Použití Subversion pro verzování TEXových dokumentů

Michal Růžička

Článek představuje výhody použití systému pro správu verzí Subversion při vývoji TEXových dokumentů a balíků maker. Popsány jsou základní principy Subversion, následuje předvedení praktického použití krok za krokem.

1. Úvod

Vývoj TEXových dokumentů, zejména balíků maker, je poměrně složitá činnost. Části dokumentů, především makra, jsou pak jejich autory často používána a dle potřeby živelně upravována velmi dlouhou dobu. I když psaní TEXových maker bezesporu je programování, zdá se, že mnoho autorů TEXových dokumentů, na rozdíl od „běžných programátorů“, zatím neobjevilo sílu a užitečnost nástrojů pro správu verzí. Nezřídka se pak stává, že ani autoři sami nejsou po nějaké době schopni znovu vysázet své staré dokumenty, neboť již nemají potřebné verze svých makrobalíků. Stejně tak mají problémy orientovat se ve vlastním kódu nebo dohledat zdroj chyby, která se náhle objevila zdánlivě odnikud.

Tento článek je proto koncipován jako kurz základů použití systému pro správu verzí Subversion[1] pro autory TEXových dokumentů a balíků maker, kteří nemají se Subversion žádné (nebo téměř žádné) zkušenosti. Zároveň by chtěl ukázat, proč je použití systémů pro správu verzí pro uživatele TEXu užitečné, a to ať už pracují na svých projektech jako jediní autoři, anebo potřebují na vývoji spolupracovat v týmu.

2. Co je Subversion

Subversion (`http://subversion.apache.org/`) je open-source (Apache/BSD-style licence) systém pro správu verzí. Jedná se o multiplatformní systém dostupný pro GNU/Linux, *BSD, Sun Solaris, Apple Mac OS X, IBM AIX, HP-UX i MS Windows.

Subversion je server–klient verzovací systém[2] (obdobně jako například CVS, jehož je Subversion nástupcem). Všechna data včetně celé historie verzí (tak zvaná

[1] Subversion se používá například při vývoji TEXové distribuce TEX Live. (Vizte `http://www.tug.org/texlive/svn/`.)

[2] Jiným typem systémů pro správu verzí jsou distribuované systémy (například GNU arch,

repository) jsou uložena na serveru, ke kterému se připojují uživatelé a potřebná data si odtud stahují (checkout) do své soukromé lokální pracovní kopie (working copy), kterou modifikují. Když jsou se změnami spokojeni, odešlou je na server (commit), a tím je zpřístupní ostatním uživatelům. Zároveň tak vytvoří pevný bod historie (revizi), který je v repository navždy zaznamenán a ke kterému je možné se kdykoli vrátit.

3. Proč použít Subversion

Existují asi dva hlavní důvody, proč začít používat Subversion, respektive obecně systémy pro správu verzí:

1. umožnění souběžné spolupráce více vývojářů na jednom projektu,
2. trvalé zachování předchozích verzí projektu s možností kdykoliv se k nim vrátit.

Pod pojmem projekt přitom máme na mysli libovolnou strukturu podadresářů a souborů uloženou na disku. Je přitom jedno, zda se jedná o adresář s „projektem“ nějakého IDE, adresář se zdrojovými texty v jazyce C upravovanými editorem Vim a překladem řízeným Makefile, TEXový dokument, zálohu konfiguračních souborů nebo sadu obrázků. Subversion pracuje na souborové úrovni. Z principu fungování Subversion jsou však z prostorového a praktického hlediska vhodnější textové soubory než soubory binární (jako revize se například fyzicky ukládají jen rozdíly oproti souborům z předchozí revize, což se u textových souborů provádí velmi jednoduše). V repository ale bez nejmenších problémů *je možné* ukládat i binární soubory.

Nasazení Subversion z důvodu č. 1 má smysl pouze v okamžiku, kdy na projektu spolupracuje více lidí. Pak je však obrovským přínosem, protože nejen zachovává historické verze projektu, ale umožňuje souběžnou práci více vývojářů nezávisle na sobě. Pokud různí vývojáři budou pracovat výhradně na různých souborech, není problém ani bez verzovacího systému. Častější případ však je, že více vývojářů může ve stejné chvíli pracovat na stejném souboru. V tomto případě Subversion umožňuje řešit slučování nezávisle na sobě provedených změn. Pokud je to jen trochu možné, jsou změny sloučeny automaticky. Pokud více vývojářů modifikovalo úplně stejnou část souboru, pak je nutné konflikt vyřešit zásahem člověka. Subversion ale v tomto případě alespoň poskytne nástroje k tomu, aby to bylo možné provést poměrně pohodlně.

Z důvodu č. 2 má smysl Subversion nasadit i na projekty jediného člověka. I když je Subversion server–klient systém, je možné používat ho i lokálně, kde serverová část odpadá a Subversion klient pracuje s lokálně uloženou reposi-

Bazaar, Git, Mercurial a další). V jejich případě každý vývojář pracuje s vlastní lokální kopií celé historie vývoje a sdílení změn mezi jednotlivými vývojáři probíhá explicitně v samostatném kroku.

tory přímo bez asistence serveru jako mezičlánku. Přístup k historickým verzím projektu přitom neznamená jen možnost prostého zkopírování starší verze dokumentu nebo makrobalíku. Jednotlivé změny všech souborů jsou okomentovány. I po dlouhé době je tedy možné sledovat vývoj maker a uvědomit si, proč je to které makro napsáno tak, jak je. Stejně tak je možné podrobně zkoumat změny mezi jednotlivými verzemi – dozvíme se tak, jak byla konkrétní funkčnost dosažena, jaké problémy se v průběhu implementace objevily a jak byly řešeny.

Velkou pomocí je Subversion také při ladění maker. Nemusíme se bát naše dokumenty razantně upravovat. Pokud se zvolený postup ukáže být slepou uličkou, je snadné všechny změny navrátit do funkčního stavu. Stejně tak snadno zjistíme, jaké změny jsme vlastně v dokumentu právě udělali oproti poslední funkční verzi, což jinak může být při intenzivním zkoušení a opětovném rušení různých řešení někdy obtížné zjistit. Pokud po delší době narazíme na chybu ve funkcionalitě, kterou jsme delší dobu netestovali (protože jsme byli mylně přesvědčeni, že na ni naše úpravy nemohly mít žádný vliv), bylo by bez systému pro správu verzí obtížné zjistit, kdy a jak k chybě došlo. Se systémem pro správu verzí v nejhorším případě budeme postupovat po již chybných verzích zpět v čase, až narazíme na verzi funkční. Prozkoumáním rozdílu mezi poslední funkční verzí a verzí chybnou pak příčinu snadno odhalíme.

4. Jak Subversion pracuje

4.1. Subversion repository

Jak bylo uvedeno výše, Subversion je server–klient systém. Všechna data jsou uložena na serveru v tak zvané repository. Repository tvoří jeden adresář obsahující několik málo konfiguračních souborů a dále vlastní úložiště dat, kam je při každém commitu trvale uložena nová revize projektu popisující podobu všech jeho souborů. Úložiště může být realizováno buď jako Berkeley DB[3] (což je již méně doporučovaný způsob), nebo jako tak zvaný FSFS, což je výchozí varianta, a nejedná se o nic víc než o sadu normálních souborů uložených běžným způsobem v souborovém systému.

Celkově tedy Subversion repository není nic tajemného (obzvláště v případě použití FSFS, kdy odpadají potenciální problémy s embedded databází). Je to jen adresář s několika podadresáři a soubory, který nemusí uživatele zajímat vůbec (většinou k němu ani nemá přímý přístup), a administrátor repository zde bude potřebovat nastavit jen přístupová práva (a to jen v případě určitého typu přístupu uživatelů k repository), případně tak zvané háčky.

Přístupová práva se nastavují editací velmi jednoduchých textových konfiguračních souborů. Háčky jsou realizovány spustitelnými programy – v unixových

[3]Berkeley DB (`http://www.oracle.com/database/berkeley-db/db/`) je open-source softwarová knihovna, která poskytuje jednoduchou databázi určenou k vestavění do jiných programů.

operačních systémech například shell skripty. Kostry těchto skriptů pro všechny háčky jsou automaticky vytvořeny spolu s repository, takže je v případě zájmu stačí jen upravit. Jméno spustitelného souboru určuje, při kterém typu akce bude tento program volán. Při provádění nějaké akce (například při commitu) Subversion spustí akci odpovídající skript (pokud existuje) a jako argumenty mu předá informace o dané akci. Dle návratové hodnoty programu pak akci buď povolí, anebo zamítne. Administrátor tak má možnost napsat si například bash skript, který před commitem z předaných argumentů zkontroluje, zda byl uživatelem zadán komentář popisující provedenou změnu a commity bez komentáře zamítne – to znamená ukončí skript s nenulovou návratovou hodnotou.

Podrobnější (a vždy aktuální) informace o formátu repository, konfiguraci přístupových práv a háčků naleznete v Subversion book (`http://svnbook.red-bean.com/`), kterou si můžete buď stáhnout, anebo přímo z webu prohlížet její HTML podobu. Subversion book se dá považovat za jednu z velkých předností systému Subversion, neboť se jedná o výborně napsanou a průběžně aktualizovanou dokumentaci, která je psána jednoduchou a čtivou angličtinou se spoustou příkladů. Na tuto vynikající knihu se v dalším textu odkážeme ještě několikrát.

4.2. Způsob přístupu k repository

V oddílu 4.1 jsme si řekli, jak vypadá Subversion repository. K této repository se přistupuje Subversion klientem. Konkrétní způsob přístupu se mírně liší podle toho, kde je repository umístěna.

4.2.1. Přímý přístup

Úplně nejjednodušším případem je, že jste jediným uživatelem repository a používáte ji čistě za účelem efektivního uchovávání historických verzí projektu. V tomto případě není třeba využívat služeb žádného serveru. Repository si jednoduše vytvoříme na libovolném místě na disku, kam máme právo zápisu (a ostatní sem přístup nemají, typicky tedy někde v domovském adresáři). Subversion klient pak pracuje přímo se soubory repository. Pro uživatele je asi nejdůležitější, že k zadání adresy repository se používá schéma `file://`. (Ukázku praktického použití naleznete v oddílu 5.)

Z toho také plyne, že tento způsob je nevhodný v případě spolupráce více lidí na jednom projektu, i kdyby byla repository umístěna ve sdíleném adresáři, kam mají všichni uživatelé právo zápisu. Jednotliví uživatelé mohou repository poškodit (ať už omylem, anebo záměrně), problém může nastat také s přístupovými právy. Hrozí i další problémy. Používat přímý přístup k repository více uživateli je *důrazně nedoporučeno.*

Přímý přístup k repository je jediný případ, kdy musí být adresář s repository umístěn na lokálním disku (pokud pomineme možnost, že repository je umístěna

na síťovém souborovém systému, který máme na svůj stroj připojen ze vzdáleného serveru). Všechny ostatní dále uvedené možnosti přístupu už jsou plně využitelné nejen lokálně, ale i přes síť.

4.2.2. Server `svnserve`

Pokud má na projektu pracovat více lidí, musíme již nasadit Subversion server. Součástí distribučního balíčku Subversion je i samostatný program `svnserve`, který může sloužit jako Subversion server.

`svnserve` může být spuštěn jako démon,[4] případně může být spouštěn přes internet super server `inetd`. Program `svnserve` běžící pod zvláštním uživatelem (typicky se jmenuje například `svn`) má pak jako jediný přímý přístup k souborům repository (přístupová práva k ní má nastaven jen uživatel `svn`). Všichni klienti se se svými žádostmi obracejí na něj. Používají k tomu schéma `svn://`.

Výhodou je, že v tomto případě již máme dobrou kontrolu nad tím, co se s repository děje. Přímo k souborům repository přistupuje pouze Subversion server. Právě v tomto případě se může aplikovat nastavení přístupových práv v konfiguračních souborech umístěných v adresáři repository, o kterých jsme se zmínili výše. Přístupová práva (nic/čtení/zápis) je možné definovat nejen pro celou repository, ale i na úrovni jednotlivých podadresářů. V tomto případě také není nutné vytvářet žádné systémové uživatelské účty. V konfiguraci repository se vytvoří vlastní uživatelé nezávisle na systému.

Nevýhodou je, že hesla uživatelů jsou v konfiguraci repository uvedena v otevřené nijak nechráněné podobě. Stejně tak není šifrován ani síťový provoz mezi klientem a serverem.[5] Na druhou stranu alespoň hesla sítí neputují, protože se využívá systému výzva odpověď (aktuálně se používá CRAM-MD5).

4.2.3. Server `svnserve` s SSH tunelem

Jedná se o kombinaci přímého přístupu a `svnserve` serveru. K přístupu k repository Subversion klientem se používá schéma `svn+ssh://`.

Uživatel se na stroj přihlásí pomocí svého běžného systémového SSH účtu. Je tedy autentizován přes SSH, jako kdyby prováděl vzdálené přihlašování. SSH klient ale dále spustí vlastní dočasnou instanci programu `svnserve`, který pracuje s repository. Díky tomu ale pochopitelně `svnserve` *běží s právy daného uživatele, nikoli pod uživatelem* ***`svn`***. V důsledku se tedy jedná o obdobu přímého přístupu k repository (schéma `file://`). Uživatel stejně musí mít právo zápisu přímo

[4] Démon je unixové označení programu běžícího na pozadí jako služba operačního systému.

[5] Subversion od verze 1.5 umožňuje zkompilovat standardně dodávaný server `svnserve` a řádkového klienta `svn` s podporou knihovny Cyrus Simple Authentication and Security Layer (`http://asg.web.cmu.edu/sasl/`). S její pomocí je pak možné `svnserve` konfigurovat tak, aby prováděl autentizaci některou ze široké škály metod podporovaných touto knihovnou, případně veškerý síťový provoz mezi server a klientem také šifroval. Konfigurace je pak ale náročnější.

do souborů repository, vyvstávají proto potencionální problémy s přístupovými právy a tak dále.

Je sice možné nastavovat přístupová práva v konfiguraci repository (ne na úrovni adresářů, pouze pro celou repository), ale vzhledem k tomu, že má daný uživatel stejně právo zápisu přímo do adresáře repository, nehraje to už příliš velkou roli.

Výhodou tedy je jen to, že je možné využít existujících SSH účtů a že je veškerý síťový provoz šifrovaný.

4.2.4. Apache HTTP server

Asi jediný skutečně použitelný způsob přístupu k repository přes veřejnou síť a při větším množství uživatelů je použití HTTP serveru Apache s moduly `mod_dav` a `mod_dav_svn` a komunikace protokolem WebDAV/DeltaV. V tomto případě přistupuje Subversion klient k repository přes schéma `http://`, respektive `https://`.

Výhodou je možnost využití všech způsobů autentizace, které Apache nabízí, veškerá komunikace může být šifrována pomocí TLS/SSL, opět není třeba vytvářet systémové účty a řízení přístupu k obsahu repository je možné na úrovni jednotlivých adresářů. K dispozici je také plné logování Apache, HTTP/HTTPS provoz dobře prochází přes firewally, obsah repository může být prohlížen přes webový prohlížeč a může být také připojen jako síťový disk.

Z výše uvedeného plyne, že tento způsob je nejkomplexnější a nabízí nejvíce možností. Díky tomu je ale také obecně náročnější na konfiguraci, což je (spolu s nutností běžícího Apache) jeho hlavní nevýhoda.

Výše uvedený přehled byl uveden především proto, abyste jako uživatelé Subversion tušili, co se asi děje na druhé straně v závislosti na tom, s jakým schématem v adrese k repository přistupujete. V dalším textu se nebudeme konfigurací a správou repository příliš zabývat. Zmíníme se jen o vytvoření a údržbě repository pro použití jediným uživatelem s lokálním přístupem (schéma `file://`), která je velmi jednoduchá a zajímavá při tomto způsobu použití Subversion. Podrobné informace o vytváření, správě a zálohování Subversion repository opět naleznete v Subversion book.

4.3. Revize

Jak bylo uvedeno výše, Subversion s projektem pracuje na souborové úrovni. Když uživatelé provedou v projektu nějakou změnu a rozhodnou se ji odeslat do repository, je na serveru vytvořena nová revize projektu.

O revizi je nejvhodnější uvažovat jako o snímku celého projektu (tedy celé struktury podadresářů a souborů pod správou Subversion) v okamžiku commitu (commity jsou serializované a atomické). Každá revize má v repository své pořadové číslo (čísluje se od 1 výše), které ji jednoznačně identifikuje. Pokud si

vyžádáme stažení konkrétní revize projektu z repository, získáme na disku obraz konkrétní adresářové struktury a podoby souborů projektu v konkrétním časovém okamžiku. Subversion tedy pracuje jako stroj času, kterým se kdykoli můžeme vrátit do kteréhokoli okamžiku existence projektu, a získat jeho přesnou podobu v tomto čase.

Pokud bychom místo každého commitu projektu do repository celý adresář projektu zkopírovali do nějakého vedlejšího adresáře na disku a už nikdy bychom v něm neprovedli žádné úpravy, získali bychom podobnou funkčnost.

Z prostorových důvodů a z důvodu efektivity pochopitelně v repository tímto způsobem data uložena nejsou. Ukládají se pouze změny. Pokud tedy v projektu s 10 000 soubory v jediném z nich změníme na jednom řádku jeden znak a provedeme commit změny do repository, nebude to mít za následek zkopírování všech 10 000 souborů, ale pouze uložení rozdílu mezi původní a novou verzí modifikovaného souboru a implicitní předpoklad, že ostatní soubory vypadají stejně jako v předchozí revizi. Stejně tak při přejmenování/přesunu/zkopírování souboru nejsou data do repository znovu vkládána, ale je zde uložena pouze poznámka, že daný soubor se v předchozí revizi jmenoval jinak.

4.4. Model práce kopie–úprava–sloučení

Když víme, co je to revize projektu a jak přistupovat k repository, je čas zmínit se také o tom, jak vlastně probíhá získávání souborů z repository, jejich modifikace a zveřejňování námi provedených změn.

Aby byla možná efektivní spolupráce více lidí na jednom projektu, Subversion používá model práce kopie–úprava–sloučení. Pokud tedy chce uživatel provést nějaké úpravy, stáhne si nejdříve z repository aktuální verzi souborů. (Pokud tedy stojí o aktuální verzi. Díky uchovávání kompletní historie projektu může sáhnout i po libovolné starší verzi. Může dokonce kombinovat i soubory z různých revizí, tím si ale v tomto jednoduchém úvodu nebudeme komplikovat život.) Stažení je provedeno do uživatelem vybraného adresáře, který se nazývá pracovní kopie (working copy). Prvnímu stažení obsahu repository, kdy na disku ještě žádnou pracovní kopii nemáme, se říká checkout.

Když máme na disku obsah repository, můžeme s ním pracovat. Soubory můžeme upravovat, přidávat, odstraňovat a tak dále. Průběžně bychom na pracovní kopii měli provádět také tak zvaný update. Tedy dotaz na server, zda jiný uživatel (na projektu přece nepracujeme sami; u jiných počítačů zatím uživatelé ve svých pracovních kopiích také provádějí úpravy souborů, soubory mažou, přidávají a tak dále) neprovedl commit do repository, to znamená nevložil do repository nějakou svou úpravu. Pokud ano, jsou z repository staženy změny, které se staly od našeho posledního updatu, a tyto změny jsou sloučeny s naší pracovní kopií.

Pokud tyto změny nekolidují se změnami, které jsme provedli ve své pracovní kopii (například někdo jen přidal další soubor, odstranil nebo upravil soubor,

který jsme my ve své pracovní kopii nemodifikovali, případně modifikovali, ale v jiném místě), jsou tyto změny zaneseny automaticky a uživateli pouze oznámeny.

Pokud „máme smůlu“ a někdo jiný modifikoval úplně stejnou část souboru jako my (například někdo upravil úplně stejný řádek textu, který jsme upravovali my), Subversion již pochopitelně nezvládne změny sloučit sama – neví, která modifikace má platit, případně jak obě modifikace sloučit do jedné. V tomto okamžiku nastane takzvaný konflikt a uživatel je vyzván k jeho vyřešení. Je pak na nás, abychom si obě změny prohlédli a rozhodli, jak má daný řádek nakonec vypadat (můžeme například některou ze změn zahodit, nějakým způsobem obě změněné podoby daného řádku zkombinovat do nové verze, která bude obsahovat informaci z obou modifikací a podobně).

Pokud jsme ve své pracovní kopii dokončili nějakou ucelenou změnu (například dopsali novou funkci programu, dokončili novou kapitolu textu a podobně), je načase ji z naší soukromé pracovní kopie commitnout do repository, a tím ji zpřístupnit ostatním.

Ještě těsně před tím, než provedeme commit, bychom měli provést update své pracovní kopie a přesvědčit se tak, že máme ve své pracovní kopii začleněnu aktuální podobou souborů z repository. (Pokud to neuděláme a nemáme staženu aktuální podobu projektu z repository, commit selže a jsme vyzváni k provedení aktualizace své pracovní kopie.) Musíme také zkontrolovat, že celý projekt je po sloučení všech změn funkční. (Pokud například vytváříme nějaký program, může se stát, že naše změny sice půjdou bez problémů automaticky začlenit, protože přímo nekolidují na úpravách stejných kusů kódu. Když ale výslednou podobu programu přeložíme, tak zjistíme, že program nefunguje, protože například naše nová modifikovaná verze nějaké funkce, kterou jsme upravili, špatně spolupracuje s aktuální podobou jiné části programu, jehož autorem nejsme my, ale někdo úplně jiný.)

Pokud po updatu zjistíme, že je vše v pořádku, můžeme provést commit. Nesmíme přitom zapomenout přidat ke commitu výstižný komentář popisující, jaké změny byly provedeny. Je to důležité proto, aby bylo možné se ve změnách rychle orientovat a ostatní uživatelé snadno zjistili, co se na projektu změnilo.

Commitem jsou do repository vloženy naše úpravy a je vytvořena nová revize projektu – tedy pevný bod v historii projektu, který je dále přístupný všem uživatelům. A teď už je na ostatních vývojářích, aby si oni provedli update svých pracovních kopií a nějak se vypořádali se začleněním námi provedených změn do svých pracovních kopií.

5. Používání Subversion krok za krokem prakticky

Po teoretickém úvodu je čas předvést si práci se Subversion prakticky.[6] Instalace Subversion je jednoduchá, a proto zde nebude popisována (navíc se liší dle operačního systému, který používáte). Pod GNU/Linuxem naleznete Subversion s velmi vysokou pravděpodobností v repositářích své distribuce. Pod operačními systémy MS Windows, Apple Mac OS X a podobně je asi nejvhodnější stáhnout si instalátor z domovské stránky projektu (`http://subversion.apache.org/`).[7]

5.1. Vytvoření repository

Začneme založením repository pro případ, že chceme Subversion používat jako jediný uživatel lokálně. K práci s repository slouží program `svnadmin`. Obdobně jako u ostatních obslužných programů Subversion, i zde pracuje schéma `<program> --help` pro získání přehledu základních příkazů daného programu a `<program> <podpříkaz> --help` pro získání podrobných informací o použití konkrétního podpříkazu.

Repository založíme příkazem:

```
~/repos$ svnadmin create dokument-repository
```

Tím jsme v adresáři `~/repos/dokument-repository/` založili novou prázdnou repository. Pokud budeme repository používat pouze my sami jako jediný uživatel lokálně (což bude náš modelový případ), tímto je vše potřebné hotovo a nemusíme nic dalšího nastavovat. Repository můžeme rovnou začít používat.

Pokud byste chtěli repository zpřístupnit jiným způsobem, nastavení přístupových práv se provádí v textových konfiguračních souborech v podadresáři `conf/`, háčky je možné definovat v `hooks/`, a v případě použití FSFS jako úložiště dat je možné některé parametry repository nastavit v souboru `db/fsfs.conf`. Do ostatních adresářů a souborů se *ručně nezasahuje*. Jsou modifikovány programově.

5.2. Prvotní naplnění repository

Repository tedy máme. Zatím je ale prázdná. Vytvoříme si proto základní adresářovou strukturu.[8]

[6] Text je psán s použitím Subversion 1.7.1 pod GNU/Linuxem. Pokud něco nefunguje / chová se jinak, ověřte si prosím, že to není způsobeno odlišnou verzí Subversion (nebo drobně odlišnou syntaxí při zadávání cest v některých jiných operačních systémech).

[7] Kromě standardního řádkového Subversion klienta jsou pro různé operační systémy k dispozici i grafické klienty Subversion. Podpora verzovacích systémů bývá často přímo vestavěna i do integrovaných vývojových prostředí pro různé programovací jazyky.

[8] Níže použitý formát adresy repository je platný pro unixové operační systémy. V operačních systémech z rodiny MS Windows je cestu k repository možné zapsat ve tvaru `"file:///S|/repos/dokument-repository/"` nebo `file:///S:/repos/dokument-repository/` (pro repository uloženou v adresáři `S:\repos\dokument-repository\`).

```
~/repos$ mkdir skel
~/repos$ cd skel/
~/repos/skel$ mkdir branches tags trunk
~/repos/skel$ svn import file://$HOME/repos/dokument-repository/ \
> -m 'Iniciální import.'
Adding         trunk
Adding         branches
Adding         tags

Committed revision 1.
~/repos/skel$ cd ..
~/repos$ rm -r skel/
```

Příkaz `import` je určen právě k iniciálnímu naplnění prázdné repository. V našem případě jsme v kořenovém adresáři repository pouze vytvořili tři prázdné adresáře (`branches`, `tags`, `trunk`) a změnu okomentovali jako iniciální import. Pokud by zpráva nebyla vložena přímo na příkazovém řádku, byl by interaktivně spuštěn textový editor, ve kterém by bylo možné zprávu vložit.

O úspěšném importu se můžeme přesvědčit z logu změn (parametr `-v` zapíná podrobnější výpis zahrnující i seznam změněných souborů).

```
~/repos$ svn log -v file://$HOME/repos/dokument-repository/
------------------------------------------------------------------------
r1 | michalr | 2011-11-27 18:14:38 +0100 (Ne, 27 lis 2011) | 1 line
Changed paths:
   A /branches
   A /tags
   A /trunk

Iniciální import.
------------------------------------------------------------------------
```

Písmeno `A` ve výpisu značí nově přidané soubory/adresáře.

Aktuální obsah repository si můžeme prohlédnout (parametr `-R` zapíná rekurzivní výpis):

```
~/repos$ svn list -R file://$HOME/repos/dokument-repository/
branches/
tags/
trunk/
```

Uvedená kostra adresářové struktury nebyla zvolena náhodně, ale jedná se o doporučený model. (Ale skutečně jen doporučený. Jakým způsobem budete soubory v repository organizovat záleží plně na vás. Subversion žádné konkrétní chování nevyžaduje.) Předpokládá se, že hlavní vývoj bude probíhat v adresáři `trunk/`. Významné revize projektu (například veřejně vydané stabilní verze 1.0.0, 1.0.2, 1.1.0 . . . vyvíjeného programu) budou zkopírovány v podadresářích v `tags/`

jako tak zvané tagy (aby byly snadno přístupné a nebylo je nutné hledat v záplavě ostatních revizí v `trunk/`), podadresář `branches/` je pak vyhrazen pro soukromé větve. Větvení a vytváření tagů bude popsáno později.

Povšimněte si také formátu adresy repository v příkladech – `file://$HOME/repos/dokum` po expanzi proměnné `$HOME` tedy `file:///home/michalr/repos/dokument-repository/`. Toto je ono schéma `file://` zmiňované v předchozím oddílu. Pokud by repository byla uložena na vzdáleném stroji a zpřístupněna přes Apache s TLS/SSL šifrování, stačilo by v příkladech použít adresu `https://nejaka.domena/svn/dokument-repository/` jinak by práce probíhala naprosto stejně.

5.3. Autentizace

Minimálně pro zápis do repository využívané více lidmi bude ve většině případů nutné se autentizovat. Při prvním přístupu k autentizované repository je tedy nutné uvést uživatelské jméno (pokud se liší od vašeho systémového uživatelského jména, pod kterým klienta Subversion používáte, můžete je specifikovat parametrem `--username`) a heslo. Heslo bude vyžádáno interaktivně, je ale možné (z bezpečnostního hlediska se to ale samozřejmě *nedoporučuje*) zadat ho také přímo na příkazové řádce parametrem `--password`.

Výchozím chováním Subversion je uživatelská jména i hesla pro jednotlivé repository trvale ukládat. Místo uložení se liší dle použitého operačního systému – v systémech MS Windows jsou data šifrována standardními kryptografickými službami Windows a uložena v `%APPDATA%\Subversion\auth\`, pod Apple Mac OS X se užívá systémové Keychain service, pod unixovými operačními systémy se může použít například GNOME Keyring, KDE Wallet, či jednoduché uložení v `~/.subversion/auth/`, pokud nic z výše uvedeného není dostupné (na rozdíl od předchozích možností jsou však v tomto případě hesla ukládána v otevřené, nijak nechráněné podobě, takže se o příliš bezpečné řešení nejedná). Ukládání uživatelských jmen i hesel je možné potlačit v konfiguračním souboru `~/.subversion/config`. Jednorázově je to možné provést i přímo na příkazovém řádku argumentem `--no-auth-cache`.

V případě přímého přístupu do repository (`file://`) jsou nastavení přístupových práv ignorována, takže v dalších příkladech se již autentizací a přístupovými právy zabývat nebudeme.

5.4. Checkout

V předchozích příkladech (výpis logu, výpis obsahu repository) jsme stále pracovali na straně serveru (přímo s repository). Pokud se nechceme na data z repository pouze ptát, ale chceme je i upravovat, přišel čas na checkout.

Jako checkout je označováno vytvoření soukromé lokální pracovní kopie stažením dat z repository. Pracovní kopii můžeme umístit do libovolného adresáře.

Každá pracovní kopie je zcela nezávislá. Můžeme si tedy vytvořit pracovních kopií klidně více.[9] (I když to nebývá zvykem. Například pro rozdělení současně zapracovávaných odlišných změn je lépe využít soukromých větví. Vizte oddíl 5.10.)

V našem modelovém případě si pracovní kopii umístíme do podadresáře `dokument-working-copy` ve stejném adresáři, v němž máme uloženou repository:

```
~/repos$ svn checkout file://$HOME/repos/dokument-repository/ \
> dokument-working-copy
A    dokument-working-copy/trunk
A    dokument-working-copy/branches
A    dokument-working-copy/tags
Checked out revision 1.
~/repos$ ls
dokument-repository  dokument-working-copy
```

Tímto máme obsah repository stažený do pracovní kopie. Pokud neřekneme jinak, stahuje se obsah poslední revize projektu. Pokud bychom měli zájem o obsah starší revize, můžeme revizi specifikovat argumentem `-r`.

V našem případě jsme si stáhli celý obsah repository. To není moc obvyklé. Zejména u velkých projektů, které mají spoustu kopií význačných revizí v `tags/`, si zpravidla stáhneme jen data, která nás zajímají – například hlavní vývojový strom v `trunk/`. To by se provedlo jednoduše tak, že bychom k URL repository připojili i cestu k adresáři, který nás zajímá (například `file://$HOME/repos/dokument-repository/trunk/`).

Pro většinu příkazů Subversion klienta existují zkratky. Místo `svn checkout` bychom mohli napsat jen `svn co`. Obdobné zkratky existují pro mnoho dalších příkazů (vizte `svn --help`, respektive `svn <podpříkaz> --help`). V tomto článku se jim ale pro názornost pokusíme vyhnout.

Pokud se podíváte na obsah pracovní kopie podrobně, zjistíte, že kromě našich souborů je v nejvyšším adresáři pracovní kopie ještě „tečkový“ (to znamená skrytý) adresář `.svn`. V něm jsou uložena metadata Subversion. *V žádném případě tento skrytý adresář neodstraňujte ani nemodifikujte.* O práci s ním se stará automaticky klient Subversion.[10]

```
~/repos$ cd dokument-working-copy/
~/repos/dokument-working-copy$ ls -lA
celkem 16
drwx------ 3 michalr users 4096 31. pro 12.38 branches
```

[9] V tomto článku bude více pracovních kopií využito k simulaci současné práce více uživatelů s jednou repository.

[10] V tomto ohledu došlo k výrazné změně od Subversion verze 1.7. Ve verzích starších byly „tečkové“ adresáře `.svn` přítomny ve všech podadresářích celé pracovní kopie a obsahovaly metadata jen pro daný adresář. Od verze 1.7 jsou veškerá metadata pro celou pracovní kopii soustředěna v jediném kořenovém `.svn` adresáři, používá se přitom také odlišný formát uložení dat.

```
drwx------ 6 michalr users 4096 31. pro 12.38 .svn
drwx------ 3 michalr users 4096 31. pro 12.38 tags
drwx------ 3 michalr users 4096 31. pro 12.38 trunk
```

5.5. Provádění změn a jejich commit do repository

Se soubory v pracovní kopii můžeme nyní pracovat téměř běžným způsobem. Při editaci obsahu souborů pracujeme naprosto stejně, jako bychom Subversion vůbec nepoužívali. Rozdíly v práci se týkají především souborových operací.

Pokud chceme pod správu Subversion přidat nový soubor nebo adresář, použijeme k tomu příkaz `svn add`. U adresářů můžeme k jejich vytvoření použít přímo příkaz `svn mkdir`.

V našem modelovém případě si v adresáři `trunk/` vytvoříme soubor pojmenovaný `dokument.tex`.

```
~/repos/dokument-working-copy$ cd trunk
~/repos/dokument-working-copy/trunk$ touch dokument.tex
```

Nyní jej musíme explicitně přidat pod správu Subversion.

```
~/repos/dokument-working-copy/trunk$ svn add dokument.tex
A         dokument.tex
```

Soubor by nemusel být prázdný, klidně si můžeme obsah souboru nejdříve připravit, a až posléze jej přidat pod správu Subversion.

Stav pracovní kopie můžeme zkontrolovat příkazem `svn status`.

```
~/repos/dokument-working-copy/trunk$ svn status
A       dokument.tex
```

Pokud přidáme ještě argument `-u`, bude stav zkontrolován ne proti původnímu stavu pracovní kopie (to znamená stavu po posledním updatu/checkoutu – tedy pouze změny, které jsme my provedli v naší soukromé pracovní kopii), ale oproti poslední revizi v repository na serveru (to znamená, že zahrnuje i změny, které mezi tím ostatní uživatelé commitovali do repository a které budou do naší pracovní kopie včleněny při dalším updatu).

Pro odstraňování souborů a adresářů slouží příkaz `svn delete`. Obdobně provádíme i kopírování (`svn copy`) a přesouvání/přejmenovávání (`svn move`) souborů a adresářů.

Pokud například chceme nějaký soubor přejmenovat, přesunout nebo zkopírovat, musíme Subversion dát vědět, že nový soubor je jen jinak pojmenovaný soubor, který již v repository je. Místo

```
mv soubor1 soubor2
```

či

```
cp soubor1 soubor2 && svn delete soubor1 && svn add soubor2
```

použijeme příkaz:

```
svn move soubor1 soubor2
```

Obdobně pro kopírování použijeme `svn copy`.[11]

Používání správných příkazů pro manipulaci se soubory pod správou Subversion je důležité. Šetříme tím místo v repository (pokud Subversion ví, že se jedná o kopii jiného souboru, který již v nějaké revizi v repository je, data jsou v repository fyzicky uložena jen jednou a z různých míst pouze odkazována), a především neztrácíme informaci o historii souboru. Pokud bychom přejmenování souboru provedli výše naznačeným nesprávným postupem, nejsme později schopni zjistit historii `soubor2` před jeho přejmenováním (museli bychom si pamatovat, že se dříve jmenoval `soubor1`, a že se tedy musíme ptát na historii tohoto souboru). Pro Subversion by byl `soubor2` úplně nový soubor bez jakékoli historie přidaný v dané revizi, nijak by si ho nemohla spojit se `soubor1` odstraněným v téže revizi.

Nyní můžeme do souboru `dokument.tex` přidat nějaký obsah.

```
~/repos/dokument-working-copy/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy/trunk$ cat dokument.tex
\documentclass{article}
\usepackage[czech]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\begin{document}
\hyphenation{Open-Of-fice UNESCO}

Ahoj světe!

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char‘\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

Podrobný přehled o námi provedených změnách čekajících v pracovní kopii na odeslání do repository získáme příkazem `svn diff`. Bez dalších parametrů vypíše změny všech souborů v pracovní kopii ve formátu unifikovaného diffu.

[11] Příkazy se nemusí odkazovat jen na soubory, které aktuálně jsou v pracovní kopii. Místo lokálního souboru můžeme zadat URL souboru v repository. A to v libovolné revizi. Bližší informace o argumentech příkazů můžete zjistit zadáním `svn <podpříkaz> --help` či `svn help <podpříkaz>`, případně v Subversion book.

Pokud zadáme konkrétní soubory, výpis omezíme na změny v těchto souborech.

Pokud jsme s úpravami spokojeni, můžeme naši první změnu odeslat do repository.

```
~/repos/dokument-working-copy/trunk$ svn commit \
> -m 'Přidána kostra dokumentu.'
Adding         trunk/dokument.tex
Transmitting file data .
Committed revision 2.
```

Nesmíme zapomenout opět přidat nějaký výstižný komentář této změny. Stejně jako při importu na něj budeme dotázáni interaktivně, pokud jej nezadáme už na příkazové řádce.

Dobře okomentované commity provádíme co nejčastěji, a to při dokončení každé drobné sady logicky souvisejících úprav. Jen tímto způsobem bude v repository zaznamenán podrobný průběh vývoje, ze kterého bude dobře patrné, proč byla která úprava provedena. Bude také mnohem jednodušší konkrétní nechtěné úpravy zase zrušit, či zpětně dohledat příčiny chyb, které se nečekaně vynoří jakoby odnikud. Naprosto špatným postupem je provádění například jednoho commitu obsahujícího celou řadu spolu nesouvisejících modifikací jednou denně na konci pracovní doby. Rozsáhlejší úpravy je vhodné provádět v samostatné vývojové větvi (vizte oddíl 5.10) rozdělené na drobné, logicky související commity jednotlivých kroků úprav.

Všimněte si také, že již nezadáváme adresu repository. Adresa repository je jednou z informací, která je uložena v adresáři `.svn` v každé pracovní kopii. Z toho ale také plyne, že pokud se adresa repository změní (například dojde k přesunu na jiný server nebo ji přemístíme do jiného adresáře na disku), bude tato informace chybná. Buď si pak můžeme stáhnout novou pracovní kopii (což není moc vhodné, pokud již máme v současné pracovní kopii nějaké necommitnuté změny), nebo informaci o adrese opravíme použitím příkazu `svn relocate`. (Vizte `svn relocate --help` nebo Subversion book.)

5.6. Update pracovní kopie a sledování změn

Jelikož používáme repository sami, máme jistotu, že nikdo jiný zatím žádnou změnu necommitoval. Správně bychom ale před naším commitem měli nejdříve provést update naší pracovní kopie a ověřit si, že po začlenění všech změn případně provedených jinými uživateli od našeho posledního updatu je projekt plně funkční.

```
~/repos/dokument-working-copy/trunk$ svn update
Updating '.':
At revision 2.
```

Po updatu se můžeme podívat na log změn.

```
~/repos/dokument-working-copy/trunk$ svn log
------------------------------------------------------------------------
r2 | michalr | 2011-11-27 18:38:45 +0100 (Ne, 27 lis 2011) | 1 line

Přidána kostra dokumentu.
------------------------------------------------------------------------
r1 | michalr | 2011-11-27 18:14:38 +0100 (Ne, 27 lis 2011) | 1 line

Iniciální import.
------------------------------------------------------------------------
```

Log samozřejmě nemusíme vždy vypisovat pro celou pracovní kopii. Argumentem můžeme specifikovat konkrétní soubor/adresář, pro který má být log vypsán.

Použití `svn diff` se neomezuje jen na zobrazení změn čekajících v pracovní kopii. S odpovídajícími parametry jej můžeme použít i pro výpis rozdílů mezi různými revizemi projektu dříve uloženými v repository. Obsah konkrétního souboru z libovolné revize vypíšeme příkazem `svn cat` s odpovídajícími parametry.

5.7. Odvolávání změn

Pro další příklady si nasimulujeme souběžnou práci dvou uživatelů. Vytvoříme si proto ještě jednu pracovní kopii.

```
~/repos/dokument-working-copy/trunk$ cd ../..
~/repos$ svn checkout file://$HOME/repos/dokument-repository/ \
> dokument-working-copy2
A    dokument-working-copy2/trunk
A    dokument-working-copy2/trunk/dokument.tex
A    dokument-working-copy2/branches
A    dokument-working-copy2/tags
Checked out revision 2.
~/repos$ cd dokument-working-copy2/trunk/
```

„Jiný uživatel“ teď může provést nějakou změnu. Například nám z nějakého důvodu zruší skoro celou naši práci…

```
~/repos/dokument-working-copy2/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy2/trunk$ cat dokument.tex
% Začátek dokumentu.
Ahoj světe!
% Konec dokumentu.
```

…a vše odešle na server.

```
~/repos/dokument-working-copy2/trunk$ svn commit --username user2 \
> -m 'Oprava dokumentu'
```

```
Sending        trunk/dokument.tex
Transmitting file data .
Committed revision 3.
```

Když první uživatel provede update…

```
~/repos/dokument-working-copy/trunk$ svn update
Updating '.':
U    dokument.tex
Updated to revision 3.
```

…a zjistí, co se stalo…[12]

```
~/repos/dokument-working-copy/trunk$ svn diff -r2:3
Index: dokument.tex
===================================================================
--- dokument.tex (revision 2)
+++ dokument.tex (revision 3)
@@ -1,18 +1,3 @@
-\documentclass{article}
-\usepackage[czech]{babel}
-\usepackage[utf8]{inputenc}
-\usepackage[T1]{fontenc}
-
-\begin{document}
-\hyphenation{Open-Of-fice UNESCO}
-
+% Začátek dokumentu.
 Ahoj světe!
-
-V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
-se vkládá jako znak tilda (\char`\~). K~práci s~nezlomitelnými
-mezerami slouží program Vlna.
-
-Nový odstavec se vytvoří vynecháním řádku.
-
-Šíleně žluťoučký kůň úpěl ďábelské ódy.
-\end{document}
+% Konec dokumentu.
```

…bude chtít tyto změny odvolat.

Vzhledem k tomu, že změna již byla commitnuta do repository, fyzicky se jí nezbavíme. Od toho nakonec Subversion používáme, abychom žádnou změnu nikdy neztratili. Máme tedy jedinou možnost – vytvořit novou revizi projektu, ve které již tato změna existovat nebude.

[12] Zkratkou pro `-r2:3` je `-c3` (změna zachycená v revizi 3, to znamená rozdíl mezi soubory v revizi 3 a $(3-1)$). Pokud chceme seznam změn od revize 2 až po současnost, můžeme zadat jen `-r2`.

Existuje více postupů, jak toho dosáhnout. Jednou z možností je jednoduše současný soubor vymazat a zkopírovat jeho podobu uloženou v revizi 2.

```
~/repos/dokument-working-copy/trunk$ ls -l
celkem 4
-rw------- 1 michalr users 55 2011-11-27 18:49 dokument.tex
~/repos/dokument-working-copy/trunk$ svn delete dokument.tex
D         dokument.tex
~/repos/dokument-working-copy/trunk$ ls -l
celkem 0
~/repos/dokument-working-copy/trunk$ svn copy -r2 dokument.tex .
A         dokument.tex
~/repos/dokument-working-copy/trunk$ ls -l
celkem 4
-rw------- 1 michalr users 467 2011-11-27 18:53 dokument.tex
```

Ani v tomto případě jsme nepřišli o žádnou historii a šetříme místem. Subversion ví, že tato podoba souboru již je v repository uložena v revizi 2 a může toho využít.

Pokud bychom si uvědomili, že tato naše změna nebyl dobrý krok, můžeme ji jednoduše vrátit. K odvolání změn, které dosud nebyly commitnuty do repository, to znamená změn, které existují pouze v naší pracovní kopii, slouží příkaz `svn revert`.

```
~/repos/dokument-working-copy/trunk$ svn revert dokument.tex
Reverted 'dokument.tex'
~/repos/dokument-working-copy/trunk$ ls -l
celkem 4
-rw------- 1 michalr users 55 2011-11-27 18:54 dokument.tex
```

Univerzálnější variantou, jak odvolat změny provedené v některém z předchozích commitů, je použití příkazu `svn merge`. Tento příkaz má za úkol porovnat dva soubory (respektive celé adresářové podstromy) a rozdíl mezi nimi sloučit do zadaného souboru (respektive adresářového podstromu). Primárně slouží pro práci s větvemi a řeč o něm bude ještě v oddílu 5.10.

My ale také nepotřebujeme provést nic jiného, než vzít rozdíl mezi dvěma soubory (mezi souborem `dokument.tex` v revizi 2 a v revizi 3) a sloučit je do souboru `dokument.tex` v naší současné pracovní kopii. Protože změnu odvoláváme, musíme získat opačnou podobu změny, to znamená ne změny provedené z revize 2 na revizi 3, ale změny provedené z revize 3 na revizi 2.

Výše uvedený složitě vypadající manévr provedeme jednoduchým příkazem.

```
~/repos/dokument-working-copy/trunk$ svn merge -r3:2 dokument.tex
--- Reverse-merging r3 into 'dokument.tex':
U    dokument.tex
--- Recording mergeinfo for reverse merge of r3 into 'dokument.tex':
```

```
 U   dokument.tex
--- Eliding mergeinfo from 'dokument.tex':
 U   dokument.tex
```

Symbol `U` nám oznamuje, že daný soubor byl aktualizován (byl změněn jeho obsah). Pokud se na jeho obsah podíváte, zjistíte, že soubor má stejnou podobu jako v revizi 2.

```
~/repos/dokument-working-copy/trunk$ ls -l
celkem 4
-rw------- 1 michalr users 467 2011-11-27 18:57 dokument.tex
```

Proč používat tak složitý postup? Všimněte si, že v tomto případě jsme nemuseli soubor nejdříve vymazat. Pokud bychom již v souboru měli své lokální změny, pomocí `svn merge` bychom již commitovanou změnu mohli odvolat bez zrušení našich dřívějších lokálních (dosud necommitovaných) úprav. Rovněž nejsme omezeni na odvolání změny provedené v poslední revizi projektu. Můžeme snadno odvolat změny provedené v libovolné starší revizi (či revizích) projektu bez toho, abychom zrušili i úpravy zachycené v revizích následujících. Nevracíme se zpět v čase se vším všudy, rušíme jen vybranou část dříve provedených změn.[13]

Nyní již nezbývá nic jiného, než odvolání změny uložit do repository.

```
~/repos/dokument-working-copy/trunk$ svn commit dokument.tex \
> -m 'Odvolání změn provedených z revize 2 do revize 3.'
Sending        dokument.tex
Transmitting file data .
Committed revision 4.
```

5.8. Řešení konfliktů

Doposud jsme si ukázali pouze takové úpravy souborů, které Subversion zvládla při updatu sloučit automaticky bez zásahu uživatele. Pokud ale na jednom projektu spolupracuje větší počet lidí, je velmi pravděpodobné, že dříve nebo později dojde k takovému souběhu změn provedených dvěma a více uživateli nezávisle na sobě, že Subversion nezvládne změny sloučit automaticky a dojde k jejich konfliktu.

Zkusme si takový konflikt ukázat prakticky. Nejdříve provede update první uživatel...

```
~/repos/dokument-working-copy/trunk$ svn update
Updating '.':
At revision 4.
```

...poté i druhý uživatel.

[13] Zde je názorně vidět, proč je důležité provádět časté commity malých logicky souvisejících změn – konkrétní změny pak můžeme snadno odvolat bez narušení jiných úprav.

```
~/repos/dokument-working-copy2/trunk$ svn update
Updating '.':
U    dokument.tex
Updated to revision 4.
```

Oba uživatelé teď pracují se stejnou verzí dokumentu.

Nyní si představme, že by první uživatel přešel na novou verzi podpory češtiny v balíčku Babel v LaTeXu a dokument dle toho upravil.

```
~/repos/dokument-working-copy/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy/trunk$ cat dokument.tex
\documentclass{article}
\usepackage[czech]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\begin{document}
\hyphenation{Open-Of-fice UNESCO}

Ahoj světe!

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char`\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Text vložíme do \uv{uvozovek} příkazem "`\verb|\uv|"', případně
dvojicí "`\verb|"`|"' a~"`\verb|"'|"'. Vložení spojovníku se
správným českým chováním bez uvádění globálního nastavení
"`\verb|\languageattribute{czech}{split}|"' či užití příkazů
"`\verb|\splithyphens|"' a~"`\verb|\standardhyphens|"' dosáhneme
vložením "`\verb|"=|"'.

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

Mezi tím by však druhý uživatel ve své pracovní kopii na dokumentu také pracoval a rozhodl se dokument sázet zastaralým CSLaTeXem. Upravil by proto odpovídajícím způsobem jeho zdrojový text.

```
~/repos/dokument-working-copy2/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy2/trunk$ cat dokument.tex
\documentclass{article}
\usepackage{czech}
```

```
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\begin{document}
\hyphenation{Open-Of-fice UNESCO}

Ahoj světe!

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char‘\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Text vložíme do \uv{uvozovek} příkazem \clqq\verb|\uv|\crqq,
případně dvojicí příkazů \clqq\verb|\clqq|\crqq{}
a~\clqq\verb|\crqq|\crqq.

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

První uživatel je s úpravami hotov o něco dříve, a tak provede update své pracovní kopie, zjistí, že je vše v pořádku a bez konfliktů, a proto odešle novou verzi dokumentu do repository.

```
~/repos/dokument-working-copy/trunk$ svn update
Updating '.':
At revision 4.
~/repos/dokument-working-copy/trunk$ svn commit \
> -m 'Přechod na novou podporu češtiny v LaTeXu.'
Sending        dokument.tex
Transmitting file data .
Committed revision 5.
~/repos/dokument-working-copy/trunk$ svn update
Updating '.':
At revision 5.
```

Když je se svou prací hotov i druhý uživatel, chystá se své úpravy odeslat do repository. Nejdříve tedy provede update, aby se přesvědčil, že pracuje s aktuální verzí projektu.

```
~/repos/dokument-working-copy2/trunk$ svn update
Updating '.':
Conflict discovered in
         '/home/michalr/repos/dokument-working-copy2/trunk/dokument.tex'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
```

```
        (s) show all options:
```

Bohužel, u souboru `dokument.tex` došlo ke konfliktu a my jej musíme vyřešit. Subversion od verze 1.5 ve výchozím nastavení nabízí výše uvedený interaktivní dialog pro slučování změn.

Nejdříve asi budeme chtít vědět, kde došlo ke konfliktu. To můžeme zjistit zadáním příkazu `df`, což vede k vypsání rozdílů ve formátu unifikovaného diffu.

```
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: df
--- /home/michalr/repos/dokument-working-copy2/.svn/pristine/27/
         2729e011f587de52b4add1bd92dc2fc0b62ac68e.svn-base
         Ne lis 27 18:47:10 2011
+++ /home/michalr/repos/dokument-working-copy2/.svn/tmp/dokument.tex.tmp
         Ne lis 27 19:07:36 2011
@@ -1,5 +1,5 @@
 \documentclass{article}
-\usepackage[czech]{babel}
+\usepackage{czech}
 \usepackage[utf8]{inputenc}
 \usepackage[T1]{fontenc}

@@ -14,5 +14,19 @@

 Nový odstavec se vytvoří vynecháním řádku.

+<<<<<<< .mine
+Text vložíme do \uv{uvozovek} příkazem \clqq\verb|\uv|\crqq,
+případně dvojicí příkazů \clqq\verb|\clqq|\crqq{}
+a~\clqq\verb|\crqq|\crqq.
+
+=======
+Text vložíme do \uv{uvozovek} příkazem "‘\verb|\uv|"’, případně
+dvojicí "‘\verb|"‘|"’ a~"‘\verb|"’|"’. Vložení spojovníku se
+správným českým chováním bez uvádění globálního nastavení
+"‘\verb|\languageattribute{czech}{split}|"’ či užití příkazů
+"‘\verb|\splithyphens|"’ a~"‘\verb|\standardhyphens|"’ dosáhneme
+vložením "‘\verb|"=|"’.
+
+>>>>>>> .r5
 Šíleně žluťoučký kůň úpěl ďábelské ódy.
 \end{document}
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
```

```
        (s) show all options:
```

Máme tedy několik možností, jak postupovat.

```
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: s

  (e)  edit             - change merged file in an editor
  (df) diff-full        - show all changes made to merged file
  (r)  resolved         - accept merged version of file

  (dc) display-conflict - show all conflicts (ignoring merged version)
  (mc) mine-conflict    - accept my version for all conflicts (same)
  (tc) theirs-conflict  - accept their version for all conflicts (same)

  (mf) mine-full        - accept my version of entire file (even
                          non-conflicts)
  (tf) theirs-full      - accept their version of entire file (same)

  (p)  postpone         - mark the conflict to be resolved later
  (l)  launch           - launch external tool to resolve conflict
  (s)  show all         - show this list

Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Nejjednodušší možností je přijmout jednu verzi (například naši (`mf`), nebo tu z repository (`tf`)) celého souboru (to znamená nejen kolidujících řádků) a druhou verzi úplně zahodit. Jemnější variantou je přijmutí jedné verze kolidujících řádků (například naši (`mc`), nebo tu z repository (`tc`)) a druhou verzi úplně zahodit. Nekolidující řádky zůstanou nedotčeny v té podobě, v jaké byly v naší pracovní kopii po updatu.

Pokud konflikt není takového charakteru, že by bylo možné jednoduše přijmout jednu z verzí, ale je třeba obě změny nějak šikovně spojit dohromady, můžeme příkazem `e` interaktivně otevřít konfliktní soubor v textovém editoru a dle potřeby jej upravit. Vodítkem nám mohou být vložená návěští, která ukazují konfliktní řádky v podobě, v jaké se vyskytují v obou verzích souboru.

```
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: e
                        ...
            <spuštění textového editoru>
                        ...
```

```
\documentclass{article}
\usepackage{czech}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\begin{document}
\hyphenation{Open-Of-fice UNESCO}

Ahoj světe!

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char‘\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

<<<<<<< .mine
Text vložíme do \uv{uvozovek} příkazem \clqq\verb|\uv|\crqq,
případně dvojicí příkazů \clqq\verb|\clqq|\crqq{}
a~\clqq\verb|\crqq|\crqq.

=======
Text vložíme do \uv{uvozovek} příkazem "‘\verb|\uv|"’, případně
dvojicí "‘\verb|"‘|"’ a~"‘\verb|"’|"’. Vložení spojovníku se
správným českým chováním bez uvádění globálního nastavení
"‘\verb|\languageattribute{czech}{split}|"’ či užití příkazů
"‘\verb|\splithyphens|"’ a~"‘\verb|\standardhyphens|"’ dosáhneme
vložením "‘\verb|"=|"’.

>>>>>>> .r5
Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
                        ...
            <ukončení textového editoru>
                        ...
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Pokud bychom soubor upravili do konečné podoby (a odstranili i pomocná návěští `<<<<<<< .mine`, `>>>>>>> .r5` a `=======`), mohli bychom po ukončení textového editoru konflikt označit za vyřešný příkazem `r`.

Pokud nechceme konflikt řešit interaktivně v průběhu updatu pracovní kopie,

mohli bychom jej odložit příkazem `p`.[14]

```
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: p
C    dokument.tex
Updated to revision 5.
Summary of conflicts:
  Text conflicts: 1
```

Subversion pak oznámí konflikt příznakem `C` a vytvoří tři dočasné pomocné soubory.

```
~/repos/dokument-working-copy2/trunk$ ls
dokument.tex  dokument.tex.mine  dokument.tex.r4  dokument.tex.r5
```

`dokument.tex.mine` je původní soubor uživatele ve stavu, v jakém byl těsně před spuštěním `svn update`. Soubory `document.tex.r4`/`document.tex.r5` obsahují verzi dokumentu ze 4./5. revize tak, jak jsou uloženy v repository. Soubor `dokument.tex` (stejný jako při interaktivní editaci příkazem `e`) je pak kombinací naší verze souboru (`dokument.tex.mine`) s poslední verzí souboru z repository (`document.tex.r5`). Bezkonfliktní změny byly automaticky sloučeny (zde to byla naše úprava `\usepackage[czech]{babel}` na `\usepackage{czech}`), konfliktní úseky kódu (zde odstavec s příklady užití maker pro sazbu uvozovek, který se v obou dokumentech liší) byly vyznačeny návěštími `<<<<<<< .mine`, `=======`, `>>>>>>> .r5` a přenechány k vyřešení uživateli.

Nyní máme v zásadě stejné možnosti jako v případě interaktivního řešení konfliktu. Pokud bychom ručně upravili do finální podoby soubor `document.tex`, vyřešení konfliktu bychom oznámili příkazem:

```
~/repos/dokument-working-copy2/trunk$ svn resolve \
> --accept working dokument.tex
Resolved conflicted state of 'dokument.tex'
```

Argument `--accept` příkazu `svn resolve` však má i jiné hodnoty (vizte `svn resolve --help` nebo Subversion book), které odpovídají možnostem v interaktivním výběru (vizte výše). Jelikož druhý uživatel z logu změn zjistil, že konfliktní změna byla provedena při přechodu na novou verzi podpory češtiny, prozkoumá situaci, rozhodne se opustit CSLaTeX a chce tedy přijmout novou verzi celého souboru tak, jak byla vložena do repository prvním uživatelem.[15]

```
~/repos/dokument-working-copy2/trunk$ svn resolve \
> --accept theirs-full dokument.tex
Resolved conflicted state of 'dokument.tex'
```

[14] Toto chování je možné dopředu vyžádat přidáním argumentu `--non-interactive` při spuštění `svn update`.

[15] V původním interaktivním režimu bychom tohoto hned dosáhli příkazem `tf`.

Po vyřešení konfliktu jsou Subversion vytvořené pomocné soubory automaticky smazány.

```
~/repos/dokument-working-copy2/trunk$ ls
dokument.tex
```

Je třeba pamatovat na to, že vyřešení konfliktu se týká začlenění změn do pracovní kopie (pokud nejsme s výsledkem spokojeni, můžeme lokální změny snadno odvolat příkazem `svn revert`, a pak se pokusit o sloučení znovu od začátku), data nejsou commitována do repository. Aby byla sloučená verze uložena v repository (a zpřístupněna ostatním uživatelům), je třeba provést commit. (Soubory s nevyřešenými konflikty není možné commitovat.)

5.9. Subversion properties

Doposud jsme si ukázali, že Subversion umí verzovat adresáře a soubory (a to včetně například symbolických linků). Kromě toho ale Subversion umí k adresářům a souborům připojit a verzovat i metadata – tak zvané properties.

Properties jsou definovány jako dvojice „klíč=hodnota“, kde klíč je krátký ASCII řetězec určený k identifikaci, hodnotou pak může být libovolný text nebo i binární data (v zásadě libovolný textový či binární soubor). Takových dvojic si může uživatel definovat libovolně mnoho a užívat je dle vlastního uvážení.

Pro práci s properties slouží podpříkazy `svn` klienta `propdel`, `propedit`, `propget`, `proplist` a `propset` (vizte `svn <podpříkaz> --help`). Nastavení, modifikace a odstranění property se chová obdobně jako změna obsahu daného souboru. Změna je tedy nejdříve provedena v lokální kopii a je třeba ji do repository commitovat. Naopak změny properties v repository jsou do lokálních kopií promítány při jejich checkoutu/updatu.

Properties jsou využívány i samotným systémem Subversion. Pro toto použití je rezervován prefix `svn:`, kterým začínají názvy všech properties klíčů, které interpretuje sama Subversion. Pokud například chceme nějaký soubor označit jako spustitelný, nastavíme mu property `svn:executable` (s libovolnou, třeba i prázdnou hodnotou) příkazem `svn propset 'svn:executable' 'ON' soubor.sh`.

Výhodou tohoto řešení je, že tato informace je uložena přímo v repository, nikoli jako příznak v souborovém systému. Subversion je multiplatformní, a pokud někdo tímto způsobem označí soubor například při práci na MS Windows, po checkoutu/updatu souboru pod unixovým systémem bude souboru automaticky nastaven příznak spustitelnosti.

Další šikovnou pomůckou je property `svn:eol-style`, která u textových souborů[16] definuje, jaké znaky konce řádků se mají používat. Spuštěním příkazu `svn propset 'svn:eol-style' 'native' soubor.txt` definujeme na souboru `soubor.txt` property `svn:eol-style` s hodnotou `native`, čímž říkáme, že tento

[16] Typ souboru se dá v případě potřeby explicitně uvést pomocí property `svn:mime-type`.

soubor má být při checkoutu/updatu na disk uložen se znaky konců řádků, které odpovídají dané platformě (to znamená `<LF>` pod Unixem, `<CR>` pod Apple Mac OS a `<CR><LF>` pod MS Windows).

5.9.1. `svn:keywords`

Velice užitečnou property je `svn:keywords`. Často je vhodné, aby verzovaný soubor přímo obsahoval například informaci o tom, z jaké revize projektu pochází. Systém Subversion obsahuje mechanismus náhrady klíčových slov v textových souborech, který umožňuje nechat automaticky vkládat přímo do souborů informace o datu poslední modifikace souboru, číslu revize, ze které soubor pochází, uživatelské jméno autora poslední modifikace souboru a podobně.

Pokud například na textovém souboru nastavíme property `svn:keywords` s hodnotou `Id`, pak každý výskyt řetězce `$Id$` v daném souboru bude Subversion automaticky nahrazován za řetězec formátu
`$Id: jméno_souboru revize datum_a_čas_změny autor $`.
(Nahrazení se děje při uložení do pracovní kopie/exportu souboru. V repository je soubor fyzicky uložen v normalizovaném tvaru s řetězcem `$Id$`.)

```
~/repos/dokument-working-copy/trunk$ svn propset \
> 'svn:keywords' 'Id' dokument.tex
property 'svn:keywords' set on 'dokument.tex'
~/repos/dokument-working-copy/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy/trunk$ head -n 3 dokument.tex
% $Id$
\documentclass{article}
\usepackage[czech]{babel}
```

Nyní můžeme provést commit změny do repository.

```
~/repos/dokument-working-copy/trunk$ svn commit \
> -m 'Přidání identifikace do záhlaví dokumentu.'
Sending        dokument.tex
Transmitting file data .
Committed revision 6.
```

Po commitu dojde v souboru k náhradě klíčových slov, u kterých je to povoleno nastavením property `svn:keywords` (v našem případě tedy k náhradě řetězce `$Id$`).

```
~/repos/dokument-working-copy/trunk$ head -n 3 dokument.tex
% $Id: dokument.tex 6 2011-11-27 18:28:20Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
```

I pokud takto označený soubor obdržíme z jiného zdroje než přímo z repository, tak z jeho obsahu vždy snadno zjistíme jeho přesnou verzi. Subversion přitom

poskytuje i další klíčová slova, která můžeme v dokumentech nechat nahrazovat (vizte Subversion book).

Bohužel, syntaxe využívající znaku $ není z pohledu TEXu zrovna nejšťastněji zvolená, neboť takový zápis má v TEXu speciální význam – zápis matematického výrazu. Pokud tedy chceme zobrazovat informace automaticky vkládané Subversion jako součást výstupního dokumentu, nemůžeme to udělat přímo (text by se sázel jako matematika). Můžeme to však udělat nepřímo.

Pro LATEX existuje předpřipravený balíček `svninfo`, který potřebnou funkcionalitu již poskytuje. Upravíme tedy náš dokument tak, aby informace o verzi souboru automaticky vkládaná Subversion byla součástí výsledného vysázeného dokumentu.

```
~/repos/dokument-working-copy/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy/trunk$ cat dokument.tex
% $Id: dokument.tex 6 2009-12-31 13:39:10Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[nofancy]{svninfo} % Balíček pro práci se Subversion
      % metadaty v dokumentu (s parametrem zakazujícím automatické
      % vložení Subversion metadat do zápatí).
\svnInfo $Id$

\begin{document}
\hyphenation{Open-Of-fice UNESCO}

Ahoj světe!

% Vložení Subversion metadat do výsledného dokumentu.
(Author: \svnInfoOwner, datum: \svnInfoDate{} \svnInfoTime{},
 revize:~\svnInfoRevision)

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char`\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Text vložíme do \uv{uvozovek} příkazem "`\verb|\uv|"', případně
dvojicí "`\verb|"`|"' a~"`\verb|"'|"'. Vložení spojovníku se
správným českým chováním bez uvádění globálního nastavení
"`\verb|\languageattribute{czech}{split}|"' či užití příkazů
```

```
"‘\verb|\splithyphens|"’ a~"‘\verb|\standardhyphens|"’ dosáhneme
vložením "‘\verb|"=|"’.

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

Řetězec `$Id$` Subversion rozezná jako klíčové slovo a expanduje jej. Expandovaný řetězec pak již zpracuje makro `\svnInfo` a patřičně definuje makra `\svnInfoOwner`, `\svnInfoDate`, `\svnInfoTime`, `\svnInfoRevision` a další. Ta již můžeme snadno použít přímo v dokumentu.

```
~/repos/dokument-working-copy/trunk$ svn commit \
> -m 'Přidání Subversion metadat přímo do dokumentu.'
Sending        dokument.tex
Transmitting file data .
Committed revision 7.
~/repos/dokument-working-copy/trunk> grep -m 1 svnInfo dokument.tex
\svnInfo $Id: dokument.tex 7 2011-11-27 18:37:15Z michalr $
```

5.9.2. `svn:ignore`

Zdaleka ne všechny soubory, které se budou v naší pracovní kopii při práci objevovat, budeme chtít také verzovat. Například při překladu TEXového dokumentu vznikají pomocné a výstupní soubory, které do repository nemá smysl vkládat. Verzovat chceme jen zdrojový text dokumentu.

```
~/repos/dokument-working-copy/trunk> ls
dokument.tex
~/repos/dokument-working-copy/trunk> pdflatex dokument.tex
            ...
Output written on dokument.pdf (1 page, 52699 bytes).
Transcript written on dokument.log.
~/repos/dokument-working-copy/trunk$ ls
dokument.aux  dokument.log  dokument.pdf  dokument.tex
```

Pokud se nyní podíváme na stav pracovní kopie, Subversion nám nahlásí tři nově vzniklé soubory, které nejsou pod její správou (příznak `?`).

```
~/repos/dokument-working-copy/trunk$ svn status
?       dokument.aux
?       dokument.log
?       dokument.pdf
```

Bylo by vhodné Subversion říct, že dané soubory má ignorovat. To můžeme provést nastavením property `svn:ignore` na daném adresáři. Hodnotou budou jména souborů (každé na samostatném řádku), které mají být ignorovány (je možno používat i zástupné znaky „?“ a „*“).

```
~/repos/dokument-working-copy/trunk$ svn propedit 'svn:ignore' .
                        ...
            <spuštění textového editoru>
                        ...
dokument.aux
??kument.log
*.pdf
                        ...
            <ukončení textového editoru>
                        ...
Set new value for property 'svn:ignore' on '.'
~/repos/dokument-working-copy/trunk$ svn propget 'svn:ignore' .
dokument.aux
??kument.log
*.pdf
```

Nyní již Subversion zadané soubory ignoruje a ve výpise je nehlásí. Místo toho symbolem `M` ve druhém sloupci oznamuje dosud necommitovanou změnu properties na souboru `.`, tedy na adresáři, ve kterém se momentálně nacházíme.

```
~/repos/dokument-working-copy/trunk$ svn status
 M      .
```

Pokud ignorované soubory přece jen chceme zobrazit, můžeme použít argument `--no-ignore`.

```
~/repos/dokument-working-copy/trunk$ svn status --no-ignore
 M      .
I       dokument.aux
I       dokument.log
I       dokument.pdf
```

Ignorované soubory jsou označeny symbolem `I`.

5.9.3. Automatické nastavování properties

Už ze jména souboru jsme často schopni odhadnout, jaké properties pro něj budeme chtít nastavit. V konfiguračním souboru Subversion klienta (`~/.subversion/config`) proto máme k dispozici volbu `enable-auto-props`, jejíž nastavení na hodnotu `yes` aktivuje automatické nastavování properties podle konfigurace v sekci `[auto-props]` daného konfiguračního souboru. O nastavení properties pro „známé“ soubory se tak nebudeme muset starat, budou nastavovány automaticky Subversion klientem.

Je také pravděpodobné, že určité typy souborů bude chtít implicitně ignorovat (například všechny „*.aux“, „*.log“ a „*.pdf“ soubory). K tomu v konfiguračním souboru slouží volba `global-ignores`.

Pokud si ji nastavíme na hodnotu `global-ignores = *.aux *.log *.pdf`, může v naší repository zrušit explicitní ignoraci zadanou proměnnou `svn:ignore`.

```
~/repos/dokument-working-copy/trunk$ svn propdel 'svn:ignore' .
property 'svn:ignore' deleted from '.'.
~/repos/dokument-working-copy/trunk$ ls
dokument.aux  dokument.log  dokument.pdf  dokument.tex
~/repos/dokument-working-copy/trunk$ svn -u status
Status against revision:      7
```

5.10. Vytváření větví a tagů

Důležitou vlastností systémů pro správu verzí je schopnost vytváření a udržování paralelních vývojových větví. Toto téma si proto v krátkosti také představíme, i když se jedná o již mírně pokročilejší techniku.

5.10.1. Větvení

Představme si v našem modelovém příkladu, že budeme chtít náš dokument převést na použití XeLaTeXu. Protože ale nemáme se XeTeXem žádné zkušenosti, a nevíme, jestli se převod povede a zda bude vše fungovat, máme obavu provést tuto úpravu přímo v hlavní vývojové větvi – zatímco budeme zkoumat možnosti XeTeXu, chceme mít pořád možnost dále upravovat náš LaTeXový dokument a mít jej k dispozici v otestovaném formátu. Toto je ideální situace pro vytvoření zvláštní vývojové větve.

XeLaTeXovou větev vytvoříme v podadresáři v `branches/`. Vytvoření větve je velice jednoduché – zkopírujeme náš hlavní vývojový strom do adresáře `branches/xelatex-test/`. Nezapomeneme přitom předem provést update celé pracovní kopie, abychom kopírovali aktuální verzi `trunk/`. Také by neměly být přítomny žádné lokální (to znamená dosud necommitované) změny.[17]

```
~/repos/dokument-working-copy/trunk$ cd ..
~/repos/dokument-working-copy$ svn update
Updating '.':
At revision 7.
~/repos/dokument-working-copy$ svn copy trunk/ branches/xelatex-test
A         branches/xelatex-test
~/repos/dokument-working-copy$ svn commit \
> -m 'Vytvoření větve pro testování XeLaTeXu.'
Adding         branches/xelatex-test
```

[17] Ještě čistějším řešením by bylo provést kopii na úrovni repository příkazem
`svn copy file://$HOME/repos/dokument-repository/trunk/ \`
`> file://$HOME/repos/dokument-repository/branches/xelatex-test \`
`> -m 'Vytvoření větve pro testování XeLaTeXu.'`.

```
Committed revision 8.
```

Jak vidíte, větev není nic jiného než kopie adresářového podstromu. Větev je to jen proto, že my jsme se dohodli, že se jedná o vývojovou větev hlavního vývojového stromu a dle toho bude s tímto adresářovým podstromem zacházet. Subversion nevyžaduje žádné explicitní oznámení, že jsme provedli větvení.

Vytvoření větve (= kopírování adresářového podstromu) je nenáročná operace. Jak bylo uvedeno výše, fyzicky se v repository ukládají jen změny. V tomto případě tedy bude jen poznamenáno, že `branches/xelatex-test/` ukazuje na stejné soubory jako `trunk/` v revizi 7.

```
~/repos/dokument-working-copy$ cd branches/xelatex-test/
~/repos/dokument-working-copy/branches/xelatex-test$ ls
dokument.aux  dokument.log  dokument.pdf  dokument.tex
```

V naší pracovní kopii pochopitelně došlo k fyzickému zkopírování celého adresářového podstromu, a to včetně souborů, které nejsou pod správou Subversion (zde `dokument.{aux,log,pdf}`).

Nyní můžeme bez obav provádět rozsáhlejší úpravy zdrojového textu.

```
~/repos/dokument-working-copy/branches/xelatex-test$ vim dokument.tex
            ...
~/repos/dokument-working-copy/branches/xelatex-test$ cat dokument.tex
%&program=xelatex
%&encoding=UTF-8 Unicode
% $Id: dokument.tex 8 2009-12-31 15:09:53Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{fontspec}
\usepackage{xunicode}
\usepackage{xltxtra}
\usepackage[nofancy]{svninfo} % Balíček pro práci se Subversion
      % metadaty v dokumentu (s parametrem zakazujícím automatické
      % vložení Subversion metadat do zápatí).
\svnInfo $Id: dokument.tex 8 2009-12-31 15:09:53Z michalr $

\def\uv#1{„#1"}
\defaultfontfeatures{Scale=MatchLowercase, Mapping=tex-text}
\setromanfont{DejaVu Serif}
\setsansfont{DejaVu Sans}
\setmonofont{DejaVu Sans Mono}

\begin{document}
\hyphenation{Open-Of-fice UNESCO}
```

```
Ahoj světe! Tento dokument je sázen \XeTeX em.

% Vložení Subversion metadat do výsledného dokumentu.
(Author: \svnInfoOwner, datum: \svnInfoDate{} \svnInfoTime{},
 revize:~\svnInfoRevision)

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char‘\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Text vložíme do \uv{uvozovek} příkazem \verb|\uv| či přímým zápisem
znaků „ a ".

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

Převedení dokumentu na XeLaTeX se povedlo, můžeme tedy provést commit.

```
~/repos/dokument-working-copy/branches/xelatex-test$ svn commit \
> dokument.tex -m 'Přechod na XeLaTeX.'
Sending        dokument.tex
Transmitting file data .
Committed revision 9.
```

Zatímco zkoumáme možnosti XeLaTeXu, v hlavní vývojové větvi v `trunk/` mohou probíhat úpravy původního LaTeXového dokumentu. Odstraníme například nepoužité pokyny pro dělení slov, upravíme sazbu Subversion metadat a přidáme další řádek textu.

```
~/repos/dokument-working-copy/branches/xelatex-test$ cd ../../trunk/
~/repos/dokument-working-copy/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy/trunk$ svn commit dokument.tex \
> -m 'Odstranění \hyphenation{...}.'
Sending        dokument.tex
Transmitting file data .
Committed revision 10.
~/repos/dokument-working-copy/trunk$ vim dokument.tex
            ...
~/repos/dokument-working-copy/trunk$ svn commit dokument.tex \
> -m 'Úprava sazby Subversion metadat.'
Sending        dokument.tex
Transmitting file data .
Committed revision 11.
~/repos/dokument-working-copy/trunk$ vim dokument.tex
```

```
            ...
~/repos/dokument-working-copy/trunk$ svn commit dokument.tex \
> -m 'Přidání dalšího kousku textu.'
Sending        dokument.tex
Transmitting file data .
Committed revision 12.
~/repos/dokument-working-copy/trunk$ cat dokument.tex
% $Id: dokument.tex 12 2011-11-27 18:53:22Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[nofancy]{svninfo} % Balíček pro práci se Subversion
      % metadaty v dokumentu (s parametrem zakazujícím automatické
      % vložení Subversion metadat do zápatí).
\svnInfo $Id: dokument.tex 12 2011-11-27 18:53:22Z michalr $

\begin{document}
Ahoj světe!

% Vložení Subversion metadat do výsledného dokumentu.
(Author: \svnInfoOwner, revize:~\svnInfoRevision)

V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char`\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Text vložíme do \uv{uvozovek} příkazem "`\verb|\uv|"', případně
dvojicí "`\verb|"`|"' a~"`\verb|"`|"'. Vložení spojovníku se
správným českým chováním bez uvádění globálního nastavení
"`\verb|\languageattribute{czech}{split}|"' či užití příkazů
"`\verb|\splithyphens|"' a~"`\verb|\standardhyphens|"' dosáhneme
vložením "`\verb|"=|"'.

Šíleně žluťoučký kůň úpěl ďábelské ódy.
Další řádek textu.
\end{document}
```

Úpravy v hlavním vývojovém stromu je velmi vhodné průběžně začleňovat i do naší vývojové větve. Zabráníme tak tomu, aby se obě verze dokumentu od sebe obsahově příliš vzdálily. Pokud by k tomu došlo, mohli bychom později obtížně slučovat velké množství rozdílů v obou větvích při zpětném začleňování naší úpravy (to znamená přechod na XeLaTeX) do `trunk/`.

Opět chceme pracovat s aktuální verzí pracovní kopie bez lokálních změn.

```
~/repos/dokument-working-copy/trunk$ cd ..
~/repos/dokument-working-copy$ svn update
Updating '.':
At revision 12.
~/repos/dokument-working-copy$ cd branches/xelatex-test/
```

K začleňování sad změn mezi větvemi slouží podpříkaz `merge` klienta Subversion, který jsme již dříve použili k odčinění nechtěné změny, která již byla commitnuta do repository. Nyní ho použijeme k začlenění změn, které se odehrály v hlavním vývojovém stromu od odštěpení naší vývojové větve.

```
~/repos/dokument-working-copy/branches/xelatex-test$ svn merge '^/trunk'
--- Merging r8 through r12 into '.':
U    dokument.tex
--- Recording mergeinfo for merge of r8 through r12 into '.':
 U   .
```

Syntax se znakem `^` je zkratkou pro URL kořenového adresáře repository. Příznak `U` nám oznamuje, že došlo k bezproblémovému začlenění změn.

```
~/repos/dokument-working-copy/branches/xelatex-test$ cat dokument.tex
%&program=xelatex
%&encoding=UTF-8 Unicode
% $Id: dokument.tex 9 2011-11-27 18:50:03Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{fontspec}
\usepackage{xunicode}
\usepackage{xltxtra}
\usepackage[nofancy]{svninfo} % Balíček pro práci se Subversion
      % metadaty v dokumentu (s parametrem zakazujícím automatické
      % vložení Subversion metadat do zápatí).
\svnInfo $Id: dokument.tex 9 2011-11-27 18:50:03Z michalr $

\def\uv#1{„#1"}
\defaultfontfeatures{Scale=MatchLowercase, Mapping=tex-text}
\setromanfont{DejaVu Serif}
\setsansfont{DejaVu Sans}
\setmonofont{DejaVu Sans Mono}

\begin{document}
Ahoj světe! Tento dokument je sázen \XeTeX em.

% Vložení Subversion metadat do výsledného dokumentu.
(Author: \svnInfoOwner, revize:~\svnInfoRevision)
```

```
V~této větě byla použita nezlomitelná mezera -- do zdrojového textu
se vkládá jako znak tilda (\char‘\~). K~práci s~nezlomitelnými
mezerami slouží program Vlna.

Nový odstavec se vytvoří vynecháním řádku.

Text vložíme do \uv{uvozovek} příkazem \verb|\uv| či přímým zápisem
znaků „ a ".

Šíleně žluťoučký kůň úpěl ďábelské ódy.
Další řádek textu.
\end{document}
```

Poznámku o tom, které změny z hlavního vývojového stromu již byly začleněny, si Subversion uložila do speciální property `svn:mergeinfo`. Tato proměnná se ručně neupravuje, je modifikována automaticky a je důležitá pro uchování informace o sloučených a nesloučených rozdílech mezi větvemi.

```
~/repos/dokument-working-copy/branches/xelatex-test$ svn status
 M       .
M        dokument.tex
~/repos/dokument-working-copy/branches/xelatex-test$ svn proplist .
Properties on '.':
  svn:mergeinfo
```

Větve byly sloučeny pouze v pracovní kopii. Teprve když si ověříme, že sloučení proběhlo v pořádku, můžeme výsledek odeslat do repository.

```
~/repos/dokument-working-copy/branches/xelatex-test$ svn commit \
> -m 'Synchronizace s hlavní vývojovou větví.'
Sending        .
Sending        dokument.tex
Transmitting file data .
Committed revision 13.
```

Nyní můžeme pokračovat v dalším vývoji našeho XeLaTeXového dokumentu s vědomím, že se příliš nerozejdeme s hlavním vývojovým stromem. Náš dokument již dokončíme...

```
~/repos/dokument-working-copy/branches/xelatex-test$ vim dokument.tex
            ...
~/repos/dokument-working-copy/branches/xelatex-test$ cat dokument.tex
%&program=xelatex
%&encoding=UTF-8 Unicode
% $Id: dokument.tex 13 2009-12-31 15:49:09Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{fontspec}
```

```
\usepackage{xunicode}
\usepackage{xltxtra}
\usepackage[nofancy]{svninfo} % Balíček pro práci se Subversion
      % metadaty v dokumentu (s parametrem zakazujícím automatické
      % vložení Subversion metadat do zápatí).
\svnInfo $Id: dokument.tex 13 2009-12-31 15:49:09Z michalr $

\def\uv#1{„#1"}
\defaultfontfeatures{Scale=MatchLowercase, Mapping=tex-text}
\setromanfont{DejaVu Serif}
\setsansfont{DejaVu Sans}
\setmonofont{DejaVu Sans Mono}

\begin{document}
% Vložení Subversion metadat do výsledného dokumentu.
(Author: \svnInfoOwner, revize:~\svnInfoRevision)

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

. . . a odešleme do repository.

```
~/repos/dokument-working-copy/branches/xelatex-test$ svn commit \
> -m 'Přechod na XeTeX dokončen.'
Sending        dokument.tex
Transmitting file data .
Committed revision 14.
```

Po úspěšném završení vývoje v XeLaTeXové větvi nastal čas pro zpětné začlenění upraveného kódu do hlavní vývojové větve `trunk/`. Nejdříve ověříme, že v `trunk/` nedošlo k dalším úpravám, které by bylo třeba začlenit. Provedeme finální synchronizaci naší XeLaTeXové větve s hlavním vývojovým stromem.

```
~/repos/dokument-working-copy/branches/xelatex-test$ svn update ../..
Updating '/home/michalr/repos/dokument-working-copy':
At revision 14.
~/repos/dokument-working-copy/branches/xelatex-test$ svn merge '^/trunk'
--- Recording mergeinfo for merge of r13 through r14 into '.':
 U   .
~/repos/dokument-working-copy/branches/xelatex-test$ svn status
 M      .
~/repos/dokument-working-copy/branches/xelatex-test$ svn commit \
> -m 'Závěrečná synchronizace XeLaTeXové větve.'
Sending        .

Committed revision 15.
```

Nyní můžeme přistoupit k integraci naší XeLaTeXové varianty dokumentu do

hlavní vývojové větve `trunk/`.

```
~/repos/dokument-working-copy/branches/xelatex-test$ cd ../..
~/repos/dokument-working-copy$ svn update
Updating '.':
At revision 15.
~/repos/dokument-working-copy$ cd trunk/
~/repos/dokument-working-copy/trunk$ svn merge --reintegrate \
> '^/branches/xelatex-test'
--- Merging differences between repository URLs into '.':
U    dokument.tex
--- Recording mergeinfo for merge between repository URLs into '.':
 U   .
~/repos/dokument-working-copy/trunk$ svn status
 M      .
M       dokument.tex
```

K integraci (včetně automatického poznačení této informace v property `svn:mergeinfo`) opět došlo pouze v naší lokální pracovní kopii.

```
~/repos/dokument-working-copy/trunk$ cat dokument.tex
%&program=xelatex
%&encoding=UTF-8 Unicode
% $Id: dokument.tex 12 2011-11-27 18:53:22Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{fontspec}
\usepackage{xunicode}
\usepackage{xltxtra}
\usepackage[nofancy]{svninfo} % Balíček pro práci se Subversion
      % metadaty v dokumentu (s parametrem zakazujícím automatické
      % vložení Subversion metadat do zápatí).
\svnInfo $Id: dokument.tex 12 2011-11-27 18:53:22Z michalr $

\def\uv#1{„#1"}
\defaultfontfeatures{Scale=MatchLowercase, Mapping=tex-text}
\setromanfont{DejaVu Serif}
\setsansfont{DejaVu Sans}
\setmonofont{DejaVu Sans Mono}

\begin{document}
% Vložení Subversion metadat do výsledného dokumentu.
(Author: \svnInfoOwner, revize:~\svnInfoRevision)

Šíleně žluťoučký kůň úpěl ďábelské ódy.
\end{document}
```

Teprve když si ověříme, že vše proběhlo v pořádku, odešleme změny do repository.

```
~/repos/dokument-working-copy/trunk$ svn commit \
> -m 'Přechod na XeLaTeX v hlavní vývojové větvi.'
Sending        .
Sending        dokument.tex
Transmitting file data .
Committed revision 16.
```

Jakmile byla vývojová větev jednou integrována zpět do hlavního stromu, už nemůže být použita pro další vývoj. Můžeme ji tedy odstranit.[18]

```
~/repos/dokument-working-copy/trunk$ cd ..
~/repos/dokument-working-copy$ svn rm branches/xelatex-test/
D         branches/xelatex-test
D         branches/xelatex-test/dokument.tex
~/repos/dokument-working-copy$ svn commit \
> -m 'Odstranění již začleněné XeLaTeXové větve.'
Deleting       branches/xelatex-test

Committed revision 17.
~/repos/dokument-working-copy$ svn update
Updating '.':
At revision 17.
```

I když jsme větev smazali, znamená to samozřejmě pouze to, že již není v poslední revizi našeho projektu. Vždy můžeme sáhnout po některé starší revizi, ve které tato větev existovala, a data si opět stáhnout. Žádná data nikdy nemizí nenávratně – proto ostatně systém pro správu verzí používáme.

Předvedli jsme si jen úplně nejzákladnější práci s větvemi, což však odpovídá účelu tohoto článku. Podrobné informace naleznete v Subversion book, kde je větvení věnována celá obsáhlá kapitola.

5.10.2. Vytváření tagů

V souvislosti s prací s větvemi je vhodné zmínit ještě jeden související postup – vytváření tagů. Tag není nic jiného, než větev, ve které po jejím vytvoření nejsou prováděny žádné další úpravy. Opět platí, že označení kopie adresářového podstromu za tag je čistě naše uživatelské rozhodnutí, Subversion nebude vyžadovat, abychom takovou kopii za tag nějak explicitně označili, stejně tak nám nebude bránit tag jakkoli modifikovat.

Tagy se typicky používají jako „pojmenování“ některých význačných revizí. Pokud například vytvoříme stabilní verzi programu, kterou chceme publikovat

[18] Pokud bychom chtěli ve vývoji pokračovat, jednoduše bychom si vytvořili novou větev (klidně i stejně pojmenovanou).

široké veřejnosti jako verzi 1.0, zkopírujeme danou revizi hlavního vývojového stromu do adresáře `tags/1.0.0/`. Protože je vytváření větví v Subversion „levné“ (ukládají se jen rozdíly), můžeme větve i tagy vytvářet dle potřeby bez negativních dopadů na výkon nebo obsazené místo na disku na straně repository. Máme přitom snadno k dispozici zdrojové texty dané verze programu. Pokud nás zajímají zdrojové texty verze 1.0, můžeme sáhnout po obsahu `tags/1.0.0/` místo toho, abychom prohledávali log `trunk/` a hledali v záplavě revizí, která z nich má být považována za verzi 1.0.[19] (Samozřejmě nám ale ani v tomto postupu nikdo bránit nebude, data budou identická, neboť podstrom `tags/1.0.0/` vznikl jako kopie dané revize a dohodli jsme se, že se jedná o tag, to znamená, že do této kopie už nikdo nebude provádět žádné úpravy.[20])

Jelikož jsme dokončili vývoj našeho ukázkového dokumentu, vytvoříme si tag s jeho konečnou verzí.

```
~/repos/dokument-working-copy$ svn copy trunk/ tags/finalni-verze
A         tags/finalni-verze
```

Až dosud jsme (naprosto správně) verzovali jen čisté zdrojové texty. S textovými soubory Subversion pracuje nejlépe a šetříme tím místo v repository. Pokud však nejsme výrazně omezeni datovým prostorem, můžeme se v případě tagů rozhodnout do repository uložit i přeloženou verzi dokumentu. Výhodou je, že i kdykoliv v budoucnu budeme mít vzor, jak má přeložený dokument vypadat, což může být zejména u LaTeXových dokumentů (využívajících externí balíčky, které neustále prochází bouřlivým vývojem) v delším časovém horizontu nesporná výhoda.

```
~/repos/dokument-working-copy$ cd tags/finalni-verze/
~/repos/dokument-working-copy/tags/finalni-verze$ xelatex dokument.tex
            ...
Output written on dokument.pdf (1 page).
Transcript written on dokument.log.
~/repos/dokument-working-copy/tags/finalni-verze$ svn add *
A         dokument.aux
A         dokument.log
A         dokument.pdf
svn: warning: W150002: '/home/michalr/repos/dokument-working-copy/tags/
                       finalni-verze/dokument.tex' is already under
                       version control
svn: E200009: Could not add all targets because some targets are already
              versioned
svn: E200009: Illegal target for the requested operation
```

[19] Z tohoto by také mělo být patrné, jak je důležité volit vhodné komentáře všech commitů.

[20] Subversion by provedení úprav do tagu nijak nebránila, z logu by však bylo patrné, že k nim došlo. A opět bychom mohli sáhnout po starší revizi dat bez těchto zásahů. Zabránit dodatečné modifikaci tagů je možné s použitím mechanismu háčků (vizte Subversion book).

Při pokusu o přidání souboru `dokument.tex`, který již je pod správou Subversion, jsme varováni. Přidání ostatních souborů je však provedeno korektně. Můžeme tedy provést závěrečný commit.

```
~/repos/dokument-working-copy/tags/finalni-verze$ cd ../..
~/repos/dokument-working-copy$ svn commit \
> -m 'Vytvoření tagu s konečnou verzí dokumentu.'
Adding         tags/finalni-verze
Adding         tags/finalni-verze/dokument.aux
Adding         tags/finalni-verze/dokument.log
Adding         tags/finalni-verze/dokument.pdf
Transmitting file data ...
Committed revision 18.
```

Všechna data jsou nyní v repository a pracovní kopii již nepotřebujeme. Můžeme ji proto smazat.

```
~/repos/dokument-working-copy$ cd ..
~/repos$ rm -rf dokument-working-copy/
```

5.11. Export dat

Pokud budeme kdykoliv v budoucnu chtít data opět upravovat, vytvoříme si novou pracovní kopii příkazem `svn checkout`. Pokud chceme jen přečíst data z repository, můžeme použít příkaz `svn export`, který do výstupů nepřidává adresář `.svn` se Subversion metadaty. Opět platí, že můžeme pracovat s libovolnou revizí, a jsme tak schopni snadno získat i staré verze dokumentu.

Pokud chceme získat původní LaTeXovou verzi našeho dokumentu před přechodem na XeLaTeX, nahlédneme do logu hlavní vývojové větve naší repository.

```
~/repos$ svn log \
> file://$HOME/repos/dokument-repository/trunk/ | head
------------------------------------------------------------------------
r16 | michalr | 2011-11-27 20:04:54 +0100 (Ne, 27 lis 2011) | 1 line

Přechod na XeLaTeX v hlavní vývojové větvi.
------------------------------------------------------------------------
r12 | michalr | 2011-11-27 19:53:22 +0100 (Ne, 27 lis 2011) | 1 line

Přidání dalšího kousku textu.
------------------------------------------------------------------------
r11 | michalr | 2011-11-27 19:52:40 +0100 (Ne, 27 lis 2011) | 1 line
```

Z logu je patrné, že si máme stáhnout dokument z revize č. 12.[21]

[21] Revize 13–15 nejsou v logu uvedeny, což je v pořádku, neboť vypisujeme log podstromu `trunk/`, ve kterém v těchto revizích nedošlo k žádným změnám.

```
~/repos/$ svn export -r12 \
> file://$HOME/repos/dokument-repository/trunk/ latex-dokument
A    latex-dokument
A    latex-dokument/dokument.tex
Exported revision 12.
```

Na začátku vyexportovaného zdrojového textu nechybí identifikace, kterou jsme si dříve vyžádali nastavením property `svn:keywords` na tomto souboru.

```
~/repos/dokument-working-copy$ head -n 3 latex-dokument/dokument.tex
% $Id: dokument.tex 12 2011-11-27 18:53:22Z michalr $
\documentclass{article}
\usepackage[czech]{babel}
```

5.12. Zálohování

Pokud pomineme lokální změny v pracovních kopiích před jejich commitem do repository, je právě repository jediná sada souborů, kterou musíme zálohovat. Nejjednodušším způsobem je prosté zálohování adresáře s repository, v našem modelovém příkladu to znamená adresáře `~/repos/dokument-repository/`. Autor článku se nesetkal s problémy ani při přenosu tohoto adresáře mezi různými operačními systémy, kdy bylo po přesunu možné běžným způsobem provést checkout pracovní kopie a pokračovat ve vývoji.

V průběhu zálohování adresáře repository by nemělo dojít ke commitu, který by repository změnil – výsledkem by mohla být nekonzistentní, a tudíž nepoužitelná záloha. Pokud toto nemůžeme zajistit (například se jedná o veřejnou repository, kterou nechceme po dobu zálohování znepřístupnit), můžeme pro vytvoření kopie repository použít podpříkaz `hotcopy` nástroje `svnadmin`, který zajistí vytvoření konzistentní kopie i používaného repositáře.

Ačkoliv formát Subversion repository již delší dobu zůstává kompatibilní, teoreticky hrozí, že by některá budoucí verze Subversion nemusela být schopna pracovat s repository vytvořenou některou starší verzí.[22] Obsah celé repository je proto možné vyexportovat podpříkazem `dump` nástroje `svnadmin` ve formátu zaručeně nezávislém na platformě a na verzi Subversion.[23] Znovu načíst data z dump streamu je možné podpříkazem `load` nástroje `svnadmin`. Tento způsob zálohování může být i inkrementální a poskytuje i některé další možnosti pro manipulaci s obsahem repository.

Export a import pomocí `svnadmin dump`/`load` vyžaduje uživatelský účet a odpovídající přístupová práva k systému souborů na počítači, kde je repository

[22] Od Subversion verze 1.7 se například výrazně změnil formát metadat pracovní kopie, který byl jednou z nejstarších částí programového kódu Subversion.

[23] Jedná se o formát podobný RFC 822. I když se může zdát, že se jedná o čistě textový formát, obsahuje binární data. Není proto možné upravovat ho běžnými textovými editory bez vážného nebezpečí poškození.

fyzicky umístěna. To může být poněkud nepraktické, pokud si chce oprávněný uživatel repositáře vytvořit kompletní zálohu historie vývoje a tento přístup nemá k dispozici. Od Subversion verze 1.7 je proto k dispozici klientský program `svnrdump`, který umožňuje provést dump i load repositáře obdobně jako program `svnadmin`, ovšem vzdáleně – po připojení stejným způsobem jako běžný klient `svn` (vizte `svnrdump --help` nebo Subversion book).

6. Závěr

Ukázali jsme si, že použití nástroje Subversion je poměrně intuitivní a uživatele při práci zbytečně neomezuje. Naopak, při vývoji mu podává pomocnou ruku. Pomocí `svn add` určíme soubory, které chceme verzovat, a pomocí `svn copy`, `svn move`, `svn delete` a `svn mkdir` s nimi pracujeme obdobným způsobem, jako bez systému pro správu verzí. Spuštění `svn status` nám dá hrubý přehled o stavu naší pracovní kopie, `svn log` nám poskytne informace o vývoji projektu v čase. Zadáním `svn diff` můžeme zjistit podrobnosti o změnách, které jsme právě provedli v naší aktuální pracovní kopii, anebo kdykoliv dříve v celé historii projektu. Pokud se ukáže, že jsme se vydali špatnou cestou, jednoduché `svn revert` nás vrátí do funkčního stavu. Jednou odladěná funkční verze projektu je po zadání `svn commit` kdykoliv dostupná v nezměněné podobě, a může být použita k dohledání příčiny záhadné chyby odhalené po několika dnech, ale i k opětovnému vysázení původní podoby dokumentu po mnoha letech vývoje použitých makrobalíků.

Autor doufá, že článek byl srozumitelným a názorným předvedením použití Subversion při vývoji TEXového dokumentu, a přispěje tak k rozšíření skupiny uživatelů systémů pro zprávu verzí i o dosud váhající autory TEXových dokumentů, kterým tak snad pomůže zlepšit jejich efektivitu při práci s TEXem.

Reference

[1] Collins-Sussman, Ben; Fitzpatrick, Brian W.; Pilato, C. Michael. *Version Control with Subversion : For Subversion 1.6* [online]. (Compiled from r3654). c2002–2009 [cit. 2009-12-30]. Dostupný z WWW: `http://svnbook.red-bean.com/nightly/en/svn-book.html`.

Summary: Version Control of TEX Documents Using Subversion

This article presents advantages of using Subversion revision control system in conjunction with development of TEX documents and macro packages. Basic

principles of Subversion are described and step-by-step tutorial follows.

Masarykova univerzita, Fakulta informatiky,
Botanická 68a, 602 00 Brno,
mruzicka@mail.muni.cz