Git

„The information manager from hell“

Robin Obůrka • robin.oburka@nic.cz

Obsah

VCS obecně

VCS = Version Control System, Systém pro správu verzí

Verzování: způsob uchovávání historie veškerých provedených změn.

- Umožňuje vrátit se:
  - v historii, pokud je něco špatně
  - k zavrženým nápadům
- Obecně umožňuje jednoduše spolupracovat v týmech:
  - nejčastěji dochází ke korektnímu slévání změn
  - případné kolize jsou detekované a uživatel je na ně upozorněn
- Přirozený způsob zálohování práce

Základní jednotkou verzování je **revize = commit**.

VCS obecně

Commit

Commit: jednotlivý, logický celek práce.

Dobrý commit:

- obsahuje jednotlivou, logickou část práce
  - oprava konkrétní chyby
  - jedna nová vlastnost programu
  - funkcionalita, která nejde logicky rozdělit
- obsahuje logickou a srozumitelnou zprávu
  - typicky v angličtině
  - ve smluveném formátu
- je správně umístěn v posloupnosti verzí
- v optimálním případě transformuje projekt mezi funkčními verzemi
  - ne vždy je to reálné — např. počátek vývoje nového projektu
  - usnadňuje hledání kódu, který zanesl chybu

Historie

- Počátek v roce 2005
- Linus Torvalds

  I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.

- Napsán pro potřeby linuxového jádra

Historie

První commit DVCS Git

```
commit e83c5163316f89bfbde7d9ab23ca2e25604af290
Author: Linus Torvalds <torvalds@ppc970.osdl.org>
Date:   Thu Apr 7 15:13:13 2005 -0700

Initial revision of "git", the information manager from hell
```

2 Příprava prostředí

- Získání Gitu
- Konfigurace Gitu

Získání Gitu

- Linux: `[apt-get|yum|zypper] install git`
- Ostatní: `http://git-scm.com/downloads`
- Zdrojové kódy: `https://www.kernel.org/pub/software/scm/git`

Základní konfigurace

Konfigurační soubor uživatele (`$HOME/.gitconfig`):

Nastavení uživatele

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Povolení obarvení výstupu Gitu (od verze 1.8.4 automaticky)

```
git config --global color.ui auto
```

Chování příkazu push

```
git config --global push.default simple
```

Nastavení editoru

```
git config --global core.editor vim
```

Základní konfigurace

Aliasy

```
git config --global alias.st status
git config --global alias.ci commit
git config --global alias.co checkout
git config --global alias.br branch
...
```

Přehledný výpis historie

```
git config --global alias.ll 'log --oneline --graph --all
--decorate'
```

Globální gitignore

```
git config --global core.excludesfile ~/.gitignore
```

.gitignore

Textový soubor, ve kterém je na každém řádku přes masku udáno, co má Git ignorovat. Soubor může být:

- Globální, definovaný v konfiguraci
- V každém repozitáři samostatně; obyčejný soubor s názvem `.gitignore`, který se verzuje společně s projektem

Podobné jako v Bashi, ale:

- / na začátku — platí od kořene pracovní složky repozitáře
- / na konci — uvažuje jen složky
- * funguje, ** má speciální význam
- ! na začátku — negace
- # na začátku — komentář
- \ je escape znak

Vnitřní implementace

Drobný pohled na vnitřní implementaci:

- Revize značeny pomocí SHA1 hashe
- Revize organizovány jako orientovaný graf
- Každá revize má jednoho nebo více rodičů
- Větve jsou ukazatelé na revize
- Máme symbolickou referenci HEAD — aktuální pozice

3 pracovní oblasti

Git má 3 pracovní oblasti:

1. Working directory
2. Staging area
3. Repozitář

Vytvoření lokálního repozitáře

Inicializace prázdného repozitáře

```
git init [DIRECTORY]
```

Zjištění stavu pracovních oblastí

Jak na tom jsme?

Přehled o stavu pracovních oblastí

```
git status
```

Změny v pracovním adresáři

```
git diff
```

Změny připravené k zapsání

```
git diff --cached
```

Vytvoření revize

Přidání souboru / změn do staging area

```
git add PATH
```

Zápis revize

```
git commit
```

Prohlédnutí revize

```
git show [REV]
```
[1]

První možnost opravy omylu:

- `git commit --amend`
- `git commit --amend --no-edit`
- `git commit --amend --reset-author`

[1] Výchozí hodnota je `HEAD`

Změny ve FS

Příkazy pro změny FS

```
git mv SRC_PATH DST_PATH
git rm PATH
git rm --cached PATH
```

Tyto příkazy zároveň přidají do staging area. Funkční jsou i postupy:

1. `mv OLD NEW`
2. `git add NEW`
3. `git rm OLD`

1. `rm PATH`
2. `git rm PATH`

ale jsou méně intuitivní a pracnější...

Procházení historie

Příkaz pro procházení historie

```
git log
```

Zajímavé varianty:

- `git log --oneline`
- `git log --oneline --graph`
- `git log --decorate`
- `git log -p`
- `git log PATH`
- `git log [-i] --grep PATTERN`
- `git log --author=mail@domain.tld`
- `git log --pretty=FMTSTR` (%ae, %an..., changelogy, statistiky)
- `git log -n INT`
- `git log --since=DATE` (také: `--after`, `--until`, `--before`)
- `git blame`

Srovnávání verzí

Příkaz pro srovnání verzí

```
git diff
```

Zajímavé varianty:

- `git diff REV REV`
- `git diff REV..REV`
- `git diff REV REV PATH`
- `git diff PATH1 PATH2`[2]

Již znáte:

- `git diff --cached`

[2] `PATH1`: soubor v repozitáři; `PATH2`: soubor na disku

Označení revizí (`REV`)

- Absolutně
  - Hash (typicky stačí prvních 6 hexa znaků)
  - Název větve / štítku
  - Místo do kterého jsme naposledy přepnuli — `HEAD`
- Relativně (vůči čemukoliv z předchozího)
  - `REV^` — o jedna před `REV`
  - `REV^^` — o dvě před `REV`
  - `REV^^^` — o tři před `REV`
  - `REV~NUM` — o `NUM` před `REV`
  - Lze kombinovat: `REV^^~2^`

Git a větve

Větve

- Nic nestojí (jednotky KB)
- „Větvěte často!“

Základem spousty workflows

- Feature branch
- Debug / Hotfix
- Integrační větve

Vždy existuje minimálně jedna větev!

- Implicitně větev master
- Teoreticky se může jmenovat libovolně a větev master nemusí vůbec existovat
- Je žádoucí dodržovat konvence

Základy práce s větvemi

Vytváření větví

Vytvoření nové větve

```
git branch NAME [REV]
```

- Pokud nebudeme specifikovat `REV`, tak se použije `HEAD`.
- Názvy větví je možné i prefixovat.

Vylistování existujících větví

```
git branch [-v]
```

Základy práce s větvemi

Přepínání větví

Přepnutí do větve

```
git checkout NAME
```

Checkout obecně: Aktualizuje soubory v pracovním adresáři tak, aby odpovídaly dané revizi

Příkaz `git checkout`

```
git checkout (REV|NAME) [PATH]
```

Často používaná alternativa:

Vytvoření nové větve včetně checkoutu do ní

```
git checkout -b NAME [REV]
```

Základy práce s větvemi

Mazání větví

Příkaz pro smazání větve

```
git branch -d NAME
```

- Nelze mazat aktuální větev
- Nelze mazat větev s nezahrnutými změnami (ale lze vynutit)

Které větve mohu smazat?

```
git branch --merged
git branch --no-merged
```

Příkaz pro přejmenování větve

```
git branch -m [OLDNAME] NEWNAME
```

Slévání změn

Počáteční verze

```
#include <stdio.h>

int main(int argc, char **argv) {
    char *str = "World";

    printf("Hello, %s!\n", str);
    return 0;
}
```

Slévání změn

Úpravy vývojáře Mr. Blue

```
#include <stdio.h>

void print_hello(char *str) {
    printf("Hello, %s!\n", str);
}

int main(int argc, char **argv) {
    char *str = "World";

    print_hello(str);
    return 0;
}
```

Slévání změn

Úpravy vývojáře Mr. Red

```
#include <stdio.h>

int main(int argc, char **argv) {
    char *str = "World";
    if (argc == 2) {
        str = argv[1];
    }

    printf("Hello, %s!\n", str);
    return 0;
}
```

Slévání změn

Stav po „slití“ změn od obou vývojářů

```
#include <stdio.h>

void print_hello(char *str) {
    printf("Hello, %s!\n", str);
}

int main(int argc, char **argv) {
    char *str = "World";
    if (argc == 2) {
        str = argv[1];
    }

    print_hello(str);
    return 0;
}
```

Merge větví

Příkaz pro merge větví

```
git merge NAME
```

- Provede merge větve `NAME` k aktuální větvi
- Existují 3 možné průběhy operace merge:
  1. FF (Přetočení vpřed)
  2. Automatický merge bez kolizí
  3. Automatický merge s kolizí

Elegantní alternativou k příkazu `git merge` je `git rebase`...

Reset

Odebrání ze stage area

```
git reset HEAD PATH
```

Provedené změny vrátí zpět do working directory.

Změna ukazatele větve

```
git reset --hard REV
```

Změní ukazatel větve na revizi `REV` a vrátí working directory do této revize.

Změna ukazatele se zachováním změn

```
git reset --soft REV
```

Změní ukazatel větve na revizi `REV` a všechny změny vrátí do stage area.

Tagy

Prosté značky

```
git tag NAME [REV]
```

- Pokud nebudeme specifikovat `REV`, tak se použije `HEAD`
- Implementačně se jedná o větev, která se nepohybuje
- Do vzdáleného repozitáře se nedostanou automaticky (`git push origin --tags`)

Anotované značky

```
git tag -a NAME [REV]
```

Podepsané značky (vytvoření a verifikace)

```
git tag -s NAME [REV]
git tag -v NAME
```

Oprava omylů

Zahození lokálních změn

```
git checkout -- PATH
```

Nevratná operace! — pro zbrklé: v IDE/editoru se hodí vypnout automatické načítání změn z disku ;-)

Zahození reference, indexu, ...

`git reset` (viz dříve)

Úprava posledního commitu

```
git commit --amend
```

Oprava omylů

Modelové situace

„V commitu nějaká změna chybí“

```
git add ...
git commit --amend
```

„V commitu nějaká změna přebývá“
„Chci rozdělit commit na dva samostatné“

```
git reset --soft HEAD^
git reset HEAD PATH
git add ...; git commit
git add ...; git commit
```

Pokročilá příprava revizí

Velmi důležité — aby nás verzování neobtěžovalo

- Nemusím myslet na to, jak svoji práci budu dělit do commitů, ale v klidu pracovat
- Nemusím dělat špatné commity, jen proto, že jsem se chtěl soustředit na práci

Editace patche v editoru

```
git add -e [PATH]
```

Veškerou práci zobrazí jako patch v editoru

Interaktivní přidání patche

```
git add -p [PATH]
```

Postupně ukazuje změněné kusy kódu a umožňuje je zahrnout, odmítnout, editovat, dále rozdělit...

Pokročilá příprava revizí

Pomocí editace patche je možné rozdělit i logicky různé změny na jednom řádku do více commitů (např. změna obsahu řádku a změna stylu řádku).

Do stage area se dá chytře nejen přidávat, ale i z ní odebírat:

Interaktivní odebrání ze stage area

```
git reset -p [PATH]
```

Stejnou metodu můžeme aplikovat na zahození lokálních změn:

Interaktivní zahození lokálních změn

```
git checkout -p [PATH]
```

Pozor: opět se jedná o nevratnou operaci.

Vzdálený repozitář

Kde vzít vzdálený repozitář:

- Hostované repozitáře
  - Cizí poskytovatel: github.com, bitbucket.org
  - Nějaká vám blízká organizace: gitlab.fit.cvut.cz
- Vlastní server

Trocha teorie

- Vzdálený repozitář je tzv. remote
- Speciální roli má remote s názvem origin
- Origin je výchozí repozitář pro mnoho operací (pull, push, fetch)
- Jako origin je automaticky označený remote, ze kterého jsme klonovali

Clone

Naklonování vzáleného repozitáře

```
git clone [OPTIONS] URL [DIRECTORY]
```

Push

Plné znění příkazu `git push`

```
git push [OPTIONS] [-u] [REMOTE [BRANCH[:REMOTE_BRANCH]]]
```

Jak příkaz funguje:

- Odešle vaše změny na remote
- Výchozí remote je origin
- Výchozí (lokální) větev je ta aktuální
- Výchozí mapování local:remote je podle shodného jména
- Parametr `[-u]` nastavuje upstream větve

Co to znamená:

- `git push` — odešle změny z aktuální větve na origin
- `git push -u` — odešle změny z aktuální větve na origin a nastaví upstream pro aktuální větev na origin (Až od verze 2. Jinak `git push -u REMOTE BRANCH`.)
- `git push production master:deploy` — odešle změny na production do větve deploy, která odpovídá aktuální větvi master

Pull

Plné znění příkazu `git pull`

```
git pull [OPTIONS] [REMOTE [REV]]
```

Jak příkaz pracuje:

- Vyzvedává vzdálené změny (`git fetch`)
- Provádí merge lokální a odpovídající vzdálené větve
- Většina voleb odpovídá volbám příkazu `git merge`

Publikování existujícího obsahu

Přidání existujícího remote

```
git remote add NAME URL
```

Publikování existujícího obsahu

```
git remote add origin URL
git push -u origin master:master
```

Vzdálené větve

Vytvoření vzdálené větve

```
git push -u REMOTE BRANCH
```

Zobrazení vzdálených větví

```
git branch -r
```

Příkaz `git branch` ukazuje pouze lokální větve. Vzdálené je nejprve nutné začít trackovat.
Lokální (trackované) větve zůstávají i když je vzdálená větev smazaná.

Prořezání smazaných větví

```
git remote prune REMOTE
```

Smazání vzdálené větve

Vzdálené větve

Vytvoření vzdálené větve

```
git push -u REMOTE BRANCH
```

Zobrazení vzdálených větví

```
git branch -r
```

Příkaz `git branch` ukazuje pouze lokální větve. Vzdálené je nejprve nutné začít trackovat.
Lokální (trackované) větve zůstávají i když je vzdálená větev smazaná.

Prořezání smazaných větví

```
git remote prune REMOTE
```

Smazání vzdálené větve

```
git push REMOTE :BRANCH
```

Děkuji za pozornost

Robin Obůrka • robin.oburka@nic.cz