angl. **branch**) je posebne vrste referenca, ki se premika, ko dodajamo nove vnose. Vsakič ko ustvarimo nov vnos, se trenutno aktivna veja premakne na novo ustvarjeni vnos.

Značka (angl. **tag**) je referenca, ki je statična in se ne premika več, ko jo enkrat ustvarimo.

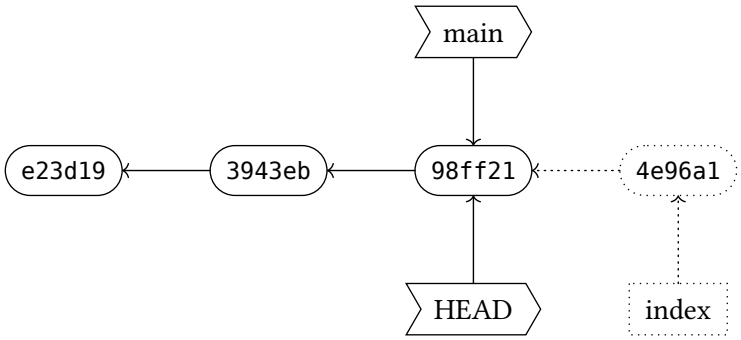


Slika 4: Ko ustvarimo nov vnos, se aktivna veja `main` premakne naprej, značka `v-1.0` pa ostane tam, kjer je bila.

Opomba
Veje in značke nimajo v Gitu nobenega posebnega pomena, razen tega, da so reference na vnose. Pomen posamenznih vej je stvar dogovora med uporabniki. Tako se pogosto uporablja različne veje za različne namene: `main` *ali* `master` *je navadno glavna veja razvoja, veje z imeni* `stable`, `production`, `development` *in podobno označujejo različne stopnje razvoja programske opreme, veje s predpono* `feature-` *označujejo razvoj novih funkcionalnosti.*

Vse te pomene damo vejam ljudje, ki sodelujemo v nekem Git repozitoriju. Za Git so vse veje in značke zgolj preprosti kazalci na določen vnos.

HEAD je posebna referenca, ki kaže na trenutno aktiven vnos. Vnos, na katerega kaže *HEAD* bo starševski vnos naslednjeg vnosa, ki ga bomo dodali.



Slika 5: **HEAD** je referenca na trenutno aktiven vnos/vejo.

4.1. Povzetek

Samostalniki:

- **Vnos** (angl. **commit**) je posnetek trenutnega stanja projekta, shranjen kot vozlišče v zgodovinskem grafu (DAG), ki vsebuje spremembe datotek ter metapodatke (avtor, čas, sporočilo).
- **Zgoščena vrednost vnosa** (angl. **commit hash**) je 40-mestna heksadecimalna vrednost, izračunana s SHA-1, ki enolično identificira vnos na podlagi njegove vsebine.
- **Veja** (angl. **branch**) je premična oznaka, ki kaže na določen vnos v zgodovini in se samodejno premakne naprej, ko dodajamo nove vnose. Veje omogočajo vzporedne razvojne linije z različnimi spremembami.
- **Oznaka** (angl. **tag**) je statična oznaka, ki trajno kaže na določen vnos. Za razliko od veje se oznaka, nikoli ne premika samodejno, zato se uporablja predvsem za označevanje pomembnih točk v zgodovini, kot so izdaje ali stabilne verzije.
- **Delovna kopija** (angl. **workout copy**) je direktorij v katerem urejamo datoteke, ki jih nato vnesemo v Git. V delovni kopiji imajo na začetku datoteke isto vsebino kot je vsebina trenutno aktivnega vnosa (`HEAD`). Spremembe, ki jih naaredimo na delovni kopiji lahko zabeležimo v nov vnos.
- **Oddaljen repozitorij** (angl. **remote**) je povezava(url) na oddaljen repozitorij, s katerim izmenjujemo vsebino.

Glagoli (akcije):

- **Checkout** prenese vsebino vnosa v delovno kopijo: `git checkout neka-veja`
- **Commit** ustvari nov vnos: `git commit -m 'Sporočilo'`
- **Add** doda vsebino, ki bo v naslednjem vnosu: `git add dodaj_me.txt`
- **Pull** poberi vsebino iz oddaljenega repozitorija in uskladi lokalno vejo z oddaljeno: `git pull`
- **Push** potisni lokalne vnose na oddaljeni repozitorij in uskladi oddaljeno vejo z lokalno: `git push`
- **Fetch** pobere nove vnose, veje in oznake iz oddaljenega repozitorija: `git fetch`
- **Reset** spremeni kam kaže trenutno izbrana veja: `git reset a239f9e91`
- **Merge** ustvari nov vnos, ki združi dve ločeni veji v eno: `git merge main`
- **Rebase** prestavi vnose v trenutno izbrani veji na izbran vnos: `git rebase main`

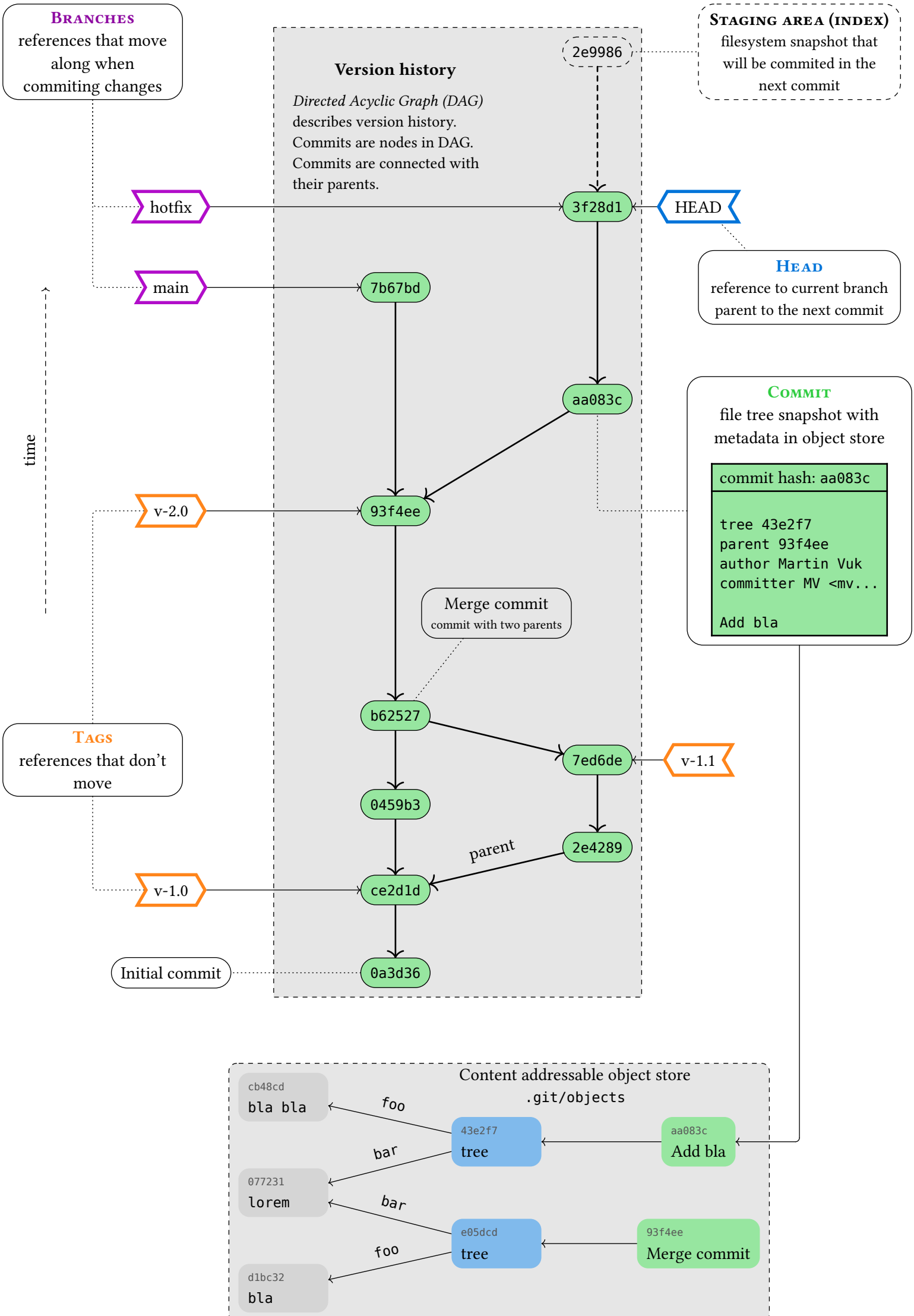
4.2. Delo z Git

Opis dela z Gitom presega namen tega dokumenta. Zato vas raje preusmerimo na uradno dokumentacijo:

https://git-scm.com/cheat-sheet

Pri pripravi dokumenta sem uporabil Gemini 3. Vse odgovore sem preveril in uredil po svoje.

Sledi še skica, ki povzame vse komponente Git repozitorija.


Branches
references that move along when commiting changes
Staging area (index)
filesystem snapshot that will be commited in the next commit
Version history
Directed Acyclic Graph (DAG) describes version history. Commits are nodes in DAG. Commits are connected with their parents.
2e9986
hotfix
3f28d1
HEAD
Head
reference to current branch parent to the next commit
main
7b67bd
aa083c
Commit
file tree snapshot with metadata in object store
commit hash: aa083c
tree 43e2f7
parent 93f4ee
author Martin Vuk
committer MV <mv...
Add bla
time
v-2.0
93f4ee
Merge commit
commit with two parents
b62527
Tags
references that don't move
7ed6de
v-1.1
0459b3
2e4289
parent
v-1.0
ce2d1d
Initial commit
0a3d36
Content addressable object store
.git/objects
cb48cd
bla bla
foo
43e2f7
tree
aa083c
Add bla
bar
077231
lorem
bar
e05dcd
tree
93f4ee
Merge commit
foo
d1bc32
bla