Altruist svn in egocentrik git: dva sistema za vodenje izvedb

Jurij Mihelič

20. oktober 2015

Laboratorij za algoritme in podakovne strukture
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani

Laboratory for Algorithms and Data Structures
Technical Report LALG-2013-1

Author: Jurij Mihelič
Title: Altruist `svn` in egocentrik `git`:
dva sistema za vodenje izvedb

Ljubljana: University of Ljubljana, Faculty of Computer and Information Science,
20. oktober 2015

Kazalo

Ta dokument vsebuje zapiske o sistemih Subversion in Git za nadzor izvedb. Zapiski so nastali kot pomoč pri prvem sklopu vaj iz predmeta Sistemska programska oprema. Zapiski so nepopolni in vsebujejo napake, tipkarske in verjetno tudi vsebinske. Zaradi tega razloga ne prevezemam nobene odgovornosti za kakršnokoli škodo, ki bi morebiti nastala z uporabo teh zapiskov. Vsekakor pa bom hvaležen, če mi v primeru odkritja napake, le-to sporočite. Samo za interno uporabo.

Različica zapiskov: 20. oktober 2015

Poglavje 1

Nadzor izvedb

1.1 Osnovni pojmi

1.1.1 Izvedbe

S pomočjo računalnika in ustrezne programske opreme obdelujemo raznovrstne dokumente, kot so npr.: izvorna koda, besedilo, slika itd. Podatki, ki so del nekega dokumenta, so shranjeni v datoteki. Pogosto se primeri, da dokument ne nastane naenkrat, ampak nastaja postopoma s spremembami oz. dopolnitvami. Vsaka sprememba dokumenta predstavlja njegovo novo *izvedbo* (angl. *revision*).

Če izvedb dokumenta ne vodimo, se pri shranjevanju (v isto datoteko) predhodna izvedba dokumenta prepiše oz. izgubi. Včasih izgube predhodne izvedbe ne želimo, še več, želimo hraniti prav vse izvedbe dokumenta, od njegovega nastanka do njegove trenutne izvedbe.

Nadzor nad izvedbami dokumenta lahko vodimo ročno, npr. tako, da hranimo različne izvedbe v datotekah z različnimi imeni. Takšen pristop ima vrsto slabosti, kot npr.:

- zahteva izjemo disciplino uporabnika,
- poimenovanje datotek, ki hranijo različne izvedbe, je lahko precej zahtevno,
- neekonomičnost izrabe prostora na pomnilnem mediju, ker ne izkorišča podobnosti med posameznimi izvedbami.

1.1.2 Skladišče

Zaradi zgornjih razlogov nadzora izvedb navadno ne izvajamo ročno, razen morda v najpreprostejših primerih. Naprednejši nadzor izvedb omogočajo sistemi za nadzor izvedb. Izvedbe se hranijo v *skladišču* (angl. *repository*). Sistem za vodenje izvedb nudi (vsaj) naslednje:

- shranjevanje izvedb v skladišče (angl. *storing*),
- avtomatsko označevanje izvedb z različicami (angl. *versioning*),
- obnavljanje izvedbe iz skladišča (angl. *retrival*), in
- vodenje dnevnika sprememb (angl. *logging*).

1.1.3 Različice

Razvoj dokumenta navadno poteka po neki poti, *glavni veji* oz. *deblu* (angl. *main branch, trunk*). Vsaka izvedba dokumenta je, zaradi lažjega naslavljanja izvedbe, posebej označena. Oznaka izvedbe je število, ki mu pravimo tudi *različica* (angl. *version*). Zadnji izvedbi v skladišču pravimo tudi *glava* (angl. *head*).

Različice v starejših sistemih, kot sta npr. RCS in CVS, se pričnejo z 1.0 in se z vsako izvedbo povečajo za 0.1. Enostavnejši način pa je številčenje izvedb, ki se prične z 0, vsaka sprememba pa poveča različico za 1. Takšen način uporablja npr. sistem Subversion.

1.1.4 Oznake

Tekom razvoja so nekatere izvedbe dokumenta lahko bolj pomembne kot druge. Predstavljajo lahko doseg nekega začrtanega cilja, ki mu večkrat rečemo *mejni kamen* (angl. *milestone*), ali pa predstavljajo izvedbo, ki smo jo namenili za nadaljnjo distribucijo, t.j. *izdajo* (angl. *release*). Pomembnejše izvedbe zato označimo s posebnimi *oznakami* (angl. *tags*).

1.1.5 Vejitve

Včasih se zgodi, da poleg glavne veje razvoja potrebujemo tudi vzporedno oz. *stransko vejo* (angl. *branch*). Stranske veje pridejo prav predvsem pri npr. razvoju delov programa, ki jih še nimamo namen takoj vključiti v glavno vejo.

1.2 Projektno delo

1.2.1 Več dokumentov

Programski sistem lahko vsebuje več dokumentov. Moderni sistemi za vodenje izvedb zato podpirajo delo s celotnim projektom oz. z več dokumenti hkrati. Dokumenti so lahko organizirani v strukturo imenikov in datotek.

1.2.2 Več razvijalcev

Na večjih projektih hkrati dela več razvijalcev. V takšnih primerih se pogosto primeri, da dva ali več razvijalcev ureja isti dokument. Večkratno hkratno spreminjanje istega dokumenta lahko vodi v konfliktne spremembe. Moderni sistemi za vodenje izvedb nudijo podporo razvijalcem tudi v takih primerih.

1.2.3 Razreševanje konfliktov

Hkratne spremembe istega izvedbe dokumenta s strani dveh ali več razvijalcev je potrebno vse upoštevati. Potrebno je združiti (angl. merge) spremembe vseh razvijalcev, ki pa so si lahko medsebojno nasprotujoče, ni pa to nujno. Nenasprotujoče spremembe, sistem za vodenje izvedb, sam razreši.

Na primer, dva razvijalca iz skladišča obnovita isto izvedbo. Oba na svoji delovni kopiji naredita spremembo. Nato najprej prvi razvijalec shrani spremembe v skladišče. Ko drugi razvijalec poskuša v skladišče shraniti svoje spremembe, je tam že nova izvedba, shranjena s strani prvega razvijalca. Nastane konflikt, ki ga je potrebno razrešiti.

Če je npr. prvi razvijalec dodal besedilo na začetku dokumenta, drugi razvijalec pa na njegovem koncu, potem gre pravzaprav za nenasprotujoči si spremebi. Takšen konflikt sistem za vodenje izvedb navadno sam razreši. V skladišče shrani izvedbo, ki upošteva spremembe obeh razvijalcev.

Če pa konflikta ni možno avtomatsko razrešiti, potem sistem za vodenje izvedb, potrebuje pomoč. Drugi razvijalec mora ročno združiti svoje spremembe in spremembe prvega razvijalca. Sistem za vodenje izvedb, mu je pri tem lahko v pomoč.

1.2.4 Delovna kopija

Tipičen potek del pri uporabi sistema za vodenje izvedb je sledeč:

- obnavljanje zadnje izvedbe projekta iz skladišča (angl. *check out*),
- urejanje in spreminjanje projekta,

- shranjevanje sprememb v skladišče (angl. *check in, commit*),

Po obnovitvi zadnje izvedbe iz skladišča se na lokalnem datotečnem sistemu pojavi kopija projekta iz skladišča, ki ji pravimo *delovna kopija* (angl. *working copy*). Urejanje dokumentov projekta nato poteka na njegovi delovni kopiji (ne pa neposredno v skladišču). Po končanem urejanju spremembe shranimo v skladišče. Urejanje in shranjevanje lahko večkrat ponovimo.

Poglavje 2

Altruist svn

2.1 Osnovni pojmi

2.1.1 Skladišče kot drevo

Tipičen projekt je navadno organiziran v neko drevesno strukturo imenikov in datotek. Subversion vodi izvedbe za celotno projektno drevo. Zato si skladišče najlažje predstavljamo kot zaporedje *posnetkov dreves*, kjer vsako drevo predstavlja svojo izvedbo.

Različica izvedbe je globalna za celotno skladišče oz. shranjeno drevo. To pomeni, da imajo vse datoteke in imeniki, ki pripadajo neki izvedbi, enako število različice. Vsaka izvedba v skladišču torej predstavlja posnetek stanja drevesa v nekem časovnem trenutku.

Oznake začetne različice je 0. Takšno oznako dobi skladišče ob stvaritvi. Vsakič, ko v skladišče shranimo spremembe, se različica poveča za 1.

2.1.2 Struktura skladišča

Subversion neposredno ne podpira dela z več projekti. Vendar lahko vodimo izvedbe za več projektov hkrati v enem skladišču enostavno tako, da za vsak projekt v skladišču ustvarimo svoj imenik.

Subversion tudi neposredno ne podpira dela z oznakami in vejitvami. Oboje zopet enostavno omogočimo z uporabo določene strukture imenikov. Za vsak projekt v njegovem imeniku ustvarimo naslednje tri podimenike:

- `trunk` — za hranjenje glavne veje projekta,
- `tags` -– za hranjenje oznak, in

- `branches` — za hranjenje vejitev.

Seveda si lahko skladišče še dodatno organiziramo, npr. projekte združujemo v skupine itd.

Primer.

```
/libraries/opengl
/libraries/sound
/tetris
/tetris/trunk
/tetris/tags
/tetris/branches
/tictactoe
/tictactoe/trunk
/tictactoe/tags
/tictactoe/branches
/picview
/picview/trunk
/picview/tags
/picview/branches
```

2.1.3 Lastnosti

- Vse transakcije so atomične. To pomeni, da commit uspe za celotno drevo ali pa ne uspe.
- Podpira delo z binarnimi datotekami.
- Dostop do skladišča je lahko: lokalni, preko spletnega strežnika Apache ali preko protokola ssh.
- Skladišče je shranjeno kot podatkovna baza Berkeley DB ali od leta 2003 naprej kot FSFS (format, razvit posebej za Subversion).
- Omogoča shranjevanje metapodatkov, t.j. lastnosti.
- V delovni kopiji se nahaja administrativni poddirektorij z imenom .svn.
- Hiter mrežni prenos je omogočen z prenosom binarnih razlik, poleg tega se, če je le mogoče uporabljajo kopije originalov, ki se nahajajo v `.svn`.

2.1.4 Naslavljanje

Pri uporabi Subversiona moramo pogosto podati naslov, kjer se nahaja projekt, skladišče itd. Primer celotnega naslova datoteke je

```
svn://svn.igre.si/tetris/src/main.c
```

Naslov sestoji iz več delov:

- protokol, npr. `svn`,
- ločilo `://`,
- ime računalnika, npr. `svn.igre.si`,
- ime skladišča, npr. `tetris`, in
- naslov oz. pot datoteke, npr. `src/main.c`.

2.1.5 Protokoli

Subversion pozna naslednje protokole:

file Dostop do skladišča, ki se nahaja na lokalnem datotečnem sistemu. V tem primeru seveda ime računalnika in skladišča izpustimo – navedemo le naslov datoteke. Na primer: `file:///home/jure/skladisce`.

svn Dostop do skladišča preko omrežja s pomočjo Subversionu lastnega protokola. Na primer: `svn://svn.host.au/repo/data/list.txt`.

svn+ssh Dostop do skladišča preko omrežja s pomočjo Subversionu lastnega protokola, pri čemer ta nadalje uporablja še varen protokol `ssh` (angl. *secure shell*). Na primer: `svn+ssh://svn.host.nz/repo/data/list.txt`.

http Dostop do skladišča preko spletnega protokola `http` (angl. *hyper-text transfer protocol*). Na primer: `http://svn.thonthon.se/thor/thunder/strom.bin`.

https Dostop do skladišča preko varnega spletnega protokola `https` (angl. *secure hyper-text transfer protocol*). Na primer: `https://svm.apa.at/oll/serber/ht`

2.2 Osnovna uporaba

2.2.1 Orodje `svn`

Orodje oz. program `svn` združuje večino funkcionalnosti sistema Subversion. Prvi argument ukaza je navadno ukaz, katerega želimo izvesti, nato sledijo parametri ukaza:

```
svn ukaz parametri
```

V nadaljevanju si oglejmo pomembnejše ukaze orodja v naslednjih sklopih:

- poizvedbe po raznih informacijah,
- delo neposredno v skladišču,
- delo na delovni kopiji projekta.

2.2.2 Pomoč: `svn help`

Kratko pomoč za uporabo orodja `svn` dobimo z

```
svn help
```

Poglavitni del kratke pomoči je spisek ukazov (in njihovih vzdevkov), ki jih orodje `svn` podpira.

Pogosto uporabna je pomoč za posamezen ukaz, ki jo dobimo z

```
svn help ukaz
```

Na primer `svn help commit` izpiše kratko pomoč za uporabo ukaza `commit`. Pri tem namesto `help` lahko uporabimo tudi enega izmed njegovih vzdevkov `?` oz. `h`. Tako `svn ? mkdir` izpiše pomoč za ukaz `mkdir` in `svn h info` za ukaz `info`.

2.3 Poizvedbe

2.3.1 Informacije o datoteki: `svn info`

Osnovne informacije o datoteki na nekem naslovu izpišemo z

```
svn info naslov
```

Če naslova ne podamo, se izpišejo informacije o lokalni kopiji. Ukaz izpiše podatke kot so:

- različica datoteke,
- tip datoteke (angl. *node kind*): imenik, datoteka, itd.,
- avtor zadnje spremembe (angl. *last changed author*),
- različica zadnje spremembe (angl. *last changed rev*), in
- datum zadnje spremembe (angl. *last changed date*).

Oglejmo si še primer. Tipična poizvedba je: ali je na danem naslovu (npr. na naslovu `svn://svn.firma.si/skladisce`) skladišče.

```
jure@ris:~> svn info svn://svn.firma.si/skladisce
Path: skladisce
URL: svn://svn.firma.si/skladisce
Repository Root: svn://svn.firma.si/skladisce
Repository UUID: 75c53443-3482-44fd-ad46-7fc87372e7bd
Revision: 6
Node Kind: directory
Last Changed Author: jure
Last Changed Rev: 6
Last Changed Date: 2012-10-17 12:55:49 +0200 (Wed, 17 Oct 2012)
```

Seveda lahko naredimo tudi poizvedbo po datoteki znotraj projekta,

```
jure@ris:~> svn info tags/v1.1/docs/preberi.txt
```

Včasih ciljna datoteka ne obstaja, takrat `svn` izpiše sporočilo o neobstoju in vrne izhodni status 1. Na primer:

```
jure@ris:~> svn info
svn: '.' is not a working copy
jure@ris:~> echo $?
1
```

2.3.2 Izpis vsebine imenika: `svn list`

Izpis seznama datotek iz skladišča, ki se nahajajo na podanem naslovu, dosežemo z

```
svn list naslov
```

Če je podani *naslov* navadna datoteka, potem se izpiše le ime datoteke; če pa je imenik, potem se izpišejo vse datoteke v njem. Pri tem lahko uporabimo nekatera stikala

- `-r` *rev* ... različica datoteke,,
- `-v` ... obsežnejši izpis, in
- `-R` ... izpis hierarhije imenikov in datotek

Če je dani naslov lokalen, potem se upošteva njemu ustrezen naslov v skladišču. Na primer izpis delovne kopije ustreza izpisu

jure@ris:~> `svn list`

2.3.3 Zgodovina sprememb: `svn log`

Zgodovino sprememb za nek dokument na lokaciji naslov v skladišču izpišemo z:

```
svn log naslov
```

Izpiše vse različice spremenjenih izdaj dokumenta na podanem naslovu, poleg tega pa še informacije kot: uporabnik, ki je shranil izdajo, kratek opis sprememb, datum in čas spremembe.

2.3.4 Vsebina datoteke: `svn cat`

Vsebino datoteke izpišemo z:

```
svn cat naslov
```

2.4 Upravljanje z datotekami

Ukaze `mkdir`, `copy`, `move` in `delete`, obravnavane v tem razdelku, lahko uporabimo nad datotekami tako v skladišču kot v delovni kopiji. V prvem primeru se pri uporabi ukaza poveča tudi različica skladišča (lahko rečemo tudi da se po ukazu izvede še avtomatski oz. implicitni `commit`), v drugem pa se različica poveča šele ob shranjevanju sprememb (potreben je torej eksplicitni `commit`) v skladišče.

2.4.1 Ustvarjanje imenika: `svn mkdir`

Enega ali več imenikov ustvarimo z

```
svn mkdir ime. . .
```

Pri tem sta uporabni še stikali:

- `-q` ... utišanje izpisa, in
- `--parents` ... ustvarjanje vmesnih imenikov.

Kot primer si oglejmo ustvarjanje štirih podimenikov v trenutnem imeniku.

jure@ris:~> `svn mkdir boo bar foo far`
```
A    boo
A    bar
A    foo
A    far
```

V izpisu vidimo, da je status vseh štirih novo ustvarjenih imenikov `A`, kar pomeni, da so označeni za dodajanje v skladišče.

Če imenik ustvarimo neposredno v skladišču, se avtomatsko poveča njegova različica.

jure@ris:~> `svn mkdir svn://svn.freebit.com/tetris/src`
```
Committed revision 42.
```

2.4.2 Kopiranje datoteke: `svn copy`

Za Subversion pomemben je tudi ukaz

```
svn copy izvor ponor
```

ki skopira datoteko $izvor$ v datoteko $ponor$. Pri tem je mogoče izvorno datoteko naslavljati tudi preko različice. Na primer

jure@ris:~> `svn copy doc/beri.txt@10 preberi.me`
```
A    preberi.me
```

skopira datoteko `doc/beri.txt` različice 10 (čeprav morda obstaja novejša različica) v datoteko `preberi.me`.

Pomembna uporaba ukaza `copy` je tudi ustvarjanje oznak in vejitev. Oznako naredimo tako, da imenik `trunk` kopiramo v `tags/`$oznaka$. Ker gre za izredno pomembno opravilo, bomo temu kasneje posvetil celoten razdelek.

2.4.3 Preimenovanje datoteke: `svn move`

Prestavljanje oz. preimenovanje datoteke dosežemo z:

```
svn move izvor ponor
```

2.4.4 Odstranjevanje datoteke: `svn delete`

Datoteko odstranimo z ukazom:

```
svn delete naslov
```

2.5 Upravljanje s projektom

Najprej seveda iz skladišča obnovimo projekt. S tem dobimo delovno kopijo projekta, ki jo v nadaljevanju urejamo. Na koncu spremembe shranimo v skladišče.

2.5.1 Obnavljanje projekta: `svn checkout`

Projekt iz skladiščne lokacije naslov obnovimo z:

```
svn checkout naslov
```

Dovoljeno je le naslavljanje imenikov. Po obnovitvi se v trenutnem delovnem imeniku pojavi imenik, ki vsebuje delovno kopijo naslovljenega projekta.

Pri uporabi ostalih ukazov nad delovno kopijo, ni potrebno podajati naslova skladišča, ker svn že sam ve iz katerega skladišča smo obnovili delovno kopijo.

2.5.2 Shranjevanje projekta: `svn commit`

Shranjevanje projekta v skladišče izvedemo z ukazom:

```
svn commit
```

Subversion si zapomni iz katere lokacije je obnovil delovno kopijo, zato ni potrebno podajati naslova, kam naj se shrani.

Če projekt dejansko vsebuje spremembe glede na zadnjo izvedbo v skladišču, potem se spremembe shranijo v skladišče in različica se poveča.

V primeru konfliktov med delovno kopijo in zadnjo izvedbo, Subversion skuša konflikt samostojno razrešiti, če mu ne uspe je potrebno konflik razrešiti ročno.

2.5.3 Status projekta: `svn status`

Status datotek v delovni kopiji izpišemo z:

```
svn status
```

ali, če želimo izvedeti status za neko datoteko:

```
svn status datoteka
```

Pri tem lahko uporabimo nekatera stikala:

- `-v` ... izpis več informacij o datotekah; in
- `-u` ... poizvedba po dejanskem stanju v skladišču.

2.5.4 Ponovna obnovitev: `svn revert`

V primeru, da želimo neko datoteko ponovno obnoviti iz skladišča, lahko to storimo z ukazom:

```
svn revert datoteka
```

2.5.5 Dodajanje datoteke: `svn add`

Ko v delovni kopiji ustvarimo novo datoteko in zanjo želimo voditi izvedbe, moramo sistemu Subversion to povedati z ukazom:

```
svn add datoteka
```

Datoteka se s tem ukazom še ne shrani v skladišče, ampak se le označi za kasnejše shranjevanje v skladišče z ukazom svn commit.

2.6 Razreševanje konfliktov

2.6.1 Primerjava datotek: `svn diff`

Dve izvedbi nekega dokumenta lahko primerjamo med seboj z ukazom:

```
svn diff datoteka
```

Pri tem se trenutna izvedba v delovni kopiji primerja z zadnjo izvedbo v skladišču.

Če želimo primerjati trenutno izvedbo s poljubno različico, potem uporabimo stikalo `-r`.

2.6.2 Posodabljanje delovne kopije: `svn update`

Naslednji ukaz se uporablja, ko shranjevanje delovne kopije v skladišče zaradi konfliktov ne uspe. V tem primeru moramo najprej posodobiti delovno kopijo z ukazom:

```
svn update
```

Datoteke, ki jih v delovni kopiji nismo spreminjali, se posodobijo, t.j. se po potrebi obnovijo iz skladišča.

Z datotekami, ki so v delovni kopiji spremenjene, pa se zgodi naslednje:

če je to mogoče, se izvede avtomatska razrešitev konflikta; sicer pa se v delovni kopiji ustvarijo datoteke s končnicami `.mine`, `.rX` in `.rY`. Pri tem `.mine` vsebuje spremenjeno različico, `.rX` in `.rY` pa različici v konfliktu. S pomočjo teh datotek sami ročno razrešimo konflikt, nato pa moramo sistemu Subversion z ukazom `svn resolved` eksplicitno sporočiti, da je konflikt razrešen.

2.6.3 Razreševanje konflikta: `svn resolved`

Razrešitev konflikta z izvedbami datoteke sistemu Subversion sporočimo z ukazom:

```
svn resolved datoteka
```

2.7 Uvoz in izvoz

2.7.1 Uvoz: `svn import`

Začetni uvoz nekega projekta v skladišče na lokacijo naslov lahko opravimo z ukazom:

```
svn import projekt naslov
```

2.7.2 Izvoz: `svn export`

Projekt lahko izvozimo iz skladišča z:

```
svn export naslov
```

2.8 Pogosta opravila

2.8.1 Ustvarjanje projektnega drevesa

Preden začnemo delati z neki projektom, moramo ustvariti imenike, ki podpirajo glavno vejo, oznake in vejitve. Npr. za projekt `tetris` v skladišču `svn://svn.shrani.si/igre/` to naredimo takole:

```
svn mkdir svn://svn.shrani.si/igre/tetris -m "Imenik projekta tetris."
svn mkdir svn://svn.shrani.si/igre/tetris/trunk -m "Glavna veja projekta tetris."
svn mkdir svn://svn.shrani.si/igre/tetris/tags -m "Oznake projekta tetris."
svn mkdir svn://svn.shrani.si/igre/tetris/branches -m "Vejitve projekta tetris."
```

2.8.2 Urejanje glavne veje

Naslednji ukaz obnovi glavno vejo projekta `tetris`:

jure@ris:~> `svn checkout svn://svn.shrani.si/igre/tetris/trunk`

Če želimo urejati projekt, se najprej prestavimo v imenik `trunk`:

jure@ris:~> `cd trunk`

Datoteko `main.c` lahko pričnemo urejati z:

jure@ris:~> `vi main.c`

Včasih želimo neko datoteko dodati projektu. Najprej jo ustvarimo, npr. z:

jure@ris:~> `touch piece.c`

nato pa jo označimo za vodenje izvedb:

jure@ris:~> `svn add piece.c`

2.8.3 Ustvarjanje oznake

Recimo, da projekt `tetris` zaključimo in želimo zadnjo izvedbo distribuirati uporabnikom kot izdajo. V tem primeru lahko naredimo novo oznako z:

jure@ris:~> `svn copy svn://svn.shrani.si/igre/tetris/trunk svn://svn.shrani.si/igre/tetris/tags/release1`

Poglavje 3

Egocentrik git

3.1 Uvod

Edina izčrpna referenca je git help ukaz. Namen tega učbenika ni popolno razumevanje vsega.

3.1.1 Izpis pomoči: `git help`

```
git help
```

```
git help ukaz
```

3.1.2 Izpis statusa: `git status`

Izpis statusa je eno izmed najpogostejših opravil pri delu s sistemom za vodenje izvedb. Vključuje izpis statusa oz. stanja datotek (npr. spremenjena, nedotaknjena), status skladišča, izpis trenutne aktivne veje, itd.

Prav pride v različnih primerih, npr. kadar:

- pričnemo z delom in se želimo seznaniti, katere datoteke smo spreminjali, dodajali itd.
- tekom dela preverjamo stanje datotek
- preverjamo, če je nek ukaz naredil tisto, kar smo od njega pričakovali.

Dobro je, če se navadimo izpisovati status tudi kar tako med delom, da hitro preverimo, če je stanje datotek res takšno, kot si predstavljamo.

Ukaz za izpis statusa uporabljamo na naslednji način

```
git status
```

Pri tem je dobro poznati še nekaj osnovnih stikal:

- `--long` ... vklop dolgega formata izpisa, dolg format pomeni, da je izpis malce bolj obširen, ta način je tudi privzeto vklopljen
- `--short` ... vklop kratega formata izpisa
- `-s` ... enako kot `--short`
- `--porcelain` ... format izpisa primeren za nadaljnjo računalniško obdelavo
- `-z` ... posamezne vrstice loči med seboj z znakom `NUL` namesto z znakom za novo vrstico (vključi se tudi `--porcelain`)
- `--branch` ... vključi izpis statusa veje (tudi v kratkem formatu)
- `-b` ... enako kot `--branch`.

Več primerov uporabe ukaza `git status` si bomo pogledali sproti v nadaljavenju. Za začetek si oglejmo le primer izpisa statusa, pri čemer še nimamo ustvarjenega skladišča; skratka še ne uporabljamo sistema git za hranjenje izvedb izvorne kode.

jure@ris:~> `git status`
```
fatal: Not a git repository (or any of the parent
directories): .git
```

Kakšen je pomen izpisanega sporočila? Git nas opozori, da niti trenutni imenik, niti nobeden od njegovih starševskih imenikov ne vsebuje skladišča. Torej vodenje izvedb torej ni mogoče – najprej je potrebno ustvariti skladišče. Kako, pa si poglejmo v nadaljevanju.

3.2 Skladišče

3.2.1 Inicializacija skladišča: `git init`

Ustvarjanje in inicializacijo novega skladišča izvedemo z naslednjim ukazom

```
git init ime
```

pri tem lahko podamo ime imenika oz. projekta, preko katerega bomo v nadaljevanju dostopali do skladišča. Če tege ne storimo, se privzame trenutni

delovni imenik. Skladišče je skrito in ga zato pri navadnem izpisu vsebine imenika ne vidimo.

Oglejmo si še par stikali:

- `--quiet` ... utišanje izpisa (obvestilo o stvaritvi novega skladišča se ne izpiše)
- `-q` ... enako kot `quiet`
- `--bare` ... surovo skladišče (več o tem kasneje)
- `--separate-git-dir=`$imenik$ ... delovni imenik in imenik, ki hrani skladišče, lahko ločimo.

Še opomba: če `git init` ponovno poženemo ne bomo povzročili škode, npr. nenamerno izgubili vse zgodovine spremembe, le morebitne novo dodane predloge bomo uvozili v projekt.

3.2.2 Imenik `.git`

Ukaz `git init` v podanem imeniku ustvari skladišče. Kako pa izgleda skladišče? Za naše potrebe bo dovolj, če vemo le, da se v podanem imeniku ustvari podimenik z imenom `.git`, ki predstavlja skladišče. Kaj se skriva v njem, za nas niti ni pomembno. Pomembno pa je, da ga pustimo pri miru – ne spreminjamo ali brišemo datotek v njem.

Ime imenika `.git` se začne s piko. Dogovor v operacijskih sistemih vrste Unix pravi, da je takšno ime skrito, zato ga pri tipični uporabi ukazov ne vidimo. S tem nas skladišče ne moti pri vsakdanjem delu. Pravzaprav niti ni nujno, da se skladišče res hrani v podimeniku `.git`. To se npr. zgodi pri uporabi stikal `--bare` ali `--separate-git-dir`.

V skladišču se hranijo vse datoteke, vsa zgodovina sprememb in vsi ostali potrebni podatki. Za radovedne pokukajmo še v tipični imenik `.git`. Vsebuje tri datoteke: `HEAD`, `config` in `description` in podimenike `hooks`, `objects`, `branches`, `info` in `refs`. Imena teh datotek že marsikaj povejo, več pa bo postalo jasno, kot bomo git podrobneje spoznali.

3.2.3 Zgled ustvarjanja skladišča

Pojasnimo ustvarjanje skladišča še s primerom. Recimo, da želimo sprogramirati igrico tetris. Ustvarili bomo ustrezen imenik, kjer bomo hranili vso izvorno kodo. Pri tem seveda želimo uporabiti git za vodenje izvedb. Postavimo se v imenik, kjer želimo ustvariti nov projekt in poženemo

```
jure@ris:~> git init tetris
Initialized empty Git repository in /home/jure/tetris/.git/
jure@ris:~> cd tetris
jure@ris:tetris>
```

Izpiše se kratko sporočilo o uspešni stvaritvi skladišča, poleg tega tudi pot do imenika, kjer se dejansko hrani skladišče. Na koncu se še postavimo v ustrezni imenik.

Vse skupaj bi seveda lahko naredili tudi na malce daljši način, pri katerem sami ustvarimo delovni imenik in šele nato skladišče:

```
jure@ris:~> mkdir tetris
jure@ris:~> cd tetris
jure@ris:tetris> git init
Initialized empty Git repository in /home/jure/tetris/.git/
```

V obeh primeri imamo na voljo imenik `tetris`, v katerem bomo v nadaljevanju delali vse v zvezi s projektom.

Prav tako izpis status v obeh primerih izpiše naslednje

```
jure@ris:tetris> git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

3.3 Shranjevanje v skladišče

3.3.1 Področja datotek

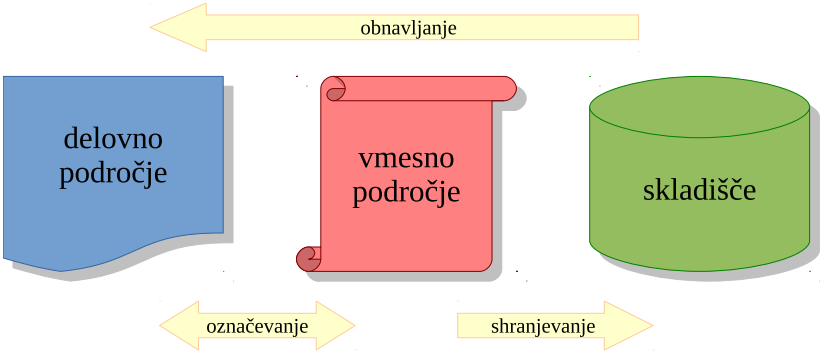
Zdaj torej znamo ustvariti skladišče. Kako pa v skladišče shranimo datoteke? Da bomo odgovorili na to vprašanje, moramo najprej spoznati osnovni način dela s sistemom git. Obdelovane datoteke se nahajajo v eni ali več *stopnjah* ali *področjih*.

Delovno področje vsebuje vse datoteke in imenike, ki so del projekta. Datoteke v tem področju urejamo, spreminjamo, ustvarjamo nove ali jih brišemo. Sprememba datoteke v tem področju se ne odraža v skladišču; datoteke lahko tukaj po mili volji obdelujemo.

V zgornjem zgledu se kot delovno področje smatra hiererahija imenikov zasidrana v imeniku `/home/jure/git/tetris`. Podimenik `.git`, ki se sicer tehnično nahaja znotraj področja, ne štejemo v delovno področje.

Vmesno področje je posebno področje, ki vsebuje datoteke, označene za shranjevanje v skladišče. Spremenjene datoteke iz delovnega področja, se ne bodo avtomatsko shranile v skladišče, ampak je potrebno to sistemu git posebej povedati. To naredimo z ukazom `git add`, ki jih označi podane datoteke za shranjevanje v skladišče – rečemo tudi, da smo datoteke prestavili v vmesno področje. Seveda pri slednjem ne gre za nikakršno prestavljanje datotek; sistem git si le zapomni, katere datoteke smo podali. Datoteke je z ukazom `git rm` iz vmesnega področja moč tudi odstraniti.

Skladišče hrani vse datoteke, njihove izvedbe in še nekatere spremljajoče podatke. Sistem git tipično hrani skladišče v podimeniku `.git`. Včasih mu rečemo tudi lokalno skladišče. Le datoteke iz vmesnega področja se ob izvedbi ukaza `git commit` shranijo v skladišče. Iz skladišča je možno datoteke tudi obnoviti. Pri tem lahko obnovimo zadnjo izvedbo, ali pa katerokoli od prejšnjih izvedb. Obnova datotek vedno poteka neposredno iz skladišča v delovno področje.



Slika 3.1: Področja in prehodi med njimi

Grafični prikaz področij je podan na sliki 3.1, kjer vidimo tudi možne prehode med področji: - označevanje - shranjevanje - obnavljanje TODO

3.3.2 Dodajanje v vmesno področje: `git add`

Datoteko ali več datotek dodamo v vmesno področje z ukazom

```
git add ime...
```

Osnovna uporaba ukaza ne vsebuje večjih posebnosti. Če podamo imenik, bo git označil vse v imeniku in njegovih podimenikih vsebovane datoteke.

Opozorimo še na to, da se v vmesno področje doda trenutna izvedba datoteke. Če datoteko v nadaljavaju pred shranjevanjem v skladišče še spremenimo, se novo nastale spremembe ne bodo shranile v skladišče. V skladišče se vedno shrani izvedba, ki smo jo označili. V izogim zapletom ukaz `git add` največkrat uporabljamo tik preden z `git commit` poženemo shranjevanje v skladišče.

3.3.3 Odstranjevanje iz vmesnega področja: `git rm`

Včasih datoteko pomotoma dodamo v vmesno področje. Storjeno lahko prekličemo tako, da datoteko odstranimo iz vmesnega področja z ukazom

```
git rm ime...
```

Podamo lahko eno ali več datotek. Če podamo imenik, potem je smiselno uporabiti še stikalo `-r`, ki vklopi obdelavo datotek v podimenikih.

Ukaz ne odstrani datoteke iz delovnega področja (datotečnega sistema), le iz vmesnega področja. Če želimo datoteko dejansko zbrisati, potem uporabimo klasičen ukaz `rm`.

3.3.4 Ponastavitev vmesnega področja: `git reset`

Včasih se tekom dela premislimo in želimo ponovno izbrati datoteke za shranjevanje v skladišče. Takrat uporabimo ukaz

```
git reset
```

Po izvedbi ukaza bomo morali vse datoteke znova dodati v vmesno področje, če jih želimo shraniti v skladišče.

3.3.5 Shranjevanje v skladišče: `git commit`

Datoteke so torej pripravljene v vmesnem področju, napočil je čas, da jih dejansko shranimo v skladišče. Preden res izvedemo shranjevanje, preverimo če je vse kot mora bit. Je izvorna koda v konsistentnem stanju? Se program normalno prevede? Pogosto imajo podjetja svoja pravila, ki določajo, kdaj lahko opravimo shranjevanje v skladišče. Skratka, če je v v najlepšem redu, potem shranjevanje v skladišče izvedemo z ukazom

```
git commit
```

Git bo sprožil privzeti urejevalnik besedil in prosil uporabnika, da vnese kratek opis izvedenih sprememb.

Izmed osnovnih stikal sta uporabna predvsem naslednja:

- `-m` *opis* ... opis podamo lahko kar v ukazni vrstici
- `-a` ... git bo ignoriral vmesno področje – sam bo poiskal datoteke, ki so se spremenile in jih shranil v skladišče. Novo nastale datoteke bo ignoriral.

3.3.6 Urejevalnik opisa

Brez uporabe stika `-m` bo git zagnal privzeti urejevalnik besedil, v katerega je potrebno vnesti kratek opis in ga shraniti. Šele nato bo git dejansko shranil spremembe v skladišče.

Kateri urejevalnik pa je privzeti? Na Unix sistemih je to navadno kar `vi`. Včasih njegove uporabe nismo vešči in bi ga radi zamenjali s čim priročnejšim. To storimo tako, da nastavimo spremenljivko `GIT_EDITOR`.

Oglejmo si primer.

jure@ris:~> `export GIT_EDITOR=nano`

Ob vsakem nadaljnjem shranjevanju se bo za urejanje opisa tako uporabil urejevalnik `nano`.

3.3.7 Opis izvedbe

Kako sestaviti dober opis izvedenih sprememb? Nekakšno pravilo je, da se opis začne z glagolom ter kratko in jedrnato obrazloži nastale spremembe. Pogosto so opis kar v angleščini (nikoli ne veš, kdaj se bo projektni skupini pridružil član iz tujine). Naštejmo še nekaj primerov.:

Dodan gumb za preklic
Popravljene tipkarske napake
Add cancel button
Correct typos

Prva vrstica opisa hkrati predstavlja tudi *naslov* opisa. Nič nas ne omejuje, da opis ne bi bil sestavljen iz več vrstic, le naslednih pravil se je potrebno držati:

- Prva vrstica je naslov opisa.
- Druga vrstica je prazna.

- Nato sledi ena ali več vrstic podrobnejšega opisa izvedenih sprememb.

Še kratek primer:

Dodan gumb za preklic

Na iskalni obrazec je na prošnjo uporabikov
dodan gumb za preklic iskanja.

3.3.8 Zgled shranjevanja v skladišče

Nadaljujmo zgled iz prejšnjega razdelka. Ustvarili smo novo prazno skladišče in se postavili v začetni imenik delovnega področja. Sedaj najprej ustvarimo in nato uredimo datoteko `main.c`. Po urejanju izpišimo status.

jure@ris:tetris> `vi main.c` *... ustvarimo in uredimo datoteko main.c*
jure@ris:tetris> `git status` *... izpišemo status*

```
On branch master

Initial commit

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

        main.c

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

Kot vidimo, git obvesti, da datoteki `main.c` ne sledi (angl. *untracked files*). Poleg tega izpiše še, da smo na veji `master`.

Dodajmo datoteko v vmesno področje in zopet izpišimo status.

jure@ris:tetris> `git add main.c` *... dodajanje datoteke v vmesno področje*
jure@ris:tetris> `git status`

```
On branch master

Initial commit

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

        new file: main.c
```

Datoteka `main.c` je sedaj pripravljena za prenos v skladišče. Storimo še to.

```
jure@ris:tetris> git commit -m "Dodan glavni program"
jure@ris:tetris> git status
On branch master
nothing to commit, working directory clean
```

3.4 Brskanje po skladišču

3.4.1 Izpis zgodovine izvedb: `git log`

Pogosto nas zanima, kaj imamo shranjeno v skladišču. Izpis shranjenih izvedb dobimo z ukazom

```
git log
```

Ukaz podpira veliko načinov delovanja, oglejmo si le nekatere:

- `-`N ... omejitev števila izpisanih izvedb na kvečjemu N
- `--format=`$oblika$ ... nastavitev formata izpisa. Možni parametri so: `oneline`, `short`, `medium`, `full`, `fuller`, `email` in `raw`. Privzeti način izpisa je `medium`.
- `--pretty=`$oblika$ ... enako kot format
- `--abbrev-commit` ... skrajšana oznaka izvedbe
- `--name-only` ... izpis spremenjenih datotek
- `--name-status` ... izpis spremenjenih datotek in statusa
- `--stat` ... vklop izpisa povzetka sprememb med posameznimi izvedbami: št. spremenjenih datoteke, vrstic itd.
- `--shortstat` ... podobno kot `--stat` le da gre za krajši povzetek
- `--graph` ... vklop grafične predstavitve odvisnosti med izvedbami
- `-p` ... vklop izpisa razlike med posameznimi izvedbami.
- `--oneline` ... enako kot `--format=oneline --abbrev-commit`

Privzeto se za vsako izvedbo izpišejo naslednji podatki:

Oznaka izvedbe sestoji iz zaporedja štiridesetih alfanumeričnih znakov, npr. `275e1cedea9f75cf48aec69ee2a4c5a9bf4a00e2`. Ker preko oznake enolično identificiramo izvedbo, morajo biti oznake vseh izvedb med seboj različne.

Avtor je oseba, ki je izvedbo shranila v skladišče. Sestoji iz uporabniškega imena in elektronskega naslova.

Datum pove čas, ko je bila izvedba shranjena v skladišče.

Opis je kratko besedilo, ki opisuje izvedene spremembe.

Oglejmo si še primer izpisa ukaza `git log`. Nadaljujmo primer iz prejšnjega razdelka, kjer smo v skladišče shranili datoteko `main.c`.

```
jure@ris:tetris> git log
commit 3cbe6c7044386893d1f7b98f14b07267479c080e
Author: jurem <jure.mihelic@fri.uni-lj.si>
Date:   Sat Feb 7 22:50:39 2015 +0100

  Dodan glavni program
```

Oznaka `3cbe6c7044386893d1f7b98f14b07267479c080e` torej enolično identificira shranjeno izvedbo. V izpisu vidimo tudi avtorja in čas izvedbe. Prav tako se izpiše tudi opis, ki smo ga podali ob shranjevanju.

3.4.2 Oznaka izvedbe

Kot smo že povedali, je oznaka izvedbe zaporedje štiridesetih alfanumeričnih znakov, npr.:

`3cbe6c7044386893d1f7b98f14b07267479c080e`.

Če oznako malce bolj podrobno pogledamo, opazimo, da gre za desetiške številke in črke od `a` do `f`. To pa so pravzaprav šestnajstiške številke, oznaka izvedbe je torej število zapisano v šestnajstiški obliki.

Ker ena šestnajstiška številka predstavlja štiri bite, je celotna oznaka v resnici 160 bitno število. Kako pa se izračuna to število? Sistem git v ta namen uporablja kriptografski algoritem `SHA-1` (angl. *secure hash algorithm*).

Če moramo pri uporabi pogosto tipkati celotno oznako, to lahko postane izredno duhamorno. Ker pa so oznake zelo drugačne druga od druge, se navadno razlikujejo že po nekaj prvih številkah. Zato sistem git omogoča, da navedemo le prvih nekaj številk (vsaj štiri) oznake. Tako so na primer vse naslednje oznake enakovredne zgoraj zapisani oznaki:

`3cbe6c7044386893d1f7`, `3cbe6c70` in `3cbe`.

3.4.3 Zgledi izpisa zgodovine

Sedaj spremenimo datoteko `main.c`, jo dodajmo v vmesno področje in shranimo v skladišče. Nato si še enkrat oglejmo zgodovino izvedb.

```
jure@ris:tetris> git log
commit 275e1cedea9f75cf48aec69ee2a4c5a9bf4a00e2
Author: jurem <jure.mihelic@fri.uni-lj.si>
Date:   Wed Feb 18 09:23:10 2015 +0100

  Izpis sporočila

commit 3cbe6c7044386893d1f7b98f14b07267479c080e
Author: jurem <jure.mihelic@fri.uni-lj.si>
Date:   Sat Feb 7 22:50:39 2015 +0100

  Dodan glavni program
```

V skladišču sta sedaj shranjeni dve izvedbi. Prva je izpisana novejša, nato starejša.

Naredimo še eno spremembo in shranimi vse skupaj v skladišče. Nato uporabimo skrajšan izpis:

```
jure@ris:~> git log --oneline
1ef75cf Dodan skok v novo vrstico
275e1ce Izpis sporočila
3cbe6c7 Dodan glavni program
```

In še primer izpisa povzetka sprememb med posameznimi izvedbami:

```
jure@ris:~> git log --oneline --stat -2
1ef75cf Dodan skok v novo vrstico
 main.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
275e1ce Izpis sporočila
 main.c | 3 +++
 1 file changed, 3 insertions(+)
```

3.4.4 Pregled sprememb: `git diff`

Zgoraj smo videli, da med posameznimi zaporednimi izvedbami moč videti povzetek sprememb. Z naslednjim ukazom pa si lahko te spremembe podrobno ogledamo. Uporaba je sledeča:

```
git diff oznaka...
```

Rezultat ukaza je izpis sprememb med podanimi izvedbami.

Oglejmo si nekaj primerov, pri čemer se spomnimo, da imamo v skladišču tri izvedbe z oznakami (od najstarejše do najnovejše):

3cbe, 275e in 1ef7.

```
jure@ris:~> git diff 275e 1ef7
diff --git a/main.c b/main.c
index b4c1988..281e755 100644
--- a/main.c
+++ b/main.c
@@ -1,6 +1,6 @@
#include <stdio.h>

  int main() {
- printf("Pozdravljen svet!");
+ printf("Pozdravljen svet!\n");
}
```

```
jure@ris:~> git diff
jure@ris:~> git diff 1ef7
jure@ris:~> git diff 275e
...
jure@ris:~> git diff 3cbe
...
jure@ris:~> git diff 275e 3cbe
diff --git a/main.c b/main.c
index 6f5b42a..b4c1988 100644
--- a/main.c
+++ b/main.c
@@ -1,3 +1,6 @@
+#include <stdio.h>
+
int main() {
+ printf("Pozdravljen svet!");
}
jure@ris:~> git diff 3cbe 275e
...
```

3.4.5 Obnavljanje iz skladišča: `git checkout`

3.5 Vejitve

3.5.1 `git branch`

3.5.2 `git branch`

3.6 Oddaljeno skladišče

3.6.1 Git hosting