Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Grafické uživatelské rozhraní pro systém správy verzí Git

2021

Jaroslav Večeřa

Vedoucí práce: Mgr. Radek Janoštík

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Jaroslav Večeřa
Název práce:	Grafické uživatelské rozhraní pro systém správy verzí Git
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Radek Janoštík
Počet stran:	17
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Jaroslav Večeřa
Title:	Graphical user interface for version control system Git
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Radek Janoštík
Page count:	17
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Grafické rozhraní pro Windows, které zprostředková přehlednou a intuitivní práci se základními funkcemi systému Git, a to převážně pomocí grafu.

Synopsis

Windows graphical user interface that allows intuitive git workflow using graph representation.

Klíčová slova: git; verzování; graf; grafické rozhraní

Keywords: git; version control; graphical interface

Děkuji, děkuji, děkuji.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Při vývoji programů, zvláště pak těch netriviálních, je často třeba dělat změny nebo nové verze. Protože však není možné vytvořit program bez chyb, objevuje se také potřeba vracet se k libovolným starším verzím a zakládat na nich nové, či dokonce vyvíjet více verzí zároveň v případě skupiny lidí. Tato činnost lze samozřejmě provádět ručně, zabírá to ale čas a zvyšuje riziko chyby. Může dojít ke změně souboru nebo jeho smazání na špatném místě. Přece jen udržovat si v uložišti desítky záloh nebo obměn dat a spravovat je ručně vyžaduje podrobný popis jednotlivých verzí, který časem musí být značně nepřehledný. Z tohoto důvodu byly vytvořeny takzvané verzovací systémy.

Systém správy verzí (dále SSV) je zpravidla softwarový nástroj, umožňující spravovat verze projektu částečně automaticky (nebo alespoň přehledně a jednoduše). Přitom daný projekt nemusí být zdrojovým kódem v nějakém programovacím jazyce, může se jednat vlastně o libovolná data. Vracet zpět provedené změny, nebo pracovat ve skupině lidí může být užitečné například i grafikům, střihačům videí, spisovatelům a podobně. Systém uchovává jak soubory samotné, tak různé informace související se správou verzí. To se samozřejmě napříč konkrétními systémy liší, obvykle je ale k dispozici:

- Popis změny (ručně zadaný)
- Čas změny
- Autor změny a jeho kontaktní údaje

Tyto údaje jsou zejména užitečné v případě týmu lidí pracujících na společném projektu, historicky však nejprve vznikla skupina takzvaných lokálních SSV.

1.1 Lokální SSV

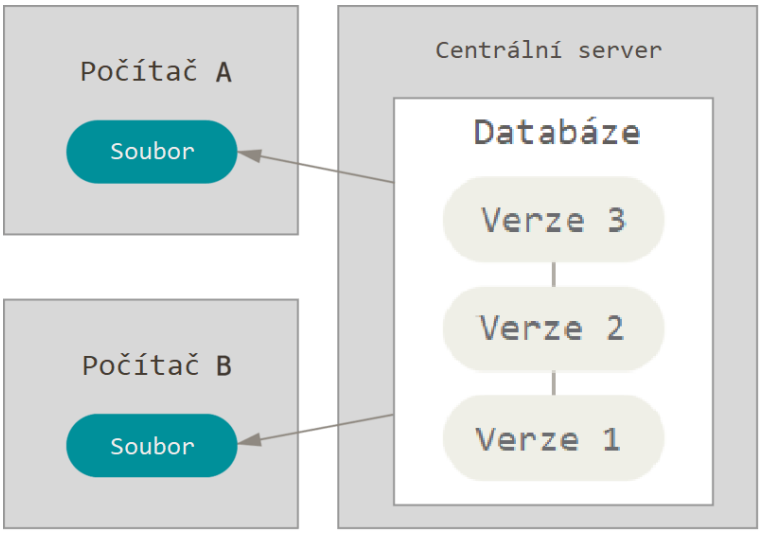
Tyto systémy se soustředily na práci jednotlivce, celý projekt byl ukládán na místním disku a nebyl nikde sdílen. Konkrétním zástupcem je například RCS (Revision Control System). RCS si uchovává poslední podobu daného souboru spolu se zpětnými rozdíly. Aplikací těchto rozdílů (delt) na soubor lze rekonstruovat některou jeho předchozí verzi.

Nevýhoda lokálního SSV je, že projekt je umístěn pouze na jednom zařízení, a při chybě disku tak hrozí ztráta dat. Další nevýhodou je nemožnost projekty pohodlně sdílet po síti.

1.2 Centralizované SSV

Centralizované systémy (CSSV) [1] jsou historicky dalším vývojovým stupněm SSV. Představují opačný extrém k lokálním SSV, na rozdíl od nich totiž většinu souborů a práci přesouvají od koncového uživatele na jeden společný centrální

server, ten je skrze síť dostupný odkudkoliv na světě. Metodu zachycuje obrázek 1 [2].



Obrázek 1: Centrálizovaný systém správy verzí

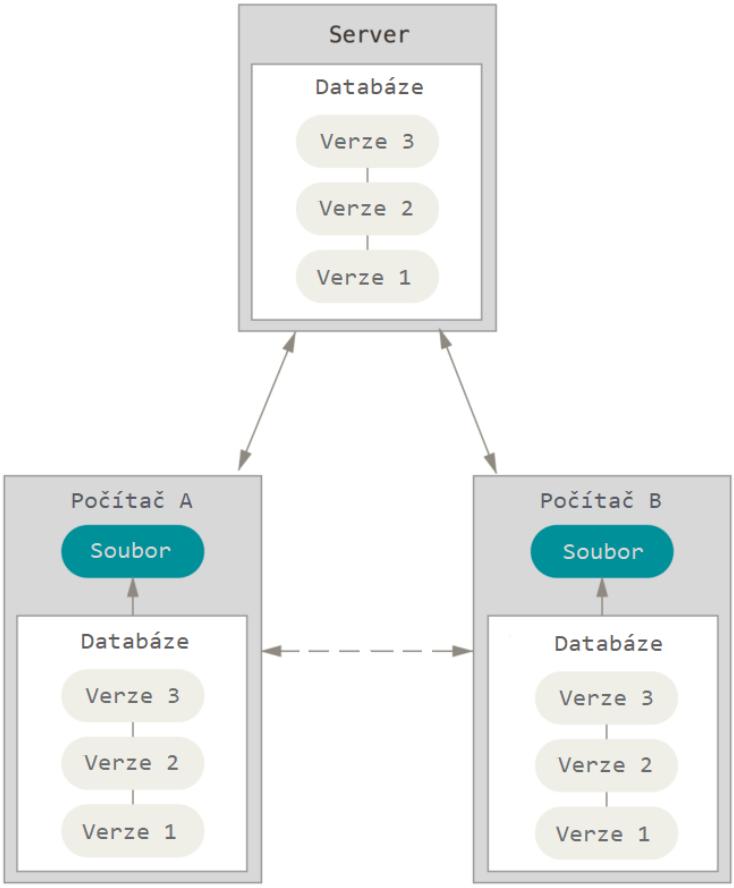
Od chvíle, kdy uživatel A udělá na souboru S nějaké změny a nasdílí je do společné databáze, už žádný další uživatel nemá aktuální verzi souboru S. Pro obdržení aktuální verze musí svoji kopii každý opět aktualizovat. V případě, že další uživatel provedl jiné změny na témže souboru, systém se je pokusí sloučit dohromady. Uživatelé se tedy nemusí starat o případy, ve kterých nedojde k závažnému konfliktu. Pokud však uživatel A upravil stejný řádek souboru S, jako uživatel B, ale jinak, je třeba se o vyřešení konfliktu postarat ručně výběrem správné verze, případně spojením obou změn.

Pro snížení počtu konfliktů je v CSSV dostupná funkce větvení. Umožňuje vytvářet historii změn s jinší než lineární strukturou. Větvení se častou používá pro implementaci ucelené funkcionality programu, nebo pro experimentální záležitosti, které nemusejí být po svém ukončení začleněny do projektu. Uživatel může pracovat na verzích větve aniž by ovlivňovaly verze větve jiného uživatele.

Výhodou CSSV je například šetření místa jednotlivých uživatelů. Ti na svých zařízeních mají fyzickou kopii pouze aktuální verze projektu, zbytek historie je uložen na serveru. To však přináší nemalá rizika v případě výpadku. Pokud uživatel není schopen připojení k síti, nemůže na projektu pracovat, jelikož veškerá práce se systémem vyžaduje síťové připojení. Pokud dokonce dojde k porušení této společné databáze, veškerá historie dat je nenávratně ztracena, stejně jako v Lokálním SSV se totiž nachází pouze na jednom místě. Drobnou výhodou zůstává, že alespoň aktuální verze se nachází na více zařízeních.

1.3 Distribuované SSV

Distribuované systémy (DSSV) [1] se vyvinuly po centralizovaných a představují jakýsi kompromis obou předchozích systémů. DSSV se snaží těžit z výhod obou metod. Projekt lze snadno sdílet pomocí sítě. Kromě uživatelů obsahuje servery, na kterých se nachází tzv. vzdálené repozitáře. Tyto repozitáře plní stejnou funkci jako v CSSV, nejedná se však o jedinou kopii projektu. Každý uživatel, který s ním pracuje, vlastní úplnou kopii celé historie. Jednotlivé repozitáře uživatelů a serverů se mezi sebou aktualizují pomocí k tomu určených příkazů. Tyto příkazy se často nazývají push (pro pokus včlenit změny do zvoleného vzdáleného repozitáře), fetch (pro stažení dat ze vzdáleného repozitáře) a pull (pro včlenění dat ze vzdáleného repozitáře).



Obrázek 2: Distribuovaný systém správy verzí

Jak je v obrázku 2 [2] naznačeno přerušovaným spojení mezi uživateli A a B, přímé sdílení projektu sice DSSV umožňuje, není však tolik využívané. Mnohem častěji sdílí uživatelé práci se společným serverem, který tak hraje roli jakéhosi pasivního uživatele, se kterým ostatní komunikují.

Příklady takových nástrojů jsou Mercurial, Bazaar, nebo Git, kterého se tato práce týká.

2 Git

Git [3] je distribuovaný SSV, který na rozdíl od ostatních SSV uchovává historii takovým způsobem, že většina operací nad soubory je výrazně rychlejší. V článku [**gitreference-history**] je popsána historie vzniku Gitu.

Většina verzovacích systémů má uložený soubor a pro každou jeho verzi dopředné, či zpětné rozdíly. Pomocí skládání těchto rozdílů lze znovu zhotovit libovolnou verzi souboru. Hlavní výhoda tohoto přístupu spočívá v ušetřeném místě, oproti ukládání celé kopie se totiž ušetří části, které se mezi jednotlivými verzemi nezměnily. Má to však dopad na rychlost opětovného sestrojení konkrétních verzí.

Git naproti tomu uchovává pro jednotlivé verze celé kopie (nazývají se snapchoty). Rychlost sestrojení souborů v historii tak není ovlivněna množstvím verzí, které od té doby byly zhotoveny. Git samozřejmě ukládání optimalizuje, a to tak, že pokud nejsou v souboru provedené změny, místo nového snapshotu se uloží odkaz na starý. Celý repozitář také podléhá bezeztrátové kompresi.

Verze projektu také budeme nazývat revize. Každá revize, kromě počáteční, má jeden nebo více předků. Jeden v případě prostého vytvoření další verze, více v případě slévání změn z více verzí (merge). Celá historie se tak dá reprezentovat jako orientovaný, souvislý a acyklický graf s uzly vyjadřujícími revize a hranami vyjadřujícími vztah rodič – potomek. Vytvořit aplikaci reprezentující tímto způsobem historii vytvořenou gitem je také hlavní cíl této práce.

Jednotlivé větve pak git ukládá vnitřně pouze jako ukazatele na poslední revizi větve. Při vytváření nových verzí se tyto ukazatele posunují na novější commit. Žádná data o tom, v jaké větvi byla revize původně vytvořena, nejsou k dispozici a často se nedají nijak dohledat. Tato informace bude později důležitá při popisu heuristiky rozmístění uzlů v grafu na straně 12

Poté ještě v repozitáři existuje speciální ukazatel HEAD, který ukazuje na větev, či přímo revizi, které jsou aktuálně prohlíženy.

Následující seznam popisuje základní funkce pro práci s Gitem.

Commit vytvoří novou verzi, přitom se na ni přesune ukazatel větve, na kterou ukazuje HEAD, případně se posune HEAD, pokud ukazuje přímo na revizi.

Branch Vytvoří nový ukazatel větve na aktuální verzi.

Checkout znovu sestrojí verzi předanou argumentem. Buď formou větve, kdy HEAD začne odkazovat na onu větev, nebo formou revize přímo, potom HEAD odkazuje na revizi a repozitář se nachází v experimentálním módu.

Stash v závislosti na argumentu ukládají, aplikují a mažou provedené změny od poslední verze na strukturu zásobníkového charakteru.

Merge spojuje vybrané větve do jedné. V případě, že nelze konflikty automaticky vyřešit, je o to uživatel požádán.

Rebase oproti merge manipuluje s historií. Přeskládá revize aktuální větve (B_1) jako by se od dané větve (B_2) oddělovaly až na konci B_2.

Diff je nástroj pro vytvoření rozdílů v souborech, nebo celých verzích.

Log ukazuje strukturu historie.

Fetch stáhne historii ze vzdáleného repozitáře.

Pull provede fetch a následně merge.

Push naopak začlení změny do vzdáleného repozitáře.

2.1 Základní postupy práce v Gitu

Git poskytuje velkou volnost ve způsobu správy větví a to jak lokálně [4] , tak v práci se vzdálenými repozitáři [5].

2.1.1 Postupy větvení

Dlouhodobé větve Při práci tímto způsobem obvykle repozitář obsahuje větve tří úrovní. První úroveň tvoří hlavní větev (často s názvem *master*, nebo *main*), která obsahuje pouze dobře otestované revize připravené k publikaci.

Vedle toho bývá k dispozici větev s názvem jako je *next*, nebo *develop*, která obsahuje méně stabilní kód, který není ucelený, nebo teprve čeká na otestování. Z této větve jsou revize podle potřeby slučovány do hlavní větve.

Poslední úroveň tvoří větve pro jednotlivé funkcionality. Tyto větve slouží k oddělení funkcionalit, které jsou v současné době ve vývoji. Z nich je práce po dokončení slučována do *develop* větve.

Krátkodobé větve Tento způsob práce (někdy nazývaný *topic branch*) není tak striktní ve způsobu slučování větví. Dovoluje slučování starších větví do novějších i naopak a tím vzniká tendence udržovat větve krátkodobější a s menším počtem revizí.

Dá se říct, že se jedná o odlehčenou verzi metody dlouhodobých větví, která obsahuje pouze nejnižší úroveň stability. Pro tyto vlastnosti je postup vhodný spíše u menších projektů.

2.1.2 Distribuovaná práce s větvemi

Centralizovaný postup Tento postup je hojně využíván hlavně pro svoji jednoduchost. Také je vhodný při přechodu z centralizovaného SSV pro jejich podobnost. Centralizovaný postup je vhodný pro týmy, ve kterých nehraje velkou roli hierarchie vývojářů, popřípadě nejsou příliš početné.

Prostředkem pro sdílení projektu je jediný vzdálený repozitář, ze kterého/který přispěvatelé aktualizují. Uživatelé, kteří chtějí na projektu pracovat, provedou operaci merge, či clone. Pokud naopak chtějí svoji práci sdílet do vzdáleného

repozitáře, provedou push. V případě, že jiný uživatel mezitím sdílel svoji práci, nemá daný uživatel aktuální verzi a musí nejprve včlenit historii z centrální databáze do své, poté až provést push. Přitom samozřejmě může nastat, sice nepravděpodobná, ale stále možná situace, kdy, než uživatel stihl včlenit změny a provést push, opět někdo aktualizoval centrální databázi. Proces je potom třeba opakovat.

Postup s integračním manažerem Tento postup lépe využívá možnosti DSSV a vyžaduje jeden centrální vzdálený repozitář a dále jeden vzdálený repozitář pro každého běžného přispěvatele.

K centrálnímu repozitáři mají opět přístup všichni, ale právo zápisu má jen správce (integrační manažer). Ostatní smí zapisovat pouze do soukromých vzdálených repozitářů.

Pokud chce uživatel aktualizovat svůj repozitář, jednoduše provede pull, či clone, jako u předchozího postupu. V případě sdílení je ale situace odlišná. Uživatel odešle data do svého vzdáleného repozitáře a uvědomí o změnách správce. Ten včlení změny uživatelova vzdáleného repozitáře do svého lokálního repozitáře a poté je sdílí do centrálního vzdáleného repozitáře. Tam k datům mají opět přístup všichni ostatní.

Postup s diktátorem a poručíky Předchozí postup lze ještě vylepšit přidáním více správců a rozdělením jejich rolí do heirarchie: jeden diktátor a ostatní poručíci. Tento postup najde uplatnění spíše u projektů extrémních rozměrů, jako například vývoj Linuxového jádra [6]

Běžní vývojáři svojí práci včleňují na vrchol diktátorovi větve *master* pomocí příkazu rebase. Poručíci potom včlení větve vývojářů do svých větví *master*. Diktátor včlení větve poručíků do své větve *master* a zpřístupní ji do centrálního repozitáře ostatním.

2.2 Algoritmus rozmístění uzlů v grafu

Závěr

Závěr práce v „českém“ jazyce.

Conclusions

Thesis conclusions in “English”.

A Obsah přiloženého CD/DVD

`bin/`
: Instalátor

Navíc CD/DVD obsahuje:

`literature/`
: Vybrané položky bibliografie, příp. jiná užitečná literatura vztahující se k práci.

Literatura

[1] OTTE, Stefan. Version Control Systems. 2009.

[2] SCOTT, Chacon. *Getting Started - About Version Control*. 2014. Dostupný z: ⟨https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control⟩.

[3] CHACON, Scott; STRAUB, Ben. *Pro git*. 2014.

[4] SCOTT, Chacon. *Git Branching - Branching Workflows*. 2014. Dostupný z: ⟨https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows⟩.

[5] SCOTT, Chacon. *Distributed Git - Distributed Workflows*. 2014. Dostupný z: ⟨https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows⟩.

[6] TORWALDS, Linus. *linux*. 2015.