Univerzita Karlova

Pedagogická fakulta

Katedra informačních technologií a technické výchovy (41-KITTV)

BAKALÁŘSKÁ PRÁCE

Využití verzovacích systémů pro podporu výuky

Using Version Control Systems to Support Learning

Milan Pokorný

Vedoucí práce: PhDr. Tomáš Jeřábek, Ph.D.

Studijní program: Specializace v pedagogice (B IT)

Studijní obor: B IT (7507R040)

2023

Odevzdáním této bakalářské práce na téma “Využití verzovacích systémů pro podporu výuky” potvrzuji, že jsem ji vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

Praha 8. 7. 2023

Děkuji PhDr. Tomáši Jeřábkovi, Ph.D. za pomoc při vedení bakalářské práce. Mé poděkováni patři též PaedDr. Evě Battistové a Mgr. Pavle Cibulkové za výbornou spolupráci v průběhu studia.

ABSTRAKT

Práce se věnuje verzovacím systémům a možnostem jejich využití ve výuce Informatiky. Na začátku práce jsou posouzena různá řešení pro správu verzí. Jako nejvhodnější je pro použití ve výuce vybrán distribuovaný systém Git, jehož principy jsou detailně popsány. Práce se dále zaměřuje na představení doplňků systému Git, ať už se jedná o výukové programy či přímo rozšiřující moduly. V další kapitole se text zaměřuje na roli verzovacích systémů ve vzdělávání, jejich výuku a použití ve smyslu didaktického prostředku. Navazující část práce je již prakticky zaměřená a soustředí se na popis postupů, které by umožnily zahrnout použití verzovacího systému v rámci třídy při výuce programování. V závěru text shrnuje pozitivní a negativní aspekty začlenění systémů pro správu verzí do výuky a podává praktická doporučení, včetně modelových řešení.

KLÍČOVÁ SLOVA

Systém pro správu verzí, výuka programování, informační technologie, systém Git

ABSTRACT

The thesis examines version control systems (VCs) and the possibilities of their use in information technology classes. At first, various solutions for version management are evaluated. As a result, a decentralized system called Git is deemed the most appropriate to be used in education. The text further explains the details of how Git works while also introducing some of the relevant training software and Git extensions. The thesis continues by exploring the role of version control systems in education, their teaching, and use as a didactic tool. The final chapters are practically oriented and include the description of processes which allow source code version control to be used within a student group environment. The thesis concludes with an overall evaluation of the use of version management systems in teaching, including practical recommendations and examples.

KEYWORDS

Version control system, teaching programming, information technology, Git

Obsah

1 Úvod

Téměř každý aktivní uživatel osobního počítače při své práci časem narazí na situaci, kdy potřebuje upravit vybraný soubor (nejčastěji textový), ale zároveň by si „pro jistotu“ rád uchoval i jeho původní verzi. Takovou situaci je možné řešit různými způsoby. Pokud postačuje návrat k předchozím verzím v omezeném rozsahu a použitý program to umožňuje, lze využít krokovacích funkcí „zpět“ (*undo*) a „opakovat“ (*redo*).

Nevýhody krokování jsou nasnadě – ačkoliv pokročilejší implementace této funkce umožňují posun i o několik kroků naráz, stále je krokování vhodné spíše jako pracovní pomůcka při přímé editaci textu. Chceme-li si uchovat verzi textu jako celku, tedy vytvořit jakýsi záchytný bod, který uloží stav textu k určitému času nebo události, je možné začít tvořit kopie souborů. Častou praxí je označovat si soubory dle čísla vytvořené verze („soubor_v1.txt“, „soubor_v2.txt“ atp.).

Tyto techniky mohou pomoci při samostatné práci s jedním souborem a při důsledném přístupu s nimi při běžné kancelářské práci lze i vystačit. Pokud je nicméně nutná spolupráce více tvůrců nad jedním či více soubory, situace se velmi rychle zkomplikuje a udržet přehled nad různými verzemi souborů je takřka nemožné. A to často i v případě užití pomůcek pro sledování úprav v textových editorech a vhodnějšího způsobu sdílení, než je přeposílání pomocí elektronické pošty.

I při tvorbě počítačových (zdrojových) kódů se všemi jejich specifiky, je zřejmé, že by programování usnadnil systém, který by zajišťoval správu verzí na pokročilejší úrovni. Nejpatrnější tato potřeba bude při spolupráci více lidí na jednom projektu, ať už jde o vývojový tým nebo skupinu studentů ve třídě. Právě k tomu mohou dobře posloužit systémy pro správu verzí, jejichž vznik se datuje již do 70. let 20. století (Rochkind, 1975).

Používání systémů pro správu verzí je běžnou záležitostí u společností zabývajících se IT projekty. Již to může být relevantním důvodem, proč s nimi studenty naučit pracovat. Absence použití systémů pro sledování změn může mít v případě rozsáhlejších projektů závažné následky – jako příklad lze uvést pád sociální sítě MySpace, jejíž obtíže komentátoři zčásti připisovali právě chybějícímu systému pro sledování změn ve vývojovém cyklu této webové aplikace (Petrazickis, 2011).

2 Cíl a metody práce

Hlavním cílem této práce je prozkoumat oblast verzovacích systémů z hlediska možnosti využití těchto systémů ve vzdělávání se zvláštním zaměřením na nejpoužívanější systém pro správu verzí, Git. Na základě této analýzy je pak dílčím cílem navrhnout a představit vhodný model pro aplikaci verzovacího systému ve výuce v podobě podpůrného nástroje pro výuku Informatiky. Součástí modelu budou i podrobné instrukce, příklady a tipy, které vyučujícím usnadní implementaci verzovacího systému do výuky a posílí jejich schopnosti předávat tuto znalost studentům.

Součástí práce bude rovněž ukázkové řešení výuky s využitím Git, využívající navržený model, konkrétní scénáře, příklady a praktické tipy. Tato ukázka bude sloužit jako inspirace pro vyučující, kteří budou chtít implementovat verzovací systémy do svých výukových plánů. Na závěr práce budou prezentovány vlastní zjištění a doporučení pro úspěšnou výuku s Gitem.

Dílčím cílem práce je vytvořit rešerši nejvhodnějších doplňků pro systém Git. Tato analýza poskytne přehled o existujících softwarových řešeních, která mohou ulehčit výuku práce se systémem Git. Budou představeny specifické doplňkové aplikace sloužící k efektivnímu ovládání samotného systému i aplikace podpůrné, které poskytují další nástroje a prostředky pro výuku s využitím Gitu.

3 Verzovací systémy

Jako systémy pro správu verzí jsou v prostředí softwarového vývoje označována řešení, která jsou určena nejčastěji k uchování postupných změn zdrojového kódu. Obecně však lze pomocí těchto systémů sledovat změny nejen zdrojového kódu, ale v zásadě libovolných souborů (Scott a Straub, 2022).

Mezi hlavní výhody užití verzovacích systémů v rámci vývoje softwaru patří:

- sledování změn; možnost se libovolně vracet zpět k dřívějším verzím (uchování kompletní historie),
- usnadnění práce více lidí nad jedním společným a postupně se rozvíjejícím řešením,
- snazší dohledávání detailů jednotlivých změn (např. kdo a za jakým účelem změnu provedl),
- řízení přístupu k různým částem vyvíjeného řešení.

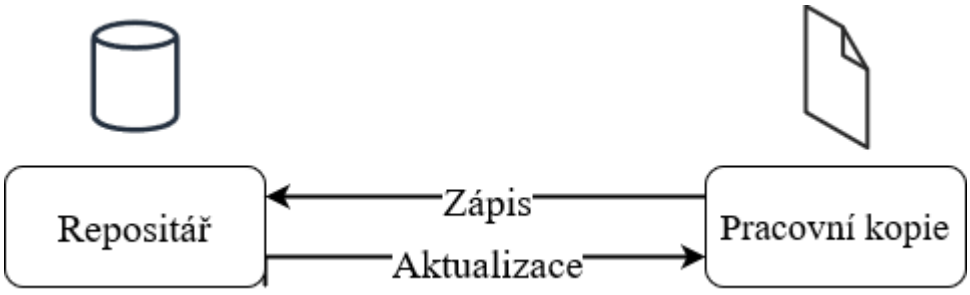
Systémy pro správu verzí mohou mít různou architekturu, která ale většinou obsahuje společné prvky. K lepšímu porozumění problematiky správy verzí je přínosné se seznámit s následujícími pojmy, které popisují základní prvky verzovacího systému (AbcLinuxu, 2009):

Repozitář (*repository)* – datové úložiště („databáze změn“), které je spravováno verzovacím systémem. Repozitář bývá realizován prostřednictvím serveru nebo může být lokální. Uchovává informace o všech verzích zdrojového kódu či jiných dat, která jsou systémem spravována.

Pracovní kopie (*working copy*) – jde o kopii repozitáře, která je určena k postupným úpravám. Tyto průběžné změny provedené v pracovní kopii se do repozitáře automaticky nezapisují a sledovány jsou pouze na lokální úrovni. Po dosažení cílového stavu změn (dokončení dílčího pracovního úkolu) lze dát pokyn k zápisu dat, který přenese změny provedené v pracovní kopii do repozitáře. Z hlediska souborového systému se pracovní kopie prezentuje jako běžný adresář.

Zápis (*commit*) – operace, která zapíše data do repozitáře. Součástí zápisu by měl být smysluplný popis (objasnění) změn, které daný zápis v repozitáři způsobí. Údaje o každém zápisu se ukládají tak, aby v repozitáři byla udržována kompletní historie změn.

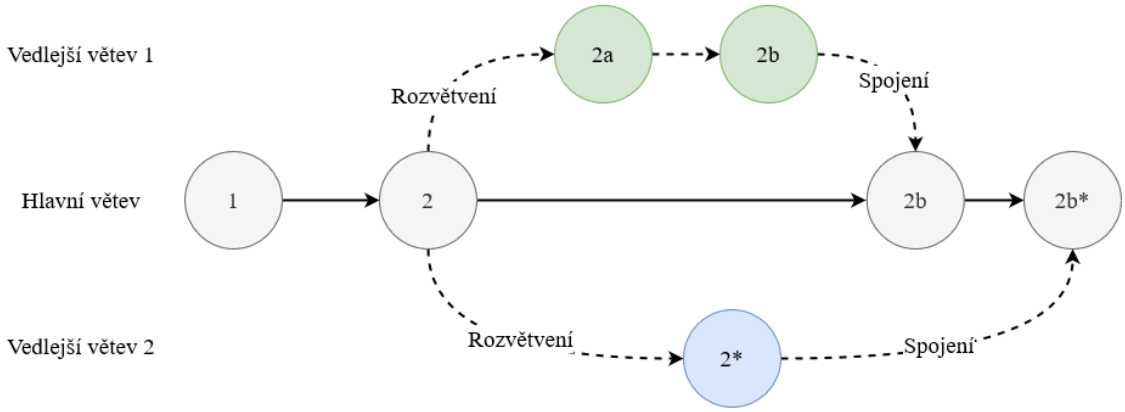
Aktualizace (*update*) – operace, která propíše aktuální stav dat repozitáře do dat pracovní kopie (obr. 1).



Obr. 1: Základní schéma fungování obecného verzovacího systému (podle Ernst, 2022)

Důležitou vlastností moderních systémů pro správu verzí je možnost vytváření tzv. **větví (*branches*)**. Tradiční přístup k verzování zachovává lineární posloupnost změn, při kterém jedna změna následuje druhou až poté, co je ta první plně aplikována.

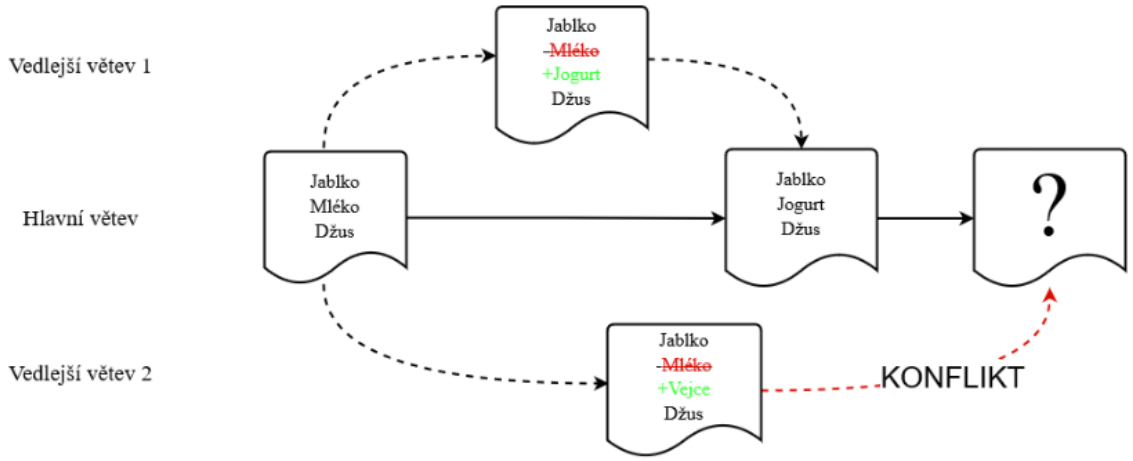
Pomocí větvení je docíleno odlišného chování – od určitého okamžiku lze vývoj rozdělit a dva či více různých uživatelů se mohou každý věnovat práci se svou vlastní pracovní kopií (větví, *branch*) (obr. 2). Pokud je třeba, uživatelé následně svoji dílčí větev s dokončenou úpravou opětovně sloučí do společného celku – větve, která je všem společná (označována často jako hlavní, *master, main* či *trunk*).



Obr. 2: Schématický příklad postupného vývoje hlavní větve, kdy poslední stav hlavní větve (zcela napravo) zahrnuje změnová data z obou oddělených vedlejších větví 1 a 2

Tvůrci tak mohou pracovat nezávisle na sobě, a přesto tvořit jedno společné řešení. Pokud uživatelé provádějí změny každý v jiné části dat, proběhne spojení do společného celku v moderním verzovacím systému z velké části automaticky. V případě, že se dva různí

uživatelé pokusí upravit ta samá data odlišným způsobem (např. přepsat stejnou větu v textovém souboru dvěma různými způsoby), může nastat **konflikt (kolize, *conflict*)**, který je nezbytné vyřešit – při **řešení konfliktu (*conflict resolution*)** je zpravidla nezbytné určit, která ze změn se má zachovat, a která naopak zahodit. Alternativně je možné zachovat obě varianty – v případě seznamu potravin na obr. 3 by takový postup vyústil ve výsledek tvaru: „Jablko, Jogurt, Vejce, Džus“.



Obr. 3: Schématické znázornění vzniku konfliktu při slučování do hlavní větve – seznam potravin byl upraven na stejném řádku dvěma odlišnými způsoby

3.1 Nejvýznamnější verzovací systémy, jejich funkce a vlastnosti

Systémy pro správu verzí lze všeobecně rozdělit podle různých parametrů (práce na úrovni souboru či adresáře, licenční politika, škálovatelnost, povaha slučování změn, podporované typy souborů). Systémy pracující s adresáři lze kategorizovat i podle uspořádání daného systému, a to na řešení centralizovaná a distribuovaná.

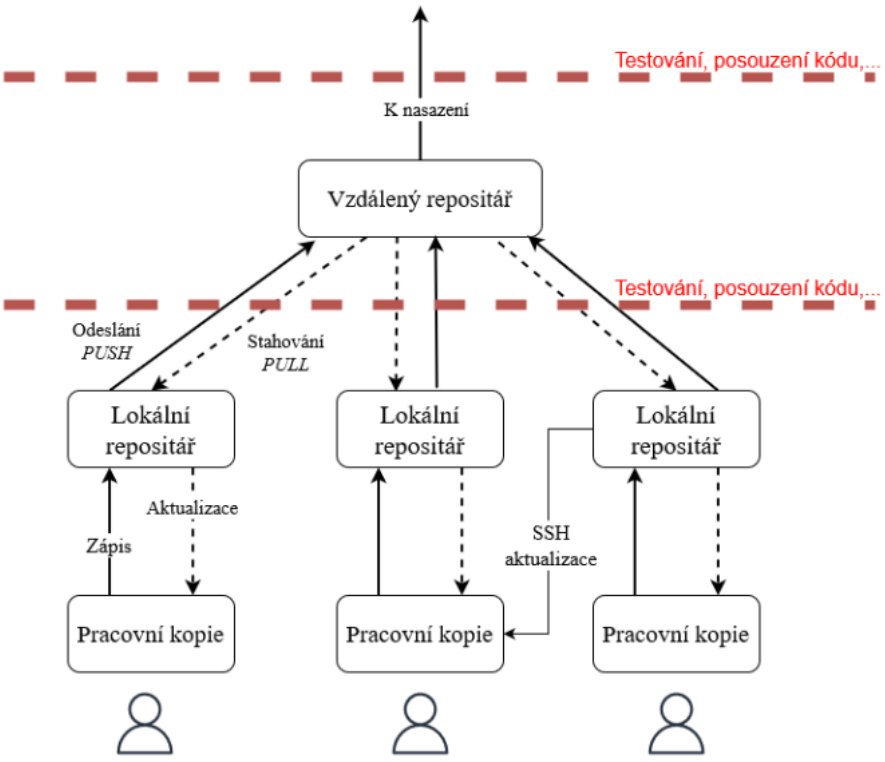
Centralizované systémy správy verzí (*Centralized version control systems*, CVCS) obsahují pouze jeden společný repozitář, který může být umístěn lokálně nebo vzdáleně na serveru. Uživatelé systému si mohou stahovat pracovní kopie, provést změny na své stanici a výsledek opět zapsat do centrálního uzlu.

Centralizované verzovací systémy mají některé nevýhody, které se týkají zejména sdílení práce s ostatními uživateli celého systému. Je složité sdílet vlastní pracovní kopii s někým jiným, případně nad ní spolupracovat s více jedinci zároveň. Centrální uzel je vždy dostupný

všem. Je nutná užší koordinace v rámci týmu, a proto jsou centralizované systémy vhodné spíše pro menší týmy (Rawat, 2022).

Distribuované systémy správy verzí (Distributed Version Control Systems, DVCS) fungují z velké části podobně jako systémy centralizované. Klíčovým rozdílem je to, že každý tvůrce má svou vlastní lokální kopii celého repozitáře (*local repository*) s kompletní historií změn. Pracovní kopii tvoří data větve, s kterou v dané chvíli tvůrce pracuje.

DVCS umožňují snáze sdílet změny mezi jednotlivými pracovníky – pokud to pravidla přístupu dovolí, lze do lokální kopie zanést změny z kopie kolegy (*pull*), nebo naopak obohatit cizí kopii o vlastní práci (*push*). Tyto operace mohou probíhat, aniž by byl dotčen vzdálený centrální repozitář, který je zpravidla přítomný i ve schématu DVCS – využívá se ke zpětnému sjednocení práce do společného celku (obr. 4). Ze vzdáleného repozitáře lze stahovat změny do repozitáře lokálního, nebo do něj naopak data nahrávat (Kučera, 2011).



Obr. 4: Schéma distribuovaného systému správy verzí s příkladem operace probíhající mimo centrální repozitář (s využitím SSH) a náznakem prvků CI/CD pipeline

V praxi může být přispívání do centrálního repozitáře v CVCS i DVCS omezeno různými pravidly. Před sloučením změn pocházejících z lokálního repozitáře do repozitáře

centrálního lze vyžadovat jejich schválení. Nastavení schvalovacího procesu je různorodé, může jít například o prosté ruční **posouzení kódu (*code review*)** třetí stranou (např. určenou osobou, vyučujícím). Pokročilejší schéma může provést i sadu automatizovaných kroků: posoudit, zda kód vyhovuje interním pravidlům formátování (*style guide*), vyhodnotit, zda příspěvek splňuje jednotkové či integrační testy (*unit* a *integration testing*) či ověřit, zda je výsledný kód se zanesením nových změn možné úspěšně sestavit (*build*).

Zároveň mohou být po nahrání dat do centrálního repozitáře spuštěny další úlohy – v případě vývoje webových aplikací např. zaslání výsledných dat na veřejný server tak, aby byla aktualizovaná verze aplikace přístupná pro návštěvníky v síti Internet. Tato automatizovaná schémata jsou zpravidla označována termínem *CI/CD Pipeline*, ve kterém zkratka *CI* znamená *continuous integration* (průběžná integrace) a zkratka *CD continuous deployment* (průběžné nasazování) (Simmons, 2022).

Při vývoji softwaru jsou dnes výrazně více používány distribuované systémy správy verzí na úkor systémů centralizovaných. Obecně panuje výrazná dominance systému Git, který dle studie serveru Stack Overflow (2022) užívalo přes 93 % vývojářů používajících jakýkoliv systém správy verzí. S výrazným odstupem následuje centralizovaný systém Subversion (5 %) a distribuovaný Mercurial (1 %). Následující text zkoumá tyto tři systémy podrobněji.

3.1.1 Apache Subversion (SVN)

Subversion je centralizovaný systém správy verzí, který je dostupný pod licencí otevřeného softwaru (*open source*). Na rozdíl od některých jiných verzovacích systémů neobsahuje integrované nástroje určené pro kontrolu verzí zdrojového kódu – Subversion je koncipován jako obecný verzovací systém, který je určen pro sledování změn souborů libovolného druhu (Pilato et al., 2008).

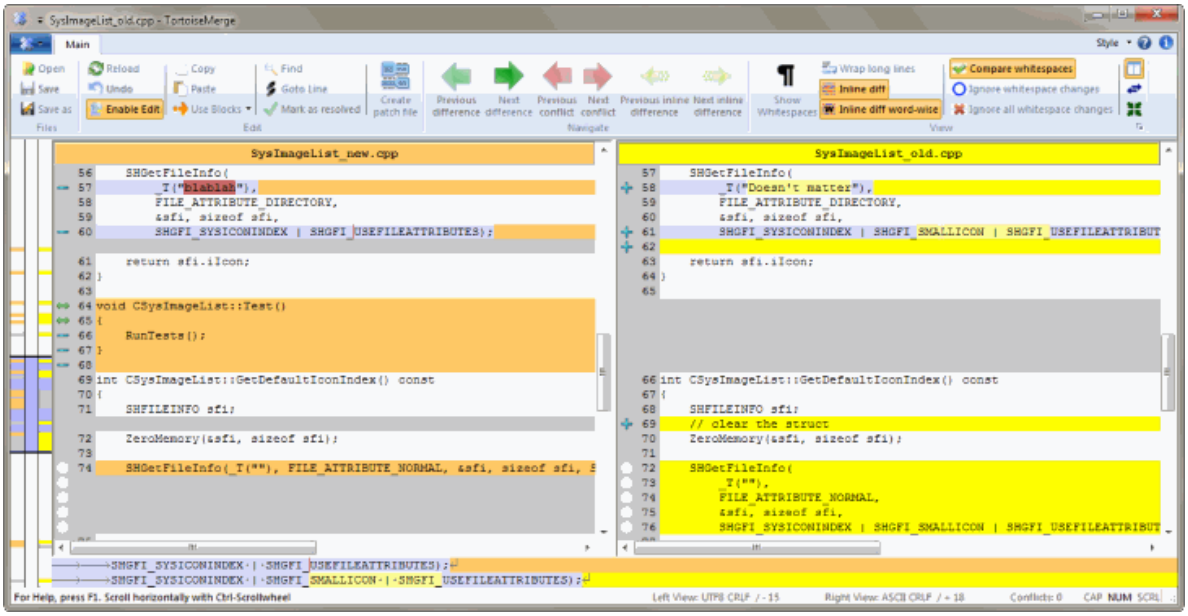
SVN je vyvíjen od roku 2000 jako náhrada staršího systému CVS (Concurrent Versions System). Subversion, stejně jako mnoho dalších verzovacích systémů, používá souběžnostní model „kopie-úprava-sloučení“ (*copy-modify-merge*), který je v obecné rovině popsán na počátku této kapitoly. V případech, že tento přístup nelze použít (např. u souborů, které neobsahují řádky textu), lze Subversion přepnout do režimu, který soubor vždy zamkne (*lock*) pro jednoho uživatele, a tím zamezí možnosti vzniku konfliktních změn.

Jednotliví uživatelé pracující nad společným řešením mohou provádět změny a následně je slučovat do centrálního repozitáře. Každý zápis vytvoří novou revizi (*revision*) celého souborového systému v repozitáři, přičemž revize je označena vždy přirozeným číslem, které je o jedničku větší, než číslo revize předchozí (např. revize 6 se po jednom zápisu změní na revizi 7). Nultá revize znamená, že je repozitář nově vytvořen a obsahuje pouze prázdný adresář.

K souborům či adresářům v repozitáři přistupují klientské aplikace pomocí URL (např. http://svn.priklad.com/projekt1/wwwroot). Pracovní kopie jsou v systému SVN striktně lokální a každý kořenový adresář pracovní kopie obsahuje pomocnou složku s názvem .svn (tzv. *administrative directory*).

Ačkoliv je možné se SVN pracovat čistě prostřednictvím příkazové řádky, existuje řada klientských aplikací, které poskytují grafické uživatelské rozhraní (GUI), jehož obsluha může být pro začátečníky jednodušší.

Příkladem je projekt TortoiseSVN, který je stejně jako celé Subversion poskytován pod svobodnou licencí. Jde o klientskou aplikaci pro systém Microsoft Windows. Nástroj je realizován jako rozšíření shellu – příkazy SVN jsou tak snadno dostupné z kontextového menu průzkumníka souborového systému operačního systému. Práce s TortoiseSVN je velice intuitivní, přestože klient obsahuje i pokročilejší funkce (obr. 5), jako je například grafický interaktivní nástroj pro řešení konfliktů (Küng, 2022).



Obr. 5: Nástroj pro porovnávání změn v souborech v rámci aplikace TortoiseSVN (zdroj: https://tortoisesvn.net)

3.1.2 Verzovací systém Git

Systém Git je v současné době nejoblíbenějším verzovacím nástrojem pro vývoj softwaru. Podobně jako systém Subversion (viz oddíl 3.1.1) je Git dostupný pod otevřenou licencí (GNU GPLv2). Jde o systém decentralizovaný, tj. každá pracovní kopie má k dispozici lokální plnohodnotný repozitář a nutně nemusí existovat žádný centrální bod (ačkoliv v praxi může být výhodné takový bod mít). Filozofií Git je časté využití větvení a opětovného slučování – hodí se i pro experimentální práci. Repozitář tvoří běžné statické soubory, systém lze provozovat i bez serveru a při samostatné práci i bez připojení k síti.

Git byl vyvinut v roce 2005 jako nástroj pro správu verzí v rámci vývoje jádra operačního systému Linux poté, co došlo k neshodě ohledně licenčních podmínek s tvůrci nástroje BitKeeper, který byl ke sledování změn v Linuxu používán dříve. Z tohoto rozkolu ve vývojářské komunitě vzešel kromě Gitu (jehož původním autorem je stejně jako u linuxového jádra programátor Linus Torvalds) i systém pro sledování změn Mercurial (viz oddíl 3.1.3). Není bez zajímavosti, že až do roku 2002 nebyl při vývoji linuxového jádra užíván žádný jednotný systém sledování změn.

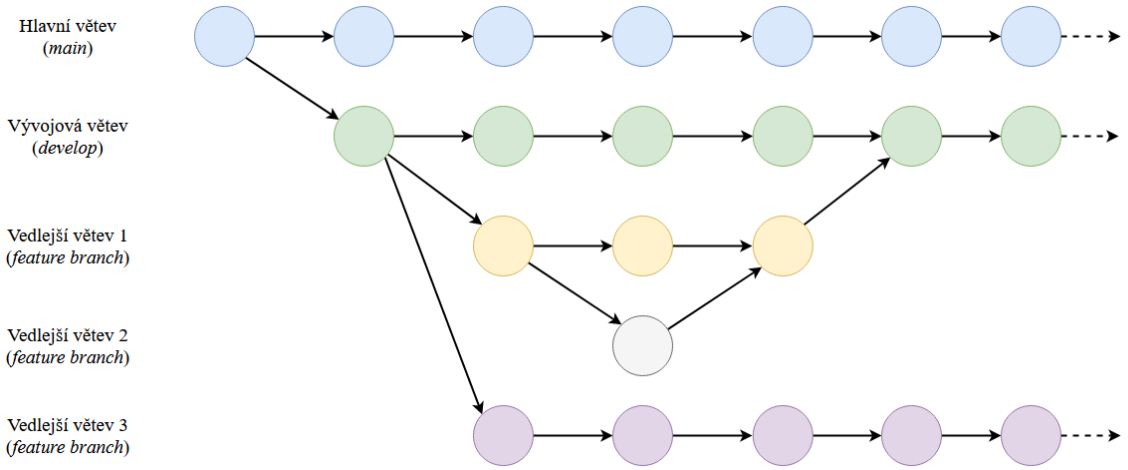
Práce s lokálním Git repozitářem (pracovní kopií) začne buď vytvořením zcela nového repozitáře nebo naklonováním již existujícího repozitáře (například ze serveru). V obou případech je výstupem kompletní repozitář, který bude v kořenovém adresáři obsahovat

pomocnou složku (nazvanou .git). V případě, že je repozitář nový, je možné jej následně nahrát na vzdálený server (tzv. *upstream*). Lze použít vlastní server nebo využít hostovacích služeb, jako jsou GitHub, GitLab či SourceForge.

Systém Git umožňuje použití různých **pracovních postupů (*workflows*)**. V nejjednodušším případě lze Git použít bez využití větvení, kdy všichni pracují nad jedinou společnou větví a do ní zapisují (*commit*) změny, které provedli. Nevýhoda tohoto pracovního postupu spočívá v nutnosti koordinovat zápisy změn. Pokud chce jeden pracovník odevzdat svoji práci (např. nahrát webovou stránku na server), ale druhý ještě s prací neskončil, je nutné, aby se oba vzájemně domluvili a jeden počkal na druhého (Antony, 2021).

Výhodnějším uspořádáním může být pracovní postup popsaný v úvodu této kapitoly. Tvůrce (či skupina tvůrců) si vytvoří vlastní větev určenou pro řešení jednotlivého úkolu (*feature branch*), přičemž tato větev se po dokončení práce sloučí zpět do větve hlavní.

Alternativou tohoto uspořádání je pracovní postup zvaný „Git Flow“, který kromě jedné společné větve přidává ještě mezistupně, z nichž nejvýznamnější má podobu **vývojové větve (*develop, development branch*)**. Vývojová větev shrnuje dokončené úkoly od různých pracovníků. Když se vývojová větev dostane do žádaného stavu, je možné ji sloučit do hlavní společné větve, která zpravidla obsahuje otestovaný kód připravený ke zveřejnění (obr. 6). Obdobných pracovních postupů je celá řada, rozšířením Git Flow je například postup GitLab Flow či GitHub Flow – oba mají pro specifická uplatnění několik dalších podvariant, např. pro postup GitLab Flow lze rozlišit postupy využívající *environment branches* a *release branches* (GitLab, 2023).



Obr. 6: Schématické znázornění větvení při použití pracovního postupu Git Flow. Většina vedlejších větví je při běžné práci vytvářena z vývojové větve, přičemž sloučení do hlavní větve zpravidla podléhá zvláštní kontrole

Při práci s existujícím vzdáleným repozitářem (obsahuje hlavní a vývojovou větev) s využitím prvků pracovního postupu Git Flow může postup jednotlivých úkonů při práci na přiděleném úkolu vypadat následovně:

1. Stažení aktuální verze repozitáře ze serveru (`git clone`, popř. `git pull`)
2. Vytvoření nové vedlejší větve pro řešení zadaného úkolu (`git branch`)
3. Přepnutí do nově vytvořené větve (`git checkout`)
4. Řešení zadaného úkolu (či jeho části)
5. Přidání změn do *staging area* (`git add`)
6. Zapsání změn ze *staging area* do lokálního repozitáře (`git commit`)
7. Sloučení změn do hlavní větve (`git merge`)
8. Odeslání změn do vzdáleného repozitáře (`git push`)

Staging area („oblast pro přípravu“) je místo, které lze v systému Git použít ke kontrole a k provedení úprav v seznamu souborů, které budeme (v rámci *commitu*) zapisovat. Pokročilejší funkce potom umožňují například rozdělovat změny zaznamenané ve staging area do několika různých zápisů.

Je-li potřeba některé soubory z našeho Git adresáře vynechat ze synchronizace se vzdáleným serverem, lze vytvořit soubor .gitignore. Jde o textový soubor, ve kterém je možné definovat, u kterých souborů v Git adresáři se nemají sledovat změny. V praxi může jít například

o pomocné soubory vývojových prostředí, které nemají být sdíleny s ostatními řešiteli projektu.

Data v systému Git jsou reprezentována pomocí objektů zvaných blob, commit, tree a tag. Tyto datové struktury drží informace o autorech zápisů, krátkých popisech provedených změn (*commit message*) a vnitřním uspořádání struktury projektu. Zároveň obsahují i seznam souborů v daném adresáři, přičemž každá položka tohoto seznamu odkazuje na další objekt, který již zaznamenává stav konkrétního souboru. Samotný obsah souborů je uchováván s využitím kompresního algoritmu z-lib.

Git navíc provádí za účelem zmenšení velikosti repozitářů automatickou správu paměti, k čemuž je využíván koncept „packfiles“. Při úkonu správy paměti je místo kopií celých souborů (každá provedená změna znamená novou kopii) uložena pouze posloupnost změn, ze které Git v případě potřeby umí zpětně vytvořit stav souboru k okamžiku zvoleného zápisu. Spuštění algoritmu pro správu paměti je automatické, nicméně lze jej iniciovat i manuálně (Scott a Straub, 2022).

V případě posloupnosti vícero commitů obsahuje každý z nich odkaz na commit předchozí. Odkaz je realizován pomocí otisku vytvořeného hashovací funkcí SHA-1. Otisk zároveň unikátně označuje každý commit. V tom spočívá jeden z rozdílů oproti systému SVN, který zápisy identifikuje prostými přirozenými čísly. Posloupnost zápisů systému Git lze interpretovat jako orientovaný acyklický graf (Virtanen, 2007).

V dokumentaci systému Git se s SHA-1 otisky lze setkat ještě v souvislosti s tzv. referenčními odkazy (*refs*) a značkami (*tags*). Referenční odkazy jsou z uživatelského hlediska aliasy pro SHA-1 otisky, které se sami o sobě obtížně pamatují i přepisují. Takovým referenčním odkazem je název větve (např. *master*) nebo ukazatel aktuální „polohy“ (zpravidla větve či commitu po operaci *checkout*) v rámci pracovní kopie, známý jako „HEAD“. Výhodou těchto odkazů je, že pokud je uvedeme v příkazové řádce jejich názvem, systém Git si je automaticky převede na kryptografický identifikátor, a tím vše spojí se správným cílovým objektem.

Značky fungují podobně, ovšem zatímco cíl referenčního odkazu se může v čase měnit (např. cíl odkazu HEAD se změní po přepnutí do jiné větve), značka je pevně svázána s určitým otiskem.

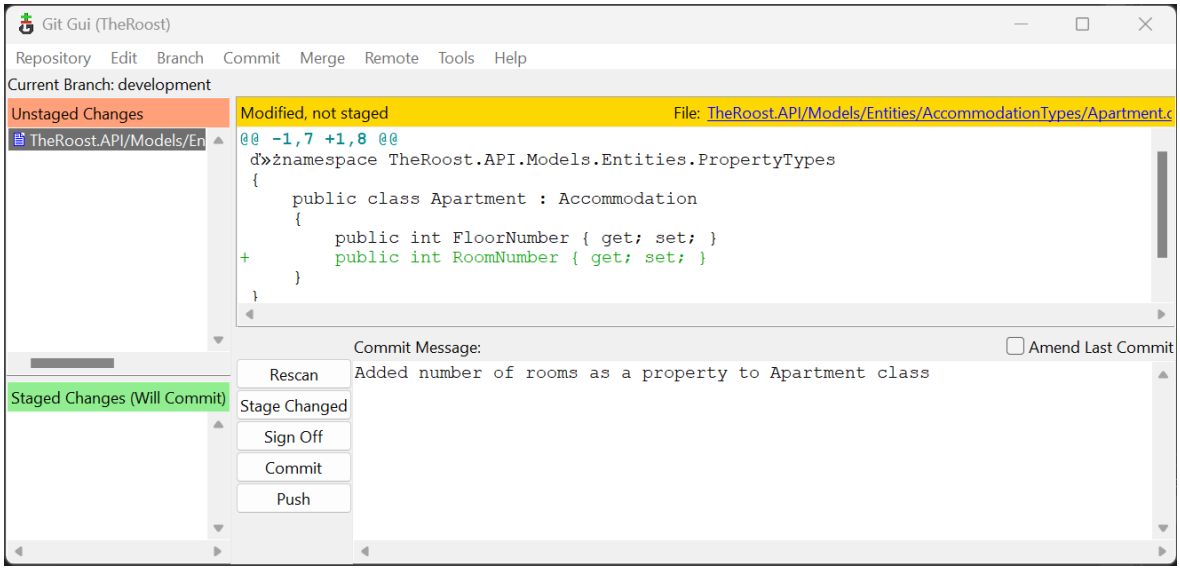
Značku tak lze použít k označení určitého pevného bodu při vývoji – například je možné určitému commitu přiřadit značku „v1.0.0“, která bude trvale označovat místo v historii úprav, ve kterém byla dokončena první verze projektu. Protože je při vývoji softwaru někdy nezbytné provádět zpětně úpravy (poskytovat podporu) i ve starších vydáních (*release*) aplikací, jsou značky v kombinaci s vhodně zvoleným pracovním postupem velice užitečným nástrojem.

Jak již bylo zmíněno, Git je systém decentralizovaný. Přehled základních operací/příkazů pro práci a obsluhu Git repozitářů v porovnáním s obsluhou repozitáře spravovaného centralizovaným systémem SVN je uveden v tab. 1.

Tabulka 1: Přehled základních operací/příkazů pro práci v systému Git a systému SVN

Základní operace systému Git a Apache Subversion (SVN)		
Příkaz Git	**Příkaz SVN**	**Popis**
checkout	clone	**SVN**: Vytvoří pracovní kopii z obsahu repozitáře či jeho části. **Git**: Vytvoří pracovní kopii, která je sama o sobě plnohodnotným Git repozitářem.
init	svnadmin create	Tvorba nového (prázdného) repozitáře.
add	add	**Git**: Přenese změny z pracovní kopie do *staging area* (více viz kap. 5). **SVN**: Zahájí sledování změn pro daný soubor.
commit	commit	Zapíše změny do repozitáře.
pull, fetch	update	Získá nové změny z repozitáře do lokálního prostředí.
log	log	Vypíše historii změn souboru či adresáře.
status, diff	status, diff	Slouží k nahlédnutí lokálně provedených změn.
revert	revert	**Git**: Inverzní zápis (*commit*). Zneguje změny konkrétního zápisu nově vytvořeným zápisem. **SVN**: Navrátí soubor či adresář do stavu před lokální změnou.
branch	copy	Vytvoří novou vedlejší větev.
merge, rebase	merge	Spustí postup pro spojování větví.

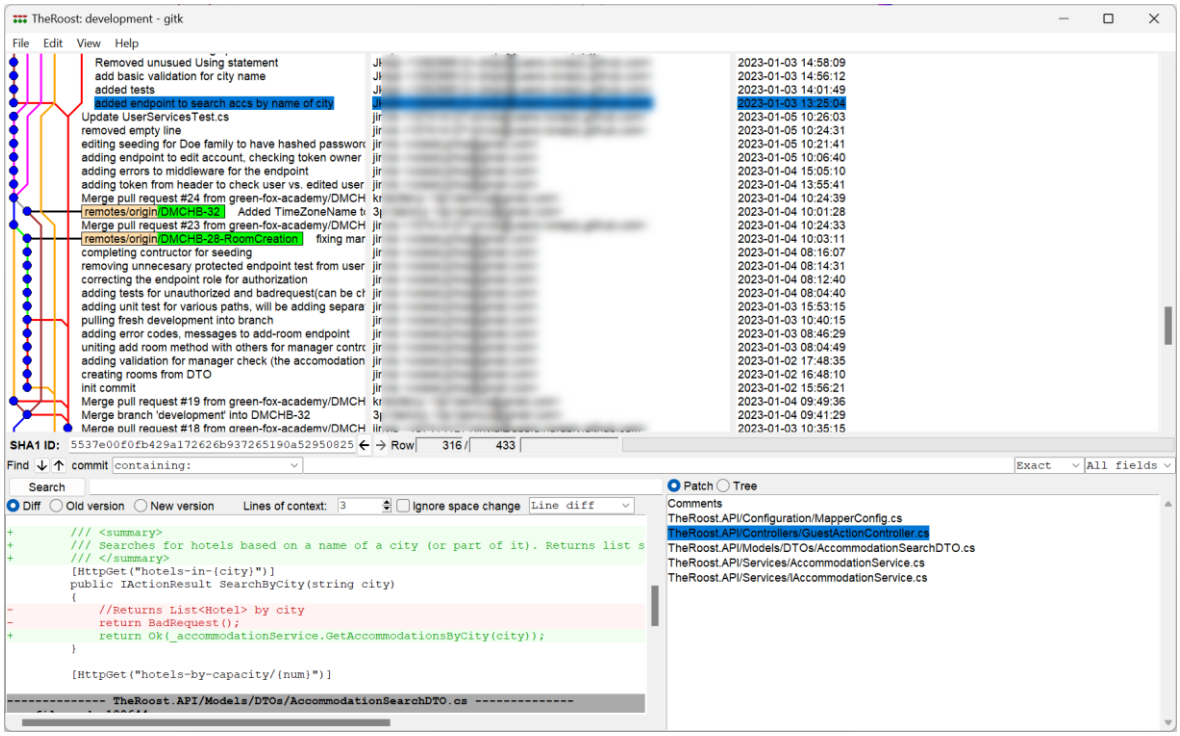
Systém Git lze ovládat přes příkazovou řádku (v prostředí Linux či macOS lze využít zabudovaný terminál, přičemž pro operační systém Windows je dostupný např. emulátor Git Bash). Do výchozí distribuce systému Git je však zabudován i multiplatformní modul **Git GUI**, který zpřístupňuje nejčastěji používané funkce v přívětivém grafickém uživatelském rozhraní (Git for Windows, 2023). Modul je stejně jako celý systém Git dostupný pod otevřenou licencí.



Obr. 7: Základní rozhraní modulu Git GUI

Uživatelské prostředí Git GUI (obr. 7) ve výchozím stavu zobrazuje v levém sloupci nahoře seznam souborů, které byly změněny, ale ještě nejsou přidány do *staging area*. Ve spodní části levého sloupce se nachází seznam souborů, které již v *staging area* jsou a při provedení příkazu *commit* budou jejich změny předány do repozitáře. V pravé části okna je obsah vybraného souboru s barevným vyznačením změn. Pod tímto přehledem je nabídnuta aktuálně dostupná akce – přidání *commitu*, který lze zároveň doplnit textovou zprávou (*commit message*).

Z horní lišty nástrojů lze spouštět další funkce, které umožní obsluhu lokálního i vzdáleného repozitáře, tvorbu nebo slučování větví či nahrávání změn. K dispozici jsou i pokročilejší funkce, jako je přidávání značek (*tags*) nebo řešení konfliktů. Tyto funkce jsou rovněž vyvedeny v interaktivním grafickém režimu, příkladem je nástroj umožňující grafické zobrazení obsahu celého repozitáře (gitk, obr. 8).



Obr. 8: Průzkumník obsahu repozitáře gitk dostupný z rozhraní Git GUI

Grafické rozhraní Git GUI poskytuje většinu funkcionalit dostupných přes příkazovou řádku. Přestože některé řidčeji používané funkce nejsou z tohoto GUI dostupné, z pohledu začátečníka jde o kompletní sadu nástrojů, kterou k běžné práci může potřebovat. Zároveň je nástroj dobře integrován s Git Bash a téměř každá interaktivní položka rozhraní odpovídá vykonání jednoho příkazu. Z tohoto důvodu je při znalosti obsluhy tohoto uživatelského rozhraní snadné následně přejít k užívání příkazové řádky.

Obecnou nevýhodou Git GUI, která je však společná i řadě dalších nástrojů, je nedostupnost českých překladů. Jazykem informačních technologií je angličtina, a to se nevyhýbá ani systémům pro správu verzí. Tento problém se projevil i při přípravě této práce – zaužívané české pojmy k některým anglickým výrazům (např. *staging area*) v podstatě neexistují a vymýšlet vlastní originální překlad nedává valný smysl, jelikož související literatura zůstane v anglickém jazyce.

3.1.3 Mercurial

Mercurial je dalším zástupcem decentralizovaných systémů pro správu verzí, který je dostupný pro UNIXové operační systémy i systém Microsoft Windows. Projekt je již od

svého založení v roce 2005 k dispozici pod otevřenou licencí. Vývoj systému byl od počátku poháněn snahou o maximální možnou úsporu systémových prostředků nutných k provozu distribuovaného verzovacího systému (Mackal, 2006).

Jelikož je Mercurial alternativou k systému Git, který je podrobně popsán v jiných částech této práce, soustředí se další text na popis odlišností tohoto projektu právě od systému Git. Systém Git je velmi komplexním nástrojem, což s sebou nese i jisté nevýhody, jako je například nižší přívětivost pro začátečníky. V Mercurialu je obecně méně příkazů, které mají zpravidla i jednodušší syntaxi a menší počet parametrů (SCM, 2023).

Další odlišností je přístup k historii změn. Zatímco v systému Git je možné se vracet zpět v historii jednotlivých zápisů a pozměňovat obsažená změnová data, Mercurial dovolí (při běžné práci) upravit pouze jeden poslední zápis. Manuální změny v historii zápisů mohou při lehkovážném použití způsobit komplikace při další práci, a proto může být v případě některých projektů vhodnější takovou možnost vůbec nemít k dispozici.

Podstatné odlišnosti existují i v přístupu k vytváření vedlejších větví. Systém Git je postaven na častém rozvětvení a zpětném sloučení dílčích příspěvků k jednomu vytvářenému celku. Mercurial je na tom jinak – větve jsou natrvalo zapsány do obsahu každého zápisu (*commitu*) a důraz na neměnnost historie znamená, že je nelze zpětně smazat. Systém rovněž neobsahuje žádný ekvivalent *staging area*, tedy oblasti, ve které by bylo možné postupně připravovat změny před jejich skutečným zápisem. Absence *staging area* znamená, že je celý systém o něco jednodušší, ale ve výsledku také méně flexibilní.

Jak Mercurial, tak i Git jsou distribuované systémy, které byly navíc vytvořeny velmi brzy po sobě. Přesto je nyní systém Git daleko populárnější. Hledat důvody tohoto stavu je spekulativní, nicméně lze uvažovat, že velký vliv na popularitu systému Git mělo spuštění platformy pro sdílení zdrojových kódů a hostingu repozitářů GitHub. A ačkoliv mohlo být použití systému Mercurial v příkazové řádce intuitivnější, tuto nevýhodu systému Git lze překonat použitím některého z řady grafických uživatelských rozhraní (některá z nich jsou představena v oddílu 0).

3.1.4 Další verzovací systémy

Dominance systému Git v prostředí softwarového vývoje neznamená, že jsou ostatní verzovací systémy zbytečné. Pro speciální účely se lépe hodí systémy centralizované, jako je SVN. Jedním z takových účelů může být grafická tvorba, 3D modelování, filmová tvorba; centralizované systémy totiž efektivněji pracují s velkými soubory.

Rovněž je nezbytné si uvědomit, že spravovat práci většího množství uživatelů nad jedním celkem je za přítomnosti netextových souborů obecně náročnější – při představě spolupráce na multimediálním projektu typu počítačové hry je ihned zřejmé, že tradiční systémy pro sledování změn v tomto případě nemusí pokrýt všechny projektové požadavky.

Pro designérskou práci často nelze použít slučování změn v podobě, která je známá z Git či SVN. Zároveň musí být použitý systém pro správu verzí dobře přístupný s ohledem na to, že jej neobsluhují přímo vývojáři. Dalším problémem je již zmíněná velikost souborů – u počítačových her se dostáváme do řádu desítek GB (Donovan, 2023).

Z těchto důvodů lze v praxi narazit i na jiná řešení, než je Git, ať už se jedná o zmíněné SVN, Mercurial či pokročilá komerční řešení, jako je například Perforce nebo Alienbrain.

3.2 Doplňky, služby a aplikace pro Git

Pro systém Git je dostupná celá řada doplňkového softwaru, služeb, rozšiřujících modulů a aplikací. V následujícím textu jsou blíže zkoumána některá z těchto řešení, která mohou práci s Gitem usnadnit a pomoci studentům s rychlejším pochopením Gitu při výuce. Jde o kombinaci praktických doplňků, vizualizačních pomůcek, výukového materiálu a hostingových služeb.

3.2.1 Hosting repozitářů

Existuje celá řada webových aplikací, které nabízejí hosting pro Git repozitáře v obchodním modelu „software jako služba“ (*Software-as-a-service*, SaaS). Největší výhodou vzdáleného repozitáře hostovaného na cloudové službě je jednoduchost instalace a konfigurace. Zatímco v případě hostování Git na vlastním serveru bez použití dalších nástrojů jsou nezbytné dovednosti týkající se správy serveru, cloudové služby se ovládají jednoduše pomocí webového uživatelského rozhraní.

Dvojicí nejznámějších cloudových hostingových služeb je GitHub a GitLab. Přestože se při zběžném pohledu jedná o obdobná řešení, existují mezi nimi i rozdíly. Společný je například platební model – v obou případech jsou základní funkcionality dostupné zdarma, ale aby bylo možno používat pokročilých funkcí, je nutné placené předplatné. Je však třeba dodat, že začínající vývojáři si zpravidla vystačí s funkcemi, které jsou dostupné bez poplatku. Obě služby nabízejí i nástroje pro správu a automatizaci vývojového procesu: management projektů, hlášení chyb, průběžnou integraci/nasazování nebo webové rozhraní pro hodnocení kvality kódu (*code review*).

Jedním z důležitých rozdílů mezi GitHub a GitLab je, že GitHub je čistě cloudová služba, zatímco GitLab nabízí i nástroje pro hosting na vlastních serverech (Gitlab Self-Managed). Pokud uživateli nevadí nutnost spravovat si vlastní server, je ovšem ke stejnému účelu možné využít i plně open source řešení, jako je například nástroj Gitea (dostupný pod licencí MIT).

Aplikace Gitea je multiplatformní a má nízké nároky na hardwarové vybavení použitého serveru (Gitea, 2023). Po instalaci je možné na serveru zprovoznit i uživatelské webové rozhraní, které z pohledu uživatele vypadá obdobně jako uživatelská rozhraní služeb GitHub či GitLab. Použití Gitea je dobrou alternativou, pokud je vhodné jít cestou hostování repozitářů na vlastním serveru a zároveň je nutné, aby uživatelé systému měli k dispozici intuitivní webové rozhraní.

Nevýhody Gitea vyplývají z povahy komunitního vývoje tohoto projektu, jde tedy například o absenci profesionální uživatelské podpory. Stejně tak pro Gitea není k dispozici tolik doplňkového softwaru, jako pro repozitáře hostované na komerčních cloudových hostingových službách.

Omezené jsou zatím i možnosti použití funkcí pro průběžnou integraci/nasazování. Z podstaty hostování na vlastním serveru plyne i absence nástrojů, jako je například GitHub Pages, který umožňuje hosting jednoduchých webových stránek přímo na doméně github.com. Hostování stránek přímo z repozitáře lze na platformě Gitea realizovat prostřednictvím vhodného doplňku (42wim, 2023).

V roce 2022 se projekt Gitea transformoval do obchodní firmy (Gitea Limited) a nabízí služby pro podnikové klienty. Samotná aplikace ovšem zůstává i nadále dostupná ve formě otevřeného kódu (Techknowlogick, 2022).

Gitea lze instalovat na operační systémy Linux, Windows, macOS i FreeBSD. Po instalaci z binárního souboru je nezbytné nakonfigurovat prostředí, přičemž program vyžaduje Git alespoň verze 2.0. Na serveru se nakonfiguruje uživatel, adresářová struktura a nastaví se pracovní adresář. Gitea se na serveru dle zvolené konfigurace spouští buď automaticky nebo manuálně s využitím příkazu zadaného do příkazové řádky/terminálu.

Program je dostupný i v kontejnerizované formě na uložišti Docker Hub. Jako alternativu k technologii Docker lze využít i systém pro orchestraci virtualizace Kubernetes. Existuje celá řada nástrojů podobných Gitea, které navenek vypadají podobně, a i funkčně se přibližují jeden druhému – jejich skutečné porovnání by však muselo být předmětem samostatné práce.

Z pohledu specifického použití ve výuce je použití Gitea vhodné i z důvodu lepší kontroly nad osobními daty studentů, jelikož informace zadané do systému nemusí opustit lokální prostředí vzdělávacího zařízení. Cloudové platformy typu GitHub nebo GitLab jsou sice v komerčním prostředí standardem, ovšem jejich skutečnou podstatou je práce se systémem Git, a tu si studenti osvojí i při práci s lokálně hostovaným Gitea prostředím.

Porovnání vlastností služeb Gitea, GitLab a GitHub je uvedeno v tab. 2.

Tabulka 2: Vlastnosti služeb GitHub, GitLab a Gitea

Porovnání služeb GitHub, GitLab a Gitea			
Vlastnost	**GitHub**	**GitLab**	**Gitea**
Repozitáře	veřejné neveřejné	veřejné neveřejné	veřejné neveřejné
Typ služby	vzdálené uložiště (cloud)	vzdálené uložiště (cloud) vlastní hosting	vlastní hosting
Spolupráce v týmu	sledování problémů v kódu a management projektu	sledování problémů v kódu a management projektu	základní sledování problémů v kódu a management projektu
Cenové náklady	základní funkce zdarma placené předplatné	základní funkce zdarma placené předplatné	zdarma
Licence	komerční	otevřený software (verze CE), komerční (verze EE)	otevřený software (MIT licence)
Průběžná integrace a nasazování	ano	ano	nativně ne
Integrace na SW třetích stran)	ano	ano	omezeně
Komunita vývojářů	velká	velká	menší

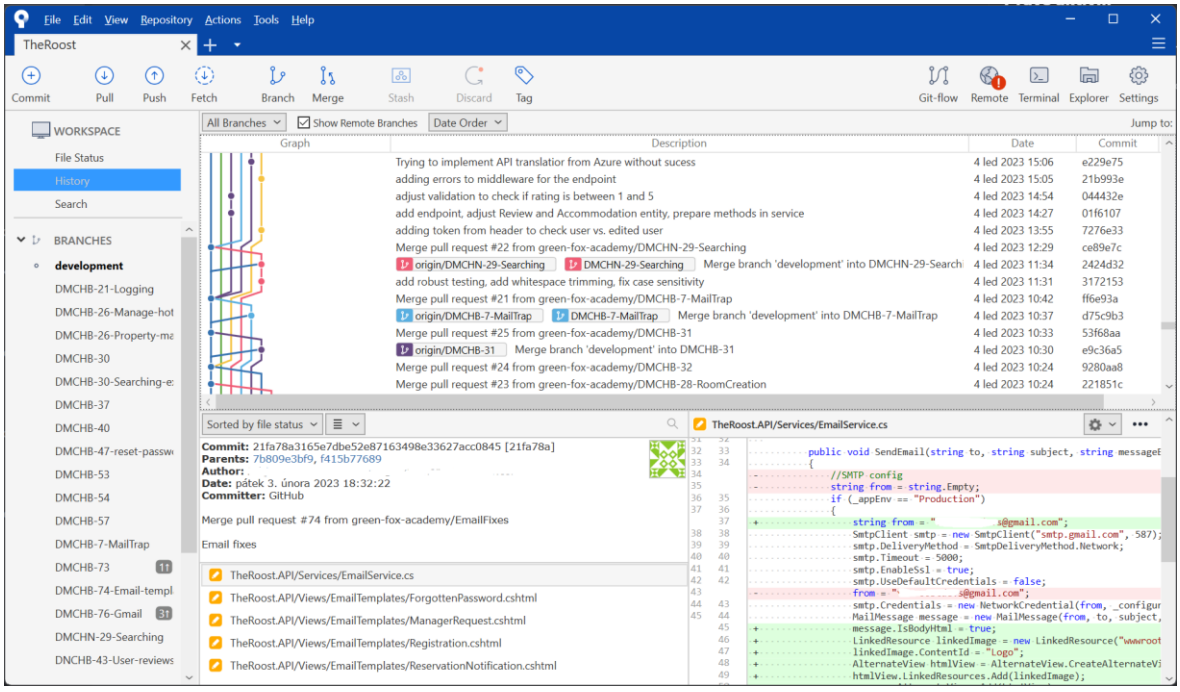
3.2.2 Doplňková grafická uživatelská prostředí

V předchozím textu byl představen zabudovaný modul Git GUI. Existuje však několik dalších grafických rozhraní.

SourceTree

Aplikace SourceTree je grafickým uživatelským rozhraním pro Git a Mercurial repozitáře. Podporuje operační systémy Windows a macOS a je dostupná zdarma pro nekomerční i komerční účely.

Nevýhodou SourceTree může být provázání s vlastníkem produktu, kterým je společnost Atlassian – instalace SourceTree vyžaduje registraci uživatelského účtu Atlassian, který je propojený s cloudovou službou BitBucket (analogie služby GitHub či GitLab). Je však možné připojit i jiné druhy repozitářů než ty spravované službou BitBucket. Samotné rozhraní se v hrubých rysech podobá modernější podobě zabudovaného nástroje Git GUI (obr. 9) (Atlassian, 2023).



Obr. 9: Základní uživatelské rozhraní aplikace SourceTree

GitHub Desktop

Cloudový poskytovatel služeb GitHub poskytuje i uživatelské rozhraní v podobě aplikace GitHub Desktop, které je dostupné pod otevřenou licencí (MIT). Ačkoliv použití tohoto

nástroje dává smysl zejména při souběžném použití služeb GitHub, lze napojit i jiné repozitáře, ať už lokální, nebo vzdálené (GitLab).

Rozhraní je jednoduššího a přehlednějšího rázu než u většiny ostatních GUI popsaných v tomto oddílu, ale pro uživatele začátečníka obsahuje všechny potřebné funkce. Při práci není patrná striktně úzká návaznost na příkazy používané při obsluze Git repozitáře z příkazové řádky. Velmi intuitivně je řešeno prohlížení historie jednotlivých commitů v repozitáři (GitHub, 2023).

GitKraken

Multiplatformní (Linux, macOS, Windows) uživatelské rozhraní, které je dostupné pod komerční licencí. K dispozici je ovšem možnost získat školní licence zdarma. Rozhraní je vizuálně přívětivé a moderní (podporuje například přesuny metodou „táhni-a-pusť“). GitKraken obsahuje i funkce pro integraci s nástroji projektového řízení, jako je JIRA nebo Trello (Axosoft, 2023).

TortoiseGit

Stejně jako nástroj TortoiseSVN, který již byl krátce popsán výše, funguje TortoiseGit jako rozšíření shellu v Microsoft Windows, základní funkce jsou tudíž dostupné přímo z kontextového menu. Rozhraní je rovněž velmi podobné, obsahuje všechny nezbytné funkce, ovšem působí subjektivně lehce zastaralým dojmem. TortoiseGit je dostupný pod otevřenou licencí (TortoiseGit, 2023).

Git Extensions

Git Extensions je GUI aplikace pro správu Git repozitářů v operačním systému Microsoft Windows. Aplikace je dostupná pod otevřenou licencí (GNU General Public License) a obsahuje integrace jak do shellu, tak do vývojových prostředí jako je Microsoft Visual Studio nebo Visual Studio Code. Git Extensions kombinuje výhody aplikace TortoiseGit s výhodami moderních samostatných grafických rozhraní typu GitHub Desktop (Git Extensions, 2023).

3.2.3 Využití FTP protokolu

V rámci systému Git je možné použít k přenosu na vzdálený server protokol FTP. Použití protokolu výrazně usnadňuje doplněk Git-ftp, který je po instalaci se systémem Git úzce

integrován. Doplněk je dostupný pod licencí GNU General Public a podporuje rovněž příbuzné protokoly SFTP, FTPS a FTPES (Git-ftp, 2022).

Git-ftp sleduje nahrané soubory prostřednictvím interního logovacího souboru (.git-ftp.log) umístěného na serveru. Při pokusu o přenos dat přenáší Git-ftp jen ty soubory, které se opravdu změnily – tato informace je získána ze systému Git. Tím tento doplněk umožňuje efektivní spolupráci serveru s lokálním repozitářem, jelikož nedochází k nadbytečnému přenosu dat a zároveň se minimalizuje prostor pro vznik nekonzistence dat (ke které by mohlo dojít, pokud by změnou dotčené soubory byly upraveny v průběhu FTP přenosu).

Výhodou použití tohoto doplňku je rovněž fakt, že Git stačí instalovat lokálně, instalace na použitý vzdálený server tedy není nutná.

Příklad práce s Git a nainstalovaným doplňkem Git-ftp může vypadat následovně:

1) Prvotní nahrání dat prostřednictvím protokolu ftp:
   `git ftp init -u "milan" -P ftp://198.133.219.25:21`
   Pokud již na serveru existuje kopie lokálních souborů, lze místo příkazu `init` použít příkaz `catchup`, který systém informuje o tom, že lokální data jsou plně synchronizována s daty na serveru.
2) Přístupová data a adresu serveru je možno pro další příkazy uložit do konfigurace:
   `git config git-ftp.<(url|user|password|syncroot|...)> <value>`
3) Pokud byla konfigurace uložena dle předchozího kroku, nahrání nových změn na server je možné provést jednoduchým příkazem:
   `git ftp push`
4) Změny dostupné na serveru lze do lokálního prostředí získat pomocí příkazu:
   `git ftp pull`
   Tento příkaz se automaticky pokusí provést i sloučení (*merge*) a následný zápis (*commit*). Pokud je nezbytné před zahájením slučování analyzovat nově ze serveru získaná data, je vhodnější využít příkazu:
   `git ftp pull --no-commit`

3.2.4 Výukové nástroje

Pro výuku Git je k dispozici několik interaktivních nástrojů, které lze využít buď přímo či alespoň jako inspiraci, jak prostředí Git žákům prezentovat. Všechny v tomto oddílu popsané nástroje jsou dostupné v anglickém jazyce.

Git-it

Git-it je multiplatformní desktopová aplikace, která představuje práci s prostředím Git s využitím reálného lokálního i vzdáleného repozitáře (umístěného na službě GitHub). Prostředí aplikace je jednoduché; obsahuje několik úkolů odstupňovaných podle obtížnosti, které lze postupně plnit. Prvním úkolem je zprovoznění systému Git a založení nového lokálního repozitáře.

Aplikace u každého úkolu obsahuje v první části textové a vizuální objasnění cíle. Následuje posloupnost kroků, které je nezbytné provést v příkazové řádce tak, aby bylo cíle dosaženo. Velkou výhodou aplikace je možnost ověřit si, zda je řešení správné – u většiny úkolů na to stačí vybrat složku s repozitářem a stisknout tlačítko „Verify“. V případě, že je řešení chybné, aplikace většinou sama poskytne zpětnou vazbu a napoví, co je třeba upravit.

Aplikace je dostupná pod otevřenou licencí (BSD-2-Clause). Při práci se vzdáleným GitHub repozitářem je nutné jej pro aplikaci nakonfigurovat a mít ve službě GitHub uživatelský účet (zdarma). Přínosem je, že uživatel vidí změny, které provedl, ve vzdáleném repozitáři (ve službě GitHub lze obsah repozitáře prohlédnout prostřednictvím webového prohlížeče) – aplikace tak provádí postupem, který je analogický skutečné práci s repozitáři (Git-it, 2023).

LearnGitBranching

Výuková webová aplikace v jazyce JavaScript, která prezentuje Git vizuálně prostřednictvím orientovaných grafů. Uživatel je navigován skrze sadu návodů a úloh, které jdou od naprostých základů až po představení středně pokročilých funkcí systému Git. První sada úloh se věnuje práci s lokálními repozitáři, druhá i práci s repozitáři vzdálenými. Obě části jsou však simulací – k vytváření reálných repozitářů, které by bylo možné prohlédnout v souborovém systému nebo na vzdáleném serveru, nedochází.

Uživatel zadává příkazy do simulované příkazové řádky a aplikace podle toho překresluje orientovaný graf, který reprezentuje obsah repozitáře. Výhodou aplikace je jednoduchost,

není nutné nic instalovat, stačí pouze zadat do prohlížeče internetovou adresu, pod kterou je projekt dostupný (https://learngitbranching.js.org/). Zdrojové kódy jsou přístupné pod licencí MIT (LearnGitBranching, 2023).

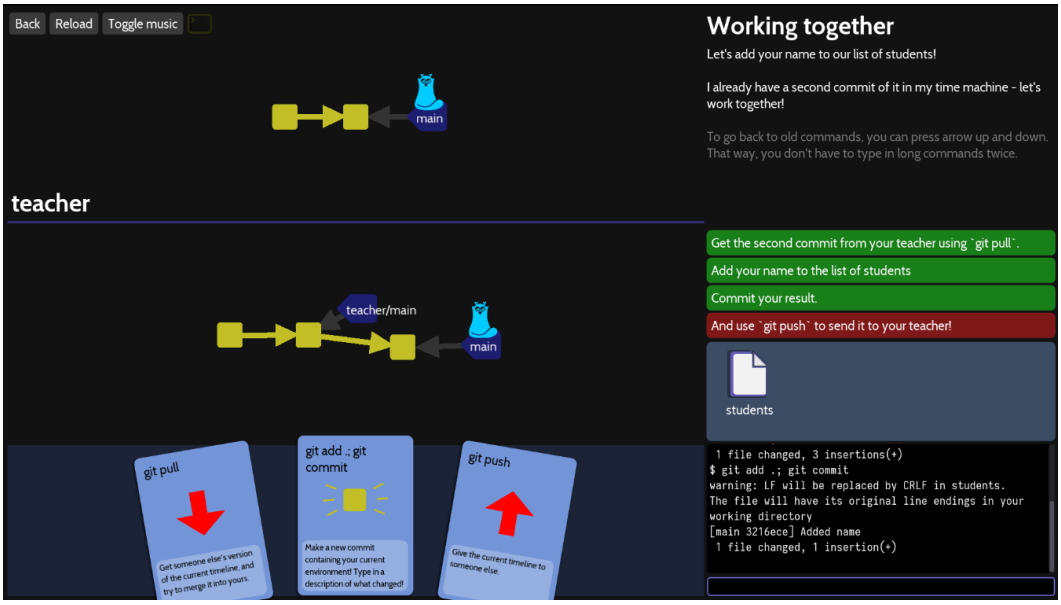
Oh My Git!

Na rozdíl od předchozích výukových nástrojů je „Oh My Git!“ spíše výukovou hrou, než jen souborem návodů a úloh. Jde o desktopovou aplikaci dostupnou pod otevřenou licencí (Blue Oak Model), která je spustitelná v operačním systému Windows, macOS i Linux (Oh My Git!, 2022).

Systém Git je zde prezentován prostřednictvím jednoduché karetní hry (obr. 10), kdy hráč plní jednotlivé úkoly vynášením karet na pomyslnou hrací desku, která je tvořena orientovaným grafem obsahu repozitáře. Vynesení každé karty znamená provedení jednoho či více příkazů s repozitářem (např. operací *add* a *commit*).

Hra má několik úrovní odstupňovaných podle obtížnosti. Kromě vynášení karet má uživatel možnost psát příkazy do simulované příkazové řádky. Použití karty hra odměňuje menším počtem bodů než přímé napsání příkazu. K již dokončeným úrovním se lze později vracet a zkoušet je vyřešit nejen prostřednictvím karet, ale právě i pomocí příkazů. Aplikace v každé úrovni obsahuje podrobný textový popis úlohy, který je stylem přizpůsoben prostředí počítačových her (Git je zde představen jako „stroj času“) a vyhýbá se popisu komplikovaných abstraktních konceptů.

V souhrnu je tato výuková hra inovativním řešením s velkým potenciálem nenásilného přiblížení mechanismů systému Git začátečníkům. „Herní“ mechanismus je plně funkční, ačkoliv prostředí samotnému by prospěl další vývoj (občas lze narazit na drobné chyby ve vykreslování).



Obr. 10: Snímek jedné z prvních úrovní ve výukovém programu „Oh My Git!“

GitHub Minesweeper

Webový kurz GitHub Minesweeper od německé společnosti Ooloo software UG je dostupný zdarma prostřednictvím webového prohlížeče (Profy, 2021). Je založen na zajímavém principu, kdy student prostřednictvím repozitáře na serveru GitHub zkoumá jednotlivá políčka na dvojrozměrném hracím poli v analogii známé hry „Hledání min“ (*Minesweeper*).

Kurz je rozdělený na několik kapitol. Každá kapitola obsahuje několik částí, ve kterých jsou vždy nejprve vysvětleny teoretické základy a následují instrukce k praktickému cvičení. V úvodní části se vytvoří repozitář na serveru GitHub, který bude aplikace i student používat. Prostřednictvím tohoto repozitáře jsou plněním úkolů v dalších kapitolách prozkoumávána jednotlivá políčka hracího pole. S „hráčem“ navíc v průběhu plnění jednotlivých úkolů kooperuje robotický mentor „Tara“, který schvaluje jednotlivé kroky a dává zpětnou vazbu. Reakce mentora jsou k dispozici na serveru GitHub, kdy tento bot věrně simuluje interakci s ostatními tvůrci při práci na reálném projektu.

Kurz postupně provede studenta jednotlivými aspekty práce s lokálním a vzdáleným repozitářem v pracovním schématu Git Flow. Uživatel v rámci hraní jedné partie hry „Hledání min“ prozkoumá i základy postupu průběžné integrace, vyzkouší si zadávání žádostí o sloučení do hlavní větve (*pull requests*), a naopak i schvalování cizích příspěvků (*code review*).

4 Role verzovacích systémů ve vzdělávání

Před začleněním verzovacích systémů do výuky je nutné se rozhodnout, zda je cílem naučit správu verzí jako takovou, nebo zda má být zapojení verzovacího systému do výuky spíše v podobě podpůrného nástroje, resp. didaktického prostředku při výuce jiných témat.

Verzovací systémy jako učivo

V rámci **výuky správy verzí** bude pravděpodobně cílem výuky to, aby žáci chápali základní koncepty, znali postupy a metody při správě verzí. Při výuce správy verzí je kladen důraz na vysvětlení důležitosti verzování a sledování změn, správu repozitářů, vytváření verzí, slučování změn a řešení konfliktů. Výuka může zahrnovat seznámení s různými nástroji správy verzí, jako je Git, Mercurial, Subversion (SVN) a další. Cílem je poskytnout studentům hlubší znalost konceptů a procesů správy verzí, tak, aby byli schopni účinně pracovat s verzovanými daty a sledovat jejich vývoj v čase.

Verzovací systémy jako didaktický prostředek

Výuka pomocí správy verzí může zahrnovat využití správy verzí jako nástroje v širším vzdělávacím kontextu. V rámci tohoto přístupu by správa verzí byla integrována jako součást výuky a byla využita při výuce jiných (se správou verzí přímo nesouvisejících) témat. Studenti by tak nejen používali konkrétní nástroje správy verzí, ale také aktivně využívali principy správy verzí k verzování svého kódu, řešení konfliktů, sledování historie změn a při společné práci celkově. Výhodou takového použití verzovacího systému by mohla být podpora týmové práce v kolektivu, snazší koordinace úkolů i efektivní sledování pokroku studentů vyučujícím.

Lze říci, že **výuka správy verzí** se zaměřuje na předávání znalostí o konceptech a postupech správy verzí v softwarovém vývoji obecně, zatímco **výuka pomocí správy verzí** zahrnuje začlenění správy verzí jako nástroje a metodiky do procesu výuky a učení s cílem podpořit zapojení studentů, spolupráci a správu změn. Tato práce se v dalších kapitolách zaměřuje na oba přístupy s vědomím toho, že před použitím správy verzí jako didaktické pomůcky musí studenti i vyučující chápat základní principy správy verzí a mít k dispozici adekvátní prostředí pro svoji práci.

Vhodným užitím systému pro správu verzí ve výuce by mohlo být začlenění takového systému do výuky při práci na dlouhodobějším projektu v rámci třídy. Výhodou by bylo, že takový projekt by studentům nepřinesl pouze úzce profilované znalosti, ale rozvíjel by i další související dovednosti. Jelikož se verzovací systémy hodí zejména pro práci v týmech, přirozeně se hodí i k tréninku komunikačních dovedností. Nejde pouze o trénink spolupráce, ale zároveň i odpovědnosti, jelikož u každé změny lze při použití systému pro kontrolu verzí dohledat, kdo ji provedl a za jakým účelem.

Komplexnější představení vybraného verzovacího systému, které by bylo nutné k jeho samostatnému použití studenty v rámci dlouhodobějšího projektu, by si patrně vyžádalo velkou časovou investici v rámci výuky (k tématu podrobněji viz oddíl 7.1). Domnívám se však, že i v případě, že by studenti společně nepracovali na velkém sdíleném projektu, by i skromnější představení verzovacích systémů mohlo přispět k rozvoji jejich znalostí a kompetencí. Důvodem je fakt, že s takovými systémy se mohou setkat ve svém dalším životě v celém spektru oboru informačních technologií, který jen v rámci ČR v současné době zaměstnává přibližně 220 tisíc osob (ČSÚ, 2022).

I v případě, že studenti budou ve své budoucí profesní dráze směřovat mimo obor IT, existuje šance, že se systémy, které jsou předmětem této práce, přijdou do styku. Příkladem může být jejich použití v rámci kreativních designérských či audiovizuálních prací (viz pododdíl 3.1.4), při vědeckém bádání či ve strojírenství. Stejně tak lze při výuce verzovacích systémů trénovat dříve nabyté dovednosti, ať už jde o obsluhu uživatelských rozhraní, programování, seznamování se s novými aplikacemi, samostatné řešení problémů, nebo práci s příkazovou řádkou.

Užitečnou dovedností, kterou verzovací systémy rovněž upevňují, je zálohování. Pokud je použit vzdálený repozitář k ukládání změn z lokálního prostředí, vytváří se tím pojistka proti ztrátě dat v případě nechtěného smazání nebo poruchy zařízení. Vzdálený repozitář zajišťuje i přenositelnost. Jsou-li k dispozici přístupové údaje nebo je repozitář veřejný, je možné pracovat z různých zařízení a bez ohledu na místo, odkud uživatel pracuje.

S přenositelností souvisí i fakt, že systém Git je dostupný pod otevřenou licencí a používá se v celé řadě open-source projektů, na kterých často spolupracují tvůrci z celého světa, aniž by se kdy fyzicky setkali. Toho je možné využít při výuce a zasvětit studenty do široké

problematiky licencí a otevřeného softwaru – základní porozumění tomu, kdy a proč můžeme software použít bezplatně a za jakých podmínek, je rovněž jednou z důležitých znalostí pro orientaci ve světě informačních technologií.

Verzovací systémy by mohly být vhodné i k osvojení návyků týkajících se dokumentování své vlastní práce. Při zapisování (*commit*) změn do repozitáře je nutné uvést krátkou textovou zprávu, která tyto změny osvětluje. Vyučující může (podobně jako projektový manažer) klást důraz na to, aby tyto zprávy byly smysluplné. Je samozřejmě možné vynutit vybraná pravidla i systémově (např. povolit studentům zápis, jen pokud připojený popisek vysvětlující změny dosáhne stanovené minimální délky).

Porozumění softwarovému nástroji, jako je verzovací systém, přináší i potřebnou sebedůvěru při práci s informačními technologiemi obecně. Pokud student porozumí Gitu, znamená to, že s dalšími systémy obdobné komplexnosti se již seznámí snáze.

4.1 Vzdělávací platformy cloudových poskytovatelů

Někteří poskytovatelé hostingu a cloudových služeb určených pro použití se systémem Git nabízí speciální licence pro vzdělávací účely.

GitLab for Education (a obdobný produkt GitHub Campus Program) poskytuje vyučujícím nástroje pro jednoduché vytváření a správu úkolů a projektů pro studenty, včetně možnosti zadávání a sledování pokroku studentů při řešení úkolů. Studenti mohou platformu GitLab používat k odevzdávání svých řešení a vyučující mohou tato řešení hodnotit a poskytovat zpětnou vazbu (GitLab, 2023).

Projekty v tomto prostředí mohou být individuální i skupinové. Studenti mohou k projektům přistupovat, vytvářet a odevzdávat svoji práci. Uživatelé v roli vyučujícího mohou projekty spravovat a zároveň sledovat výsledky studentů. Platforma obsahuje i možnost vytváření diskusních fór, kde mohou studenti probírat a sdílet své nápady nebo řešit problémy v rámci projektů a úkolů. Vyučující také mohou komunikovat s jednotlivými studenty i celou třídou.

4.1.1 GitHub Classroom

GitHub Classroom je nástroj, který pomáhá pedagogům a mentorům při výuce Gitu a GitHubu. Jedná se o bezplatný projekt, který poskytuje široké spektrum funkcí pro organizaci výuky pomocí Gitu a GitHubu (GitHub, 2023). GitHub Classroom umožňuje

snadno spravovat úlohy, vytvářet a přidělovat úkoly studentům, sledovat postup studentů při řešení úkolů, poskytovat zpětnou vazbu a hodnotit práci studentů.

V rámci prvotního nastavení GitHub Classroom je nutné vytvořit novou organizaci a korektně jí přidělit přístup. Organizace funguje jako hlavní sdružující účet pro veškeré repozitáře vytvořené v GitHub Classroom. Při zakládání organizace je vhodné zvolit možnost „Organization belongs to my personal account" – druhá možnost je dostupná pouze v případě, že je organizace partnerem programu GitHub Campus nebo využívá placených služeb společnosti GitHub. V České republice je v době psaní práce (2023) partnerem v programu GitHub Campus celkem 10 škol a jedna další organizace.

Po založení organizace lze vytvořit první třídu. Třídě je možné přidělit název a pomocí vygenerovaného odkazu zaslat pozvánku členům, kteří budou mít roli administrátora (zpravidla vyučující). Aby se mohl uživatel stát ve třídě administrátorem, musí mít k dispozici nejen pozvánku, ale zároveň musí být jeho jméno ve službě GitHub nastaveno v patřičné roli v rámci organizace.

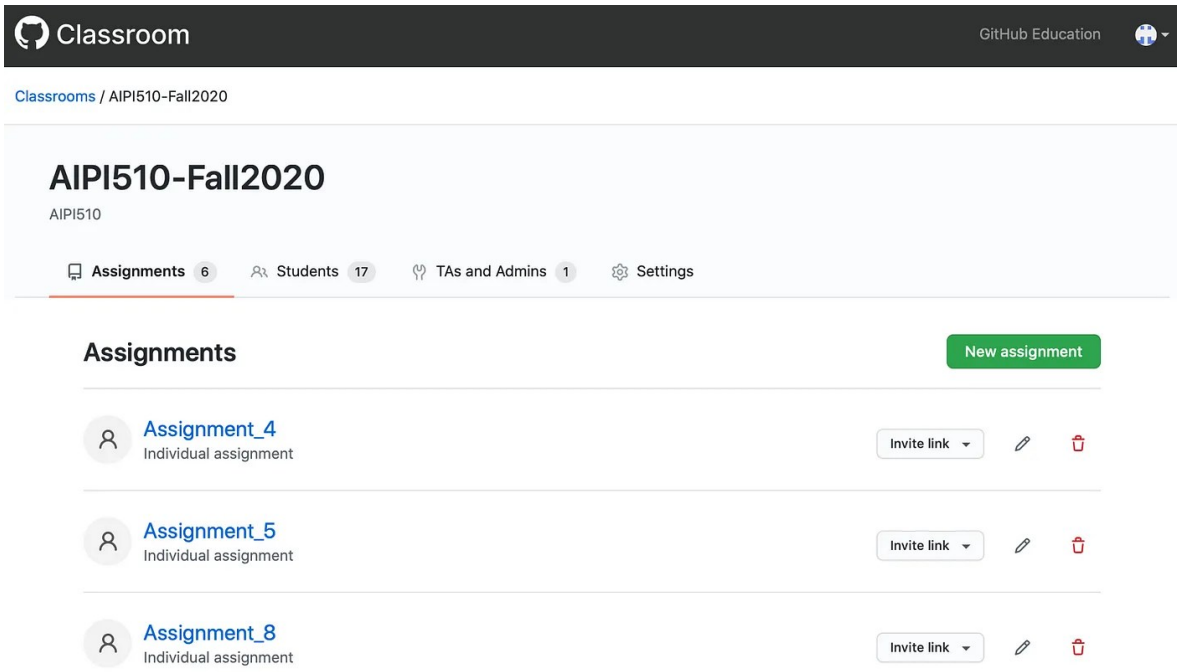
V dalším kroku je již možné do třídy přidat studenty. Seznam studentů lze buď nahrát z existujících služeb, jako je Google Classroom nebo Moodle či jej naimportovat ze souboru CSV. Alternativně lze do textového pole vypsat seznam studentů ručně. Studenti mohou být označeni libovolně – jménem, identifikátorem atp. V záložce „Students" poté lze jednotlivé studenty napárovat na jejich účty ve službě GitHub. Tento krok je možné nechat i na studentech samotných – při převzetí prvního zadání je k napojení účtu služba sama vybídne.

GitHub Classroom je po přidání studentů nastaven a již je možné přidat první zadání úloh (obr. 11). Pedagog snadno vytvoří repozitář pro každou úlohu a přidělí ji studentům. Studenty lze k zadání pozvat přes unikátní odkaz, který služba vygeneruje. Každý student má svoji vlastní kopii repozitáře, kde může řešit úlohu a ukládat svůj postup, který může vyučující sledovat.

Vyučující mohou také vytvářet šablony pro další projekty (*template repositories*) a zadávat úkoly, které jsou založeny na těchto šablonách. To umožňuje studentům i vyučujícím rychle se zapojit do projektů a snadno porozumět povaze projektů i zásadám verzovacích systémů.

Webové prostředí obsahuje i pokročilejší funkce: je možné nastavit automatické vyhodnocování studentských prací a projekty mohou být nejen individuální, ale i skupinové. Službu lze i propojit s některými výukovými aplikacemi, jako je např. Microsoft MakeCode Arcade (využívá vizuální programovací jazyk).

Při použití služby GitHub Classroom je třeba myslet na to, že bezplatný plán je v některých směrech omezen. Veškeré automatizované úkony, které využívají GitHub Actions, se započítávají do celkového limitu dostupného strojového času.



Obr. 11: Rozhraní webové aplikace GitHub Classroom (zdroj: https://towardsdatascience.com/)

Celkově poskytují vzdělávací platformy navázané na Git ucelené prostředí pro správu projektů a úkolů. Navíc usnadňují komunikaci a spolupráci mezi studenty a vyučujícím. Nevýhodou je nutnost prvotního nastavení a omezené možnosti v bezplatných verzích těchto platforem.

5 Návrh zapojení Gitu do výuky

V tomto textu budou představeny možné pracovní postupy pro práci se systémem Git ve výuce. Návrh počítá primárně s využitím hostingu repozitářů na vlastním serveru. Alternativou by bylo využití služeb hostingu repozitářů jako je GitHub či GitLab. Postup pro obě varianty je uveden v příloze 1 této práce. V jednotlivých oddílech této kapitoly je vysvětlen postup získání zadání, provedení změn (řešení zadané úlohy) a konečného odevzdání práce. Vždy je uveden postup při použití Gitu bez doplňků. Ke každému postupu je v příloze 2 uveden postup alternativní – použitelný pro GUI SourceTree, aplikaci PhpStorm a vývojářské prostředí Visual Studio Code. Jednotlivé kroky jsou řešeny z pohledu studenta (uživatele přispívajícího do existujícího vzdáleného repozitáře). V navazující kapitole bude představeno ukázkové řešení, které využívá obecných postupů prezentovaných v této kapitole k realizaci konkrétních výukových plánů.

5.1 Postup žáků při řešení zadané úlohy

5.1.1 Stažení zadání ze serveru

Při použití systému Git bez rozšíření, musí student pro stažení zadání naklonovat repozitář pomocí příkazové řádky:

1. Nejprve je nezbytné přejít do adresáře, do kterého chce student repozitář naklonovat.
2. Zkopíruje se cesta k repozitáři, který chceme naklonovat. Pokud používáme služeb pro hostování repozitáře (např. GitHub nebo GitLab), najdeme odkaz na webových stránkách příslušné služby.
3. Naklonujeme repozitář do aktuálního adresáře příkazem „`git clone`“, kterému jako argument dodáme cestu (odkaz) získanou v předchozím kroku.

Tím je repozitář zkopírovaný a student může pokračovat v práci vytvořením nové větve pro své řešení pomocí příkazu „`git branch`“ a „`git checkout`“.

5.1.2 Řešení zadané úlohy

Studenti v průběhu řešení pracují se soubory v repozitáři, vytvářejí a zapisují změny (*commit*). Dle postupu z předchozí části si studenti naklonovali repozitář na své počítače.

Poté si vytvořili novou větev, ve které pracují. V průběhu řešení následně řešitelé úlohy uplatní tento postup:

1. Provedou požadované změny tak, aby bylo vyřešeno zadání úlohy.
2. Pro přidání všech změn do staging area zadají do terminálu příkaz „`git add .`“
3. Za účelem samotného zápisu využijí příkaz „`git commit -m "popis změn"`“.

5.1.3 Odevzdávání řešení na server

Když studenti dokončí řešení zadané úlohy, mohou vyučujícímu výsledek předat tak, že jej pomocí Git nahrají na vzdálený repozitář. Z hlediska systému Git tímto dojde k nahrání vedlejší větve do hlavního repozitáře.

Vyučující poté zkontroluje změny v repozitáři na vzdáleném serveru a poskytne zpětnou vazbu studentům. V případě potřeby lze při kontrole využít příkaz v Gitu pro stažení nejnovějších změn ze vzdáleného serveru na svůj počítač („`git pull`“).

K nahrání bude opět využita příkazová řádka:

1. Otevře se příkazový řádek ve složce s naklonovaným repozitářem.
2. Poté je nutné se přepnout do vlastní větve, ve které byly provedeny změny, a to pomocí příkazu „`git checkout "nazev_vetve"`“.
3. Příkazem „`git status`“ se zkontroluje, zda jsou změny správně zapsány do commitů.
4. Pokud jsou změny správně zapsány, lze je předat na server: „`git push origin "vlastní větev"`“.

5.2 Postup vyučujícího při přípravě úloh pro studenty

Modelové řešení přípravy jednotlivých úloh pro studenty si ukážeme pouze s použitím systému Git v příkazové řádce. Předpokládejme, že každý student má vlastní repozitář, který je nastaven tak, aby bylo možné vytvářet vlastní větve. Vyučující rozdá studentům připravená zadání. Studenti poté pracují na svých vlastních řešeních pomocí postupu, které již byli popsány v první části této kapitoly – nejprve si vytvoří své vlastní větve příkazem:

```
git branch nazev_vetve
```

Poté studenti pracují na svých řešeních a průběžně zapisují dílčí výsledky:

```
git add .
git commit -m "zprava_commitu"
```

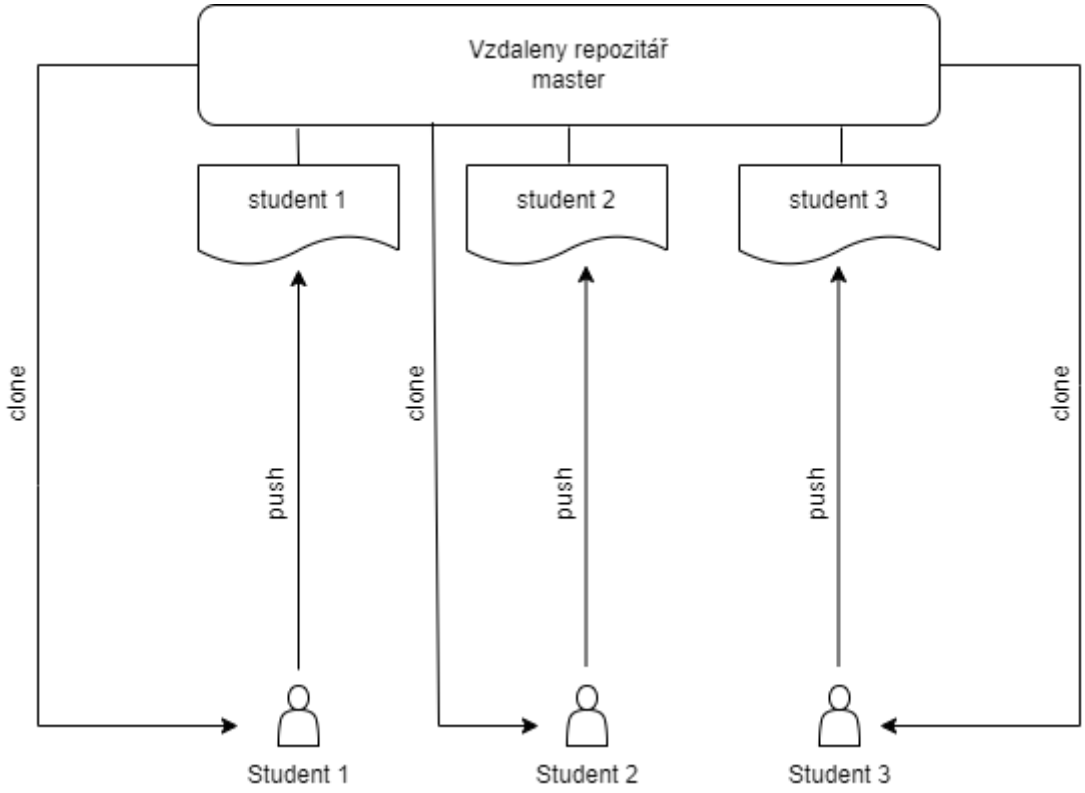
Když dokončí svou práci na větvi příslušné k zadanému úkolu, odešlou výsledek na server:

```
git push origin nazev_vetve
```

Pokud by student odesílal práci přímo na hlavní větev, musel by nejprve stáhnout aktuální verzi hlavní větve ze vzdáleného repozitáře a vyřešit případné konflikty. Odesílá-li řešení obsažené v rámci vlastní vedlejší větve, ke konfliktům by dojít nemělo.

Pokud učitel nalezne nějaké chyby nebo problémy, může předat jejich seznam studentovi. Ten je v ideálním případě následně opraví ve svém lokálním repozitáři, a poté znovu provede zápis a nahrání na vzdálený server pomocí příkazů „`add`“, „`commit`“ a „`push`“.

Tento cyklus zpětné vazby, opravy a opětovného nahrání řešení do vzdáleného repozitáře se může opakovat, dokud není kód v dané větvi dokončený a řešení neodpovídá zadání (schematické znázornění viz obr. 12). Zbude-li čas, může vyučující představit i koncept slučování větví do větve hlavní (*master*), a to například tak, že vybere jedno nejlepší odevzdané řešení a demonstruje sloučení s hlavní větví s využitím příkazu „`git merge`“.



Obr. 12: Práce studentů s repozitářem při řešení zadané úlohy

5.3 Domácí úkoly

Domácí úkoly se připravují obdobným způsobem, jako úlohy pro práci v hodinách. V tomto oddílu je popsán postup tvorby jednotlivých repozitářů pro studenty z pohledu vyučujícího a následně i postup řešení z pohledu studenta. Nejprve se vyučující přihlásí jako správce na linuxový server. Poté je vhodné vytvořit zvláštní složku určenou pro repozitáře domácích úkolů. To lze provést příkazem „`mkdir`“. Správná oprávnění (čtení, zápis, spouštění) pro tuto složku se nastavují pomocí příkazu „`chmod`“:

```
mkdir homework_repos
chmod 775 homework_repos
```

Pro studenty poté vytvoříme nové prázdné repozitáře pomocí příkazu „`git init`“. Například:

```
cd homework_repos
git init student1_repo.git
```

A opět nastavíme správná oprávnění pro každý repozitář:

```
chmod 770 student1_repo.git
```

Dále změníme vlastníka pomocí příkazu „chown“ tak, aby každý repozitář patřil danému studentovi:

```
chown -R student1:student1 student1_repo.git
```

Tyto kroky opakujeme pro každého studenta a jeho repozitář. Tím jsou všechny repozitáře připraveny a správně nastaveny tak, aby každý student měl přístup pouze ke svému vlastnímu repozitáři.

Vyučující může po této přípravě poskytnout studentům odkazy na jejich repozitáře, aby si mohli stáhnout (naklonovat) jejich obsah a začít pracovat na svých domácích úkolech.

Studenti si stáhnou zadání domácího úkolu a naklonují si prázdný repozitář, který mají připraven na serveru. Použijí k tomu příkaz „`git clone`“ a odkaz na repozitář, který jim učitel poskytne.

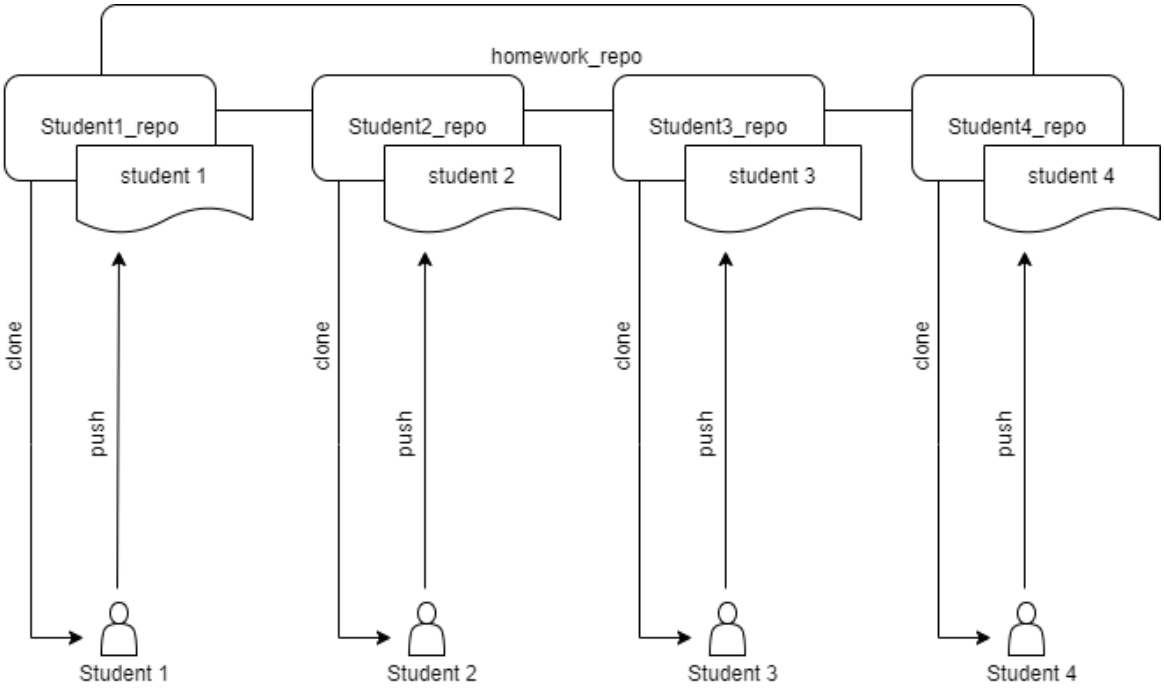
```
git clone student1@server:/path/to/student1_repo.git
```

Poté studenti vytvoří novou větev pro řešení svého úkolu. Větev pojmenují podle zadání a na této větvi budou pracovat. K vytvoření větve použijí například příkaz:

```
git checkout -b novy_ukol.
```

Jakmile student dokončí svůj úkol, odešle svou větev na server. Pokud je řešení naprosto nevyhovující, může vyučující požádat studenta o zcela nové vypracování zadané úlohy. V takovém případě musí student vrátit svou větev do výchozího stavu. K tomu lze využít příkaz „`git reset –hard`“.

Postup práce studentů při řešení domácího úkolu je schematicky znázorněn na obr. 12.



Obr. 13: Práce studentů při řešení domácího úkolu

5.4 Společná práce studentů

Společná práce studentů je simulací vývoje v pracovních týmech. Několik studentů bude pracovat na jednom projektu, a to buď současně na stejných souborech nebo každý v jiné části projektu rozděleného na jednotlivé části. Realizace je možná na serveru nebo lze využít služeb GitHub či GitLab. Následující text se bude zabývat řešením na vlastním serveru, rozdělením projektu, kontrolou zpracování a následným sloučením celého projektu.

5.4.1 Rozdělení projektů do více sekcí

Učitel nejprve vytvoří prázdný repozitář na serveru a naklonuje si ho k sobě na počítač pomocí příkazu „git clone“.

```
git clone ssh://user@server:/cesta/k/repozitari
```

V tomto případě je „user“ uživatelské jméno učitele, „server“ je název serveru, na kterém je repozitář umístěn a „/cesta/k/repozitari“ je cesta ke složce, kde se repozitář nachází. Poté je nutné inicializovat větve pro každou sekci projektu. Uvažujme, že projekt má tři části (sekce). Vytvoří se tedy tři větve:

```
git branch sekce1
```

```
git branch sekce2

git branch sekce3
```

Následně se lze přepnout do jedné z větví pomocí příkazu:

```
git checkout nazev_vetve .
```

Poté je možné vytvořit v rámci této větve soubory, které budou součástí zadání. Příklad pro vytvoření souboru pro první sekci je následující:

```
git checkout sekce1

touch soubor_pro_sekci1.php

git add soubor_pro_sekci1.php

git commit -m "Přidán soubor pro sekci1"
```

Tyto kroky se opakují pro každou větev, tedy pro každou sekci projektu.

5.4.2 Odevzdávání a kontrola jednotlivých sekcí projektu

Každý řešitel si naklonuje celý repozitář na svůj počítač pomocí příkazu „`git clone`“. Poté se přepne do větve odpovídající sekci projektu pomocí příkazu „`git checkout`“. Například pro práci na „sekci1“:

```
git checkout sekce1
```

Student poté opět vytvoří vlastní větev, ve které bude dále pracovat. Například pro vytvoření vlastní větve s názvem „moje_vetev“ pro práci v „sekci1“:

```
git branch moje_vetev
```

Student následně provede úpravy v souborech patřících do vybrané sekce a výsledek odevzdá příkazem „`git push`“. Například přidá soubor „novy_soubor.html“ do „sekce1“ a změny odevzdá ve vlastní větvi:

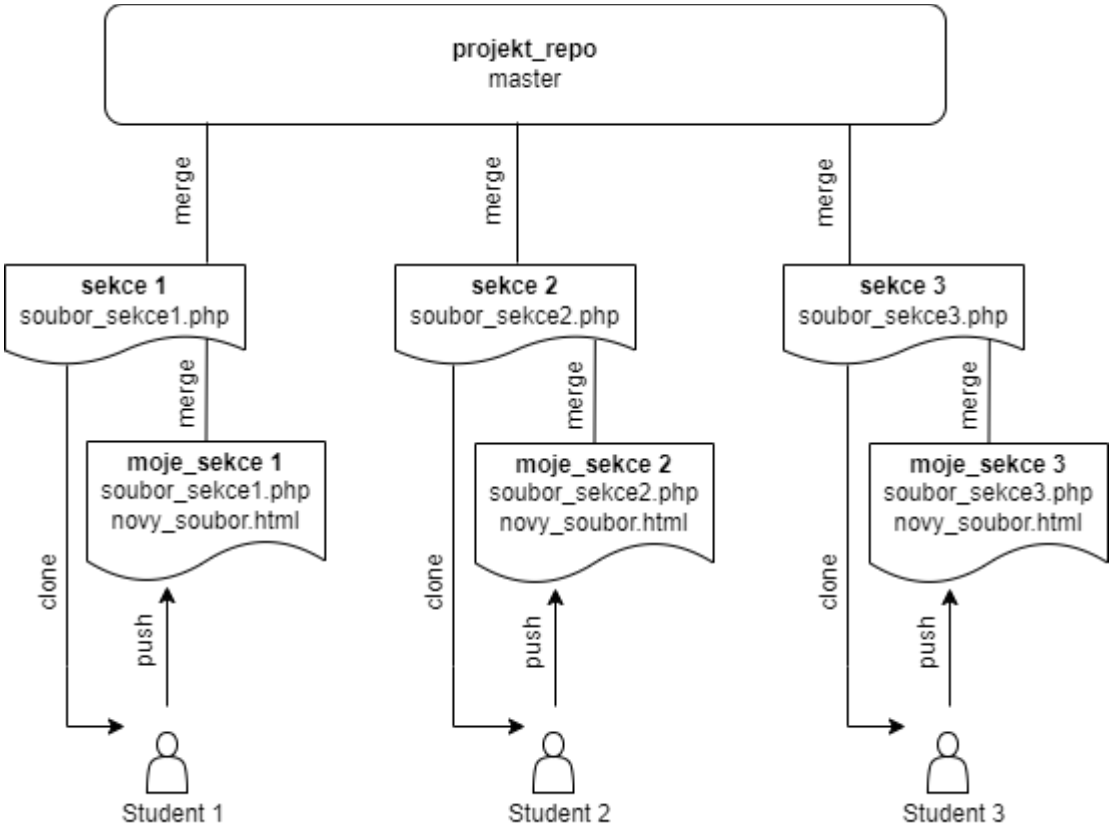
```
git add novy_soubor.html

git commit -m "Přidán nový soubor do sekce1"

git push origin moje_vetev
```

Vyučující provede kontrolu, a pokud jsou změny v pořádku, spojí studentovu větev s větví odpovídající sekci projektu pomocí příkazu „`git merge`“. Například pro spojení studentovy větve "moje_vetev" s větví „sekce1“:

```
git checkout sekce1
```

```
git merge moje_vetev
```

Pokud nebylo odevzdané řešení dostatečné, vyučující pošle zadání studentovi zpět k dopracování a novému odevzdání. Proces se opakuje až do doby, kdy jsou změny přijatelné. Postup je schematicky uveden na obr. 14.



Obr. 14: Práce studentů nad společným řešením

5.4.3 Kompletace (merge master) celého projektu

Vyučující se přepne do větve master, která je zpravidla hlavní větví projektu:

```
git checkout master
```

Poté se spojí všechny větve s hlavní větví pomocí příkazu „`git merge`“.

```
git merge sekce1 sekce2 sekce3
```

Pokud by došlo ke konfliktu během spojování, musí být vyřešen. Vyučující dále musí ověřit, že projekt funguje správně, a že nevznikly žádné nové chyby během spojování. Pokud jsou všechny větve úspěšně sloučeny, může se repozitář uzamknout a projekt je považován za dokončený. Pokud má být projekt veřejně dostupným, může být repozitář publikován na webových službách pro sdílení kódu, jako je například GitHub.

Vyučující také může vytvořit novou značku (*tag*) pro označení dosaženého milníku v rámci projektu a pro uchování verze kódu v určitém okamžiku. Může být vhodné, aby spojení všech větví do hlavní větve a kompletaci projektu provedla jedna určená osoba. V opačném případě může docházet k chybám a konfliktům, protože studenti pracují na svých větvích nezávisle a mohou mít rozdílné představy o tom, jak by měl projekt vypadat. Proto je snazší, pokud spojení všech větví do jednoho celku provede vyučující, který má odpovídající znalosti a zkušenosti.

6 Ukázkové řešení při výuce HTML

Následující ukázkové řešení bylo navrženo pro studenty 2. - 3. ročníků středních škol (obchodních akademií, gymnázií apod.) v rámci výuky informačních technologií. Ukázkové řešení zahrnuje dvě lekce a domácí úkol:

- Ukázka výukové hodiny, během které studenti budou pracovat v HTML. Jednotlivé kroky budou verzovat a odevzdávat pomocí systému Git.
- Ukázka domácího cvičení, které studenti vypracují samostatně a tím si ukotví nabyté vědomosti. Zadání bude velmi podobné jako při výukové hodině. Studenti opět budou pracovat s HTML kódem v systému Git.
- Ukázka společné práce na projektu v rámci výukové hodiny. Studenti budou pracovat na jednotlivých částech webové stránky a na závěr se jejich práce spojí do jednotného celku.

6.1 Ukázkové řešení výukové hodiny

Cílem výukové hodiny je naučit studenty používat verzovací systém Git, a přesněji, přimět studenty, aby si osvojili rutinní úkony spojené se správou verzí v tomto systému. Výuka obsluhy Gitu je poměrně obsáhlé téma a je vhodné ji kombinovat spíše s jednodušší látkou. Domnívám se však, že není vhodné učit studenty verzování pouze s ukázkovými texty, jelikož pak správě verzí nepřikládají dostatečný význam.

Jako téma pro ukázkové řešení bylo proto zvoleno pozicování elementů v HTML. Strukturovaný text ve značkovacím jazyce je poměrně jednoduchý na pochopení a zároveň je vhodný i jako příklad pro správu verzí. Navíc výuka HTML a CSS většinou probíhá právě v 2. - 3. ročnících středních škol (v závislosti na školním vzdělávacím plánu).

Doporučený maximální počet studentů v rámci paralelky je 16. Je to dáno tím, že v průběhu výuky učitel provádí i kontrolu odevzdaných řešení. Na větší počet studentů by mu již nemusela zbývat časová kapacita.

6.1.1 Ukázkové řešení výukové hodiny (dvouhodinový blok)

V průběhu první vyučovací hodiny si studenti naklonují obsah vzdáleného repozitáře. Pro tento příklad byl vypracován HTML soubor v příloze 3. Samotné zadání práce je umístěno v HTML komentáři v obsahové části zdrojového kódu.

Studenti si naklonují připravený repozitář a zpracují zadání. Vždy po dokončení jedné dílčí části zadaného úkolu vytvoří nový zápis do pracovní kopie. Na konci cvičení by měli mít svou větev se třemi zápisy (*commit*), kterou poté nahrají (*push*) do vzdáleného repozitáře. Zadání úlohy je následující:

1. Vytvořte obsah rozdělený na dvě části s šířkou 60 a 40 % šířky stránky.
2. Do části o velikosti 60 % vložte obrázek s nastavením obtékání textu zprava. Vložte libovolný text o vhodné délce tak, aby se zobrazoval vpravo vedle obrázku.
3. Do části o velikosti 40 % vložte nadpis a libovolný text o vhodné délce.

Jednotlivé body zadání vždy verzujte a uveďte v commitech vysvětlující komentář.

Postup studentů při práci ve výuce

1. Nejprve si studenti naklonují vzdálený repozitář příkazem:
   ```
   git clone ssh://user@server:/cesta/k/repozitari
   ```
2. Studenti si vytvoří vlastní větev pomocí příkazu:
   ```
   git branch nazev_vetve
   ```
   Poté se do nově vytvořené větve přepnou:
   ```
   git checkout nazev_vetve
   ```
   Alternativním postupem může být:
   ```
   git checkout -b nazev_vetve
   ```
   Tento příkaz je zkráceným ekvivalent předchozích dvou příkazů.
3. Studenti vypracují první bod zadání. Vloží do HTML layoutu v zadání kontejnery div a tím rozdělí obsah stránky na dvě části v poměru 3:2.
4. Tento krok zaverzují pomocí příkazů:
   ```
   git add .
   git commit -m "popis změn"
   ```

5. Následně studenti vypracují druhý bod zadání. Vloží do obsahové části o šířce 60 % obrázek pomocí tagu `<img>` a nastaví obtékání textu. Dále vloží libovolný text o vhodné délce.
6. Tento krok opět zaverzují pomocí příkazů „`add`“ a „`commit`“.
7. Jako třetí úkol vloží studenti do části o šířce 40 % nadpis a text o vhodné délce. Tento krok opět zaverzují příkazy „`add`“ a „`commit`“.

Vyučující během první hodiny bude se studenty řešit dotazy související zejména s problematikou verzování tak, aby studenti bez problémů zvládli vše vypracovat a správně využít příkazů systému pro správu verzí Git.

Studenti před odevzdáním zkontrolují, zda jsou všechny změny řádně připraveny příkazem „`git status`“ a odešlou své větve do vzdáleného repozitáře pomocí příkazu „`git push origin nazev_vetve`“. Vyučující si pomocí příkazu „`git branch -r`“ zkontroluje, zda byly odevzdány větve všech studentů.

Během druhé hodiny se vyučující pomocí příkazu „`git checkout nazev_vetve`“ přepne vždy do větve, kterou odevzdal konkrétní vybraný student. Vyučující zkontroluje řešení úlohy a případně studenta požádá o přepracování. V takovém případě student upraví v lokálním repozitáři svou práci a opět pomocí příkazů „`add`“ , „`commit`“ a „`push`“ odevzdá svou větev do vzdáleného repozitáře.

V závěru hodiny provede vyučující rekapitulaci nabytých znalostí a případně připomene jednotlivé kroky. Následně vyhodnotí, zda studenti látku pochopili natolik, aby mohli sami vypracovat zadání domácí úlohy. V kladném případě může studentům zadat následující domácí cvičení.

6.2 Ukázkové řešení domácí úlohy

Cílem domácího cvičení je vést studenty k opakování nových dovedností a prostřednictvím samostatné práce jim dodat sebedůvěru při ovládání systému Git. Ovládání systému Git je potřeba opakovat, aby si studenti navykli na jednotlivé opakující se úkony spojené se správou verzí.

Tato část se od řešení pro výuku liší především ošetřením repozitářů na serveru tak, aby studenti nemohli při řešení využít již odevzdané práce ostatních. Podrobný popis přípravy

repozitářů pro domácí úkoly je uveden v příloze 1. Pro modelový domácí úkol byl vypracován HTML soubor v příloze 4. Studenti budou řešit obdobný úkol jako při výuce tak, aby si získané znalosti ukotvili. Samotné zadání práce opět najdou i přímo v HTML komentáři.

Studenti si nejprve naklonují připravený repozitář a zpracují zadání. Vypracované jednotlivé body v zadání vždy verzují. Řešení by mělo obsahovat vlastní větev se čtyřmi zápisy (*commit*), kterou poté studenti nahrají (*push*) do vzdáleného repozitáře. Zadání domácího úkolu je následující:

1. Vytvořte obsah rozdělený na tři části o šířce 30, 30 a 40 % šířky stránky.
2. Do první části vložte obrázek a text o vhodné délce.
3. Do druhé části vložte obrázek a bodový seznam.
4. Do poslední části vložte nadpis a libovolný text o vhodné délce

Jednotlivé body zadání vždy verzujte a uveďte v commitech vysvětlující komentář.

Postup studentů při práci na domácím úkolu

1. Naklonovat repozitář příkazem:
   ```
   git clone student@server:path/to/student_repo.git
   ```
2. Následně vytvořit vlastní větev pomocí příkazu:
   ```
   git checkout -b nazev_vetve
   ```
3. Do obsahové části HTML layoutu vložit kontejnery div pro rozdělení obsahu na tři části s šířkou v poměru 3:3:4.
4. Tento krok zaverzovat pomocí příkazů:
   ```
   git add .
   git commit -m "popis změn"
   ```
5. Do první části o šířce 30 % vložit obrázek pomocí tagu `<img>` a text o vhodné délce. Tento krok zaverzovat.
6. Do druhé části o šířce 30 % vložit obrázek a bodový seznam. Tento krok zaverzovat.
7. Do poslední části o šířce 40 % vložit nadpis, text vhodné délky a opět verzovat.

Na závěr by studenti měli zkontrolovat, zda jsou připraveny všechny změny příkazem „`git status`“ a poté odeslat svou větev do vzdáleného repozitáře pomocí příkazu „`git push origin nazev_vetve`“.

Postup vyučujícího při zadání a kontrole domácího úkolu

Vyučující před zadáním domácího úkolu vytvoří repozitáře pro jednotlivé studenty a nastaví jim potřebná práva. Nejprve je nutné vytvořit nový adresář s využitím příkazů:

```
mkdir homework_repos
chmod 775 homework_repos
cd homework_repos
```

Následně pro každého studenta:

```
git init student1_repo.git
chmod 770 student1_repo.git
chown -r student1: student1 student1_repo.git
```

Pedagog poté vloží do každého jednotlivého repozitáře soubor index.html (soubor je k dispozici v příloze 4) a zadá studentům domácí úlohu spolu s termínem vypracování.

Po uplynutí termínu na vypracování domácího úkolu provede vyučující kontrolu odevzdané práce. Za tím účelem se přepne do složky s repozitáři příkazem „`cd homework_repos`”. Následně projde jednotlivé repozitáře např. příkazem „`cd student1_repo.git`” s tím, že se vždy přepne přímo do studentovy větve příkazem „`git checkout nazev_vetve`“. V případně nevyhovujícího řešení může být student požádán o přepracování odevzdané práce.

6.3 Ukázkové řešení společné práce studentů

Cílem společné práce studentů je osvojit si principy vývoje používané v profesionálních týmech. Během této aktivity studenti získají hlubší znalosti verzování a jeho praktického využití. Zároveň se studenti naučí efektivně spolupracovat a komunikovat v rámci společného projektu. Tímto způsobem se připraví na budoucí vývojářské výzvy a posílí své schopnosti spolupracovat a komunikovat v týmovém prostředí.

Doporučený maximální počet studentů v rámci paralelky je 16. Důvodem je opět časová náročnost kontroly odevzdaných řešení vyučujícím. Navíc je zde nutné počítat s časem navíc při řešení kolizí a slučování celého projektu. Právě z tohoto důvodu je zadání voleno tak, aby studenti zvládli úlohy zpracovat do přibližně 30 minut od zadání.

6.3.1 Ukázkové řešení společné práce studentů (dvouhodinový blok)

Vyučující nejprve rozdělí studenty na týmy po čtyřech studentech. Pro každý tým určí vyučující téma webové stránky, jejíž návrh bude tým zpracovávat (např: květinářství, truhlářství, kovářství a kadeřnictví). Následně se přiřadí každému týmu jeden repozitář, který bude obsahovat soubor index.html (dostupný v příloze 5).

V průběhu první vyučovací hodiny si studenti naklonují obsah vzdáleného repozitáře. Samotné zadání práce studentů je umístěno v HTML komentáři. V následujícím textu je uveden pouze postup pro jeden vybraný tým, přičemž každému studentovi bude přidělena jedna část (odpovídající jedné sekci HTML souboru v příloze 5) projektu k řešení. Pro tým o čtyřech studentech může být rozvržena práce dle jednotlivých sekcí následujícím způsobem (viz také obr. 15):

Jednotlivé kroky vždy verzujte a v commitu uveďte vhodný komentář.

Zadání pro sekci 1:

1. Vytvořte obsah pro stránku „Úvod“ v sekci 1. 2. Obsahovou část rozdělte na tři části a do každé části vložte obrázek a text vztahující se k tématu. 3. Upravte odkaz v menu tak, aby odkazoval na id sekce-1 a zobrazoval popis „Úvod“.

Zadání pro sekci 2:

1. Vytvořte obsah pro stránku „O nás“ v sekci 2. 2. Obsahovou část rozdělte na dvě části a do každé části vložte vhodný obsah (o nás). 3. Upravte odkaz v menu tak aby odkazoval na id sekce-2 a zobrazoval popis „O nás“.

Zadání pro sekci 3:

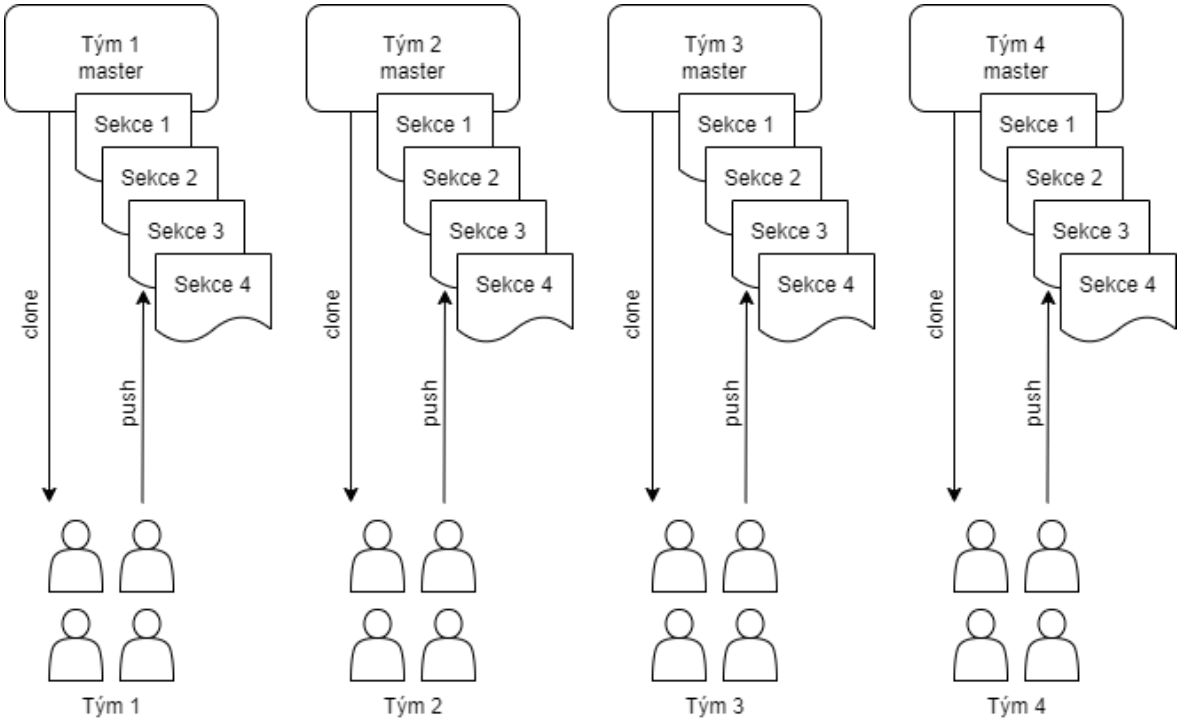
1. Vytvořte obsah pro stránku „Portfolio“ v sekci 3.
2. Obsahovou část rozdělte na čtyři části a do každé části vložte vhodný obsah (portfolio).
3. Upravte odkaz v menu tak, aby odkazoval na id sekce-3 a zobrazoval popis „Portfolio“.

Zadání pro sekcí 4:

1. Vytvořte obsah pro stránku „Kontakt“ v sekci 4.
2. Obsahovou část rozdělte na dvě části a do první části vložte vhodný obsah (kontakty). Do druhé části vložte Google mapku vhodné velikosti.
3. Upravte odkaz v menu tak, aby odkazoval na id sekce-4 a zobrazoval popis „Kontakt“.

Postup studentů při společné práci

1. Naklonovat repozitář příkazem:
   ```
   git clone student@server:path/to/team1_repo.git
   ```
2. Vytvořit svou vlastní větev:
   ```
   git checkout -b nazev_vetve
   ```
3. Do obsahové části své sekce vložit obsah dle zadání.
4. Provedenou práci zaverzovat pomocí příkazů:
   ```
   git add .
   git commit -m "popis změn"
   ```
5. V hlavičce webu upravit odkaz (*anchor point*) na obsahovou část své sekce a provedenou práci opět zaverzovat.
6. Provést kontrolu, zda jsou všechny změny řádně okomentovány a připraveny („`git status`“) a odeslat větve do vzdáleného repozitáře pomocí příkazu
   ```
   git push origin nazev_vetve
   ```



Obr. 15: Rozvržení zadání při společné práci studentů ve čtyřech týmech

Postup vyučujícího při řešení společné práce – příprava a zadání

V rámci přípravy na společnou práci je třeba studentům nejprve vytvořit prostředí, ve kterém budou pracovat.

1. Vyučující vytvoří pro týmy potřebný počet repozitářů např. příkazem: `git init team1_repo` (pro první tým)
2. Přepne se do repozitáře příkazem „`cd team1_repo`“ a vloží připravený soubor index.html (příloha 5)
3. Zadá studentům téma webového projektu.

Během první vyučovací hodiny budou studenti pracovat na svých projektech. Vyučující bude řešit případné dotazy a posuzovat návrhy realizace. Dále bude řešit organizaci spolupráce v týmech. V závěru první hodiny provede kontrolu dokončené práce jednotlivých týmů a případně požádá o nápravu.

Postup vyučujícího při řešení společné práce – kontrola a sloučení projektů

Během druhé hodiny provede vyučující sloučení jednotlivých částí projektů a otestuje jejich funkčnost. Postup pro každý tým může být následující:

1. Vyučující se přepne do repozitáře daného týmu:
   `cd team1_repo` (pro první tým)
2. Zkontroluje, zda jsou odevzdány větve celého týmu:
   `git branch -r`
3. Projde jednotlivé větve příkazem:
   `git checkout nazev_vetve`
   a zkontroluje vypracování zadání. V případě nedostatků požádá studenta o přepracování.
4. Následně provede postupné sloučení všech větví s hlavní větví (*master*) příkazy:
   `git checkout master`
   `git merge nazev_vetve`
   Během tohoto kroku může docházet ke konfliktům, které bude nutné vyřešit.
5. Poté vyučující zapíše provedené změny s vhodným komentářem pomocí příkazu:
   `git commit -m "sloučení nazev_vetve s master"`
6. Na závěr je vhodné ověřit, zda projekt funguje správně a sloučení proběhlo úspěšně.

V závěru hodiny provede vyučující rekapitulaci nabytých znalostí a případně připomene jednotlivé kroky.

7 Doporučení pro práci s Git ve výuce

Verzovací systémy včetně Gitu byly původně vytvořeny ke správě verzí zdrojového kódu, a právě to je oblast, ve které excelují. Vzhledem k tomu, že výuka práce s verzovacími systémy je poměrně komplexní a časově náročná, je pravděpodobně nejlepším řešením ji přirozeně propojit právě s výukou programování, kde se výhody verzování projeví nejvýrazněji.

Studenti by měli být motivováni k používání verzovacích systémů při programování tak, aby si zvykli na rutinní práci spojenou s verzováním, a mohli tak efektivněji pracovat na svých projektech. Pro snazší pochopení principů verzovacích systémů (zejména nejpopulárnějšího systému Git) lze v hodinách využít i výukové programy, které byly představeny v pododdílu 3.2.4 této práce.

Pokud vyučující nemá dostatečné znalosti ovládání Linuxu či prostě nechce provozovat vlastní server, může s výhodou využít hostingových služeb jako je GitHub nebo GitLab, které nabízí snadnou a přehlednou správu repozitářů. Kromě ušetřené práce se správou serveru lze využít i speciální programy a koncepty, které platformy GitHub a GitLab poskytují.

Jedním z takových programů je Github Classroom, který umožňuje vyučujícím i studentům získat přístup ke všem nástrojům a funkcím Githubu zdarma, včetně neomezeného množství soukromých repozitářů a dalších výhod. Využití Github Classroom může být pro studenty i vyučující velmi užitečné, zejména pokud se jedná o rozsáhlejší projekty. Díky této službě mohou být projekty lépe organizovány a řízeny, což znamená snazší spolupráci a efektivnější sledování vývoje celého projektu.

Další výhodou cloudových služeb pro hosting repozitářů je možnost využít služeb CI/CD pipeline. Ačkoliv žáci samotní tyto techniky při své práci nemusí přímo využít, mohou být využity vyučujícím např. k automatizované kontrole odevzdaných řešení. Studentům je zároveň vhodné principy CI/CD alespoň předvést, neboť v praxi se s nimi lze běžně setkat a automatizace procesů je důležitou oblastí komerčního programování.

Celkově je klíčové nezapomínat na Git při výuce programování a využívat jeho výhod pro lepší správu kódu a příjemnější spolupráci mezi studenty a vyučujícími.

7.1 Nejčastější problémy s používáním Gitu ve výuce

Jedním z nejčastějších problémů je náročnost ovládání Gitu. Git má poměrně složitou syntaxi a některé jeho koncepty mohou být pro začínající uživatele těžko pochopitelné. Studenti zpravidla nemají zpočátku žádné znalosti toho, jak Git správně používat. Mohou tak při výuce zažívat zvýšenou frustraci nebo pociťovat nedostatek motivace, jelikož výhody verzovacích systémů jim nemusí být na první pohled patrné.

Pomoci může, pokud vyučující s kolektivem k výhodám verzovacích systémů dospěje po společné diskuzi, která se může vyvíjet podobně, jako úvaha popsaná v úvodu této práce (verzování pomocí undo/redo → označování verzí v názvech souborů → revize v textových editorech → specifika zdrojového kódu → verzovací systémy).

Domnívám se, že je nejvýhodnější představit Git co nejdříve, ještě před výukou programování. Studenti by měli porozumět základním konceptům, jako jsou například větve, zápisy (*commit*) a nahrávání či stahování materiálů ze vzdálených repozitářů (*push/pull)*. Pokud studenti nejsou dostatečně obeznámeni s těmito koncepty, budou snáze dělat chyby, které mohou způsobovat problémy při výuce a spolupráci na zadaných projektech.

Obecným problémem může být rovněž nedostatek času pro výuku Gitu. Výuka programování často zahrnuje mnoho různých konceptů a technologií, takže studenti mohou mít problém najít dostatek času ještě na Git. Pokud studenti nepoužívají Git pravidelně, mohou postupně zapomenout dříve naučené koncepty a mít následně potíže při jejich praktické aplikaci.

Z těchto důvodů je nezbytné studenty nejen naučit základní principy Gitu, ale poté i dohlédnout na to, aby získali opakovaným nácvikem důvěru ve své schopnosti. Pokud spojíme výuku programování s výukou Git, tak je třeba část věnovanou verzovacímu systému nepodceňovat, protože řešení individuálních problémů s Gitem u studentů jinak zabere mnoho času určeného k samotné výuce programování.

Pokud se studenti naučí používat Git pravidelně a vytvoří si návyky spojené se správou verzí, může to výuce programování prospět. Konečně, i pokud verzovací systémy v hodinách použijeme pouze v omezeném měřítku, celkově tím práci studentů výrazně přiblížíme principům práce užívaným v komerční sféře.

8 Závěr

Systémy pro správu verzí jsou důležitým nástrojem při vývoji softwaru. Jejich začlenění do výuky však žáky nepřipravuje pouze na úzce zaměřenou práci vývojáře, ale rozvíjí řadu dalších digitálních kompetencí. Při práci s verzovacími systémy dochází k osvojení návyků spojených s využitím konkrétních prvků informačních technologií, jako je souborový systém, grafické uživatelské rozhraní nebo příkazová řádka. Zároveň jde o nácvik společné práce nad jedním větším celkem, při které jedinec musí dodržovat nastavená pravidla tak, aby mohl tým (třída, skupina žáků) efektivně spolupracovat.

V rámci této práce byly zkoumány verzovací systémy nejprve z obecného teoretického pohledu. Představeny byly dva hlavní typy verzovacích systémů: centralizované a decentralizované. Pro lepší možnost porovnání obou přístupů byli blíže prozkoumáni dva konkrétní zástupci obou skupin: centralizovaný systém Subversion (SVN) a distribuovaný systém Git. Bylo zjištěno, že práce s oběma systémy je z uživatelského pohledu podobná a používáním jednoho verzovacího systému získá uživatel znalosti, které jsou přenositelné i na systémy jiné.

Pro bližší představení byl vybrán systém Git, který je v oboru informačních technologií nejpoužívanější. Byla provedena rešerše vhodných doplňků, které by bylo možné využít při výuce obsluhy Git. Systém Git je komplexní a obsahuje velké množství funkcí a nástrojů, doplňky však mohou orientaci v systému usnadnit. Zároveň bylo zjištěno, že je ke Git dostupných jen málo přímo aplikovatelných metodik výuky. Proto je v druhé části práce představeno modelové řešení výuky, včetně metod distribuce, příkladových materiálů a postupů usnadňujících společnou práci studentů.

V poslední části práce rozebírá problematiku týmových projektů, jejich organizace a zpětné kompletace do společného celku. Zatímco první část práce se věnuje i technickému pozadí verzovacích systémů, druhá od něj naopak v maximální míře upouští a poskytuje praktické postupy toho, jak lze do výuky verzovací systémy zavést. Závěrem jsou v práci uvedeny nejčastější obtíže při výuce Git v kolektivech, které byly sesbírány při rešerších provedených pro předcházející texty. Výsledkem je poznání, že ačkoliv je zavedení verzovacích systémů do výuky náročné, přináší studentům i vyučujícím hodnotné zkušenosti.

9 Seznam použitých informačních zdrojů

1. ROCHKIND, Marc J. The source code control system. IEEE Transactions on Software Engineering [online]. 1975, SE-1(4), 364-370 [cit. 2023-03-13]. ISSN 0098-5589. Dostupné z: doi:10.1109/TSE.1975.6312866
2. PETRAZICKIS, Leo. MySpace's death spiral due to no test environment, no source control, no code review [online]. 2011 [cit. 2023-03-13]. Dostupné z: https://lpetr.org/2011/03/29/myspaces-death-spiral-due-to-no-test-environment-no-source-control-no-code-review/
3. SCOTT, Chacon a Ben STRAUB. Pro Git. Verze 2.1.359-2-g27002dd. Apress, 2022. ISBN 9781484200766.
4. KRÁTKÝ, Robert a přispěvatelé. Repozitář – výkladový slovník. AbcLinuxu [online]. 2009 [cit. 2023-03-09]. Dostupné z: https://www.abclinuxu.cz/slovnik/repozitar
5. ERNST, Michael. Version control concepts and best practices. Advice compiled by Michael Ernst [online]. University of Washington, 2022 [cit. 2023-03-09]. Dostupné z: https://homes.cs.washington.edu/~mernst/advice/version-control.html
6. RAWAT, Ashok. What is a version control system?: Basics knowledge and understanding of Version Control System [online]. 2022 [cit. 2023-03-09]. Dostupné z: https://medium.com/@ashokrawat086/what-is-a-version-control-system-2f3509066b72
7. KUČERA, František. Distribuované verzovací systémy [online]. AbcLinuxu, 2011 [cit. 2023-02-21]. Dostupné z: https://www.abclinuxu.cz/clanky/distribuovane-verzovaci-systemy-uvod-1
8. SIMMONS, TJ. Understanding the CI/CD Pipeline: What It Is, Why It Matters [online]. Plutora, 2022 [cit. 2023-03-02]. Dostupné z: https://www.plutora.com/blog/understanding-ci-cd-pipeline
9. Stack Overflow Developer Survey 2022: Version control systems [online]. Stack Overflow [cit. 2023-01-14]. Dostupné z: https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems

10. PILATO, C. Michael, Ben COLLINS-SUSSMAN a Brian W. FITZPATRICK. Version control with subversion [online]. Beijing: O'Reilly, 2008 [cit. 2023-01-28]. ISBN 05-965-1033-0. Dostupné z: https://svnbook.red-bean.com/
11. KÜNG, Stefan, Lübbe ONKEN a Simon LARGE. TortoiseSVN documentation: Subversion klient pro Windows. TortoiseSVN [online]. 2022 [cit. 2023-03-02]. Dostupné z: https://tortoisesvn.net/docs/release/TortoiseSVN_cs/index.html
12. ANTONY, Anish. 5 Different Git Workflows [online]. Medium, 2021 [cit. 2023-03-02]. Dostupné z: https://medium.com/javarevisited/5-different-git-workflows-50f75d8783a7
13. Introduction to GitLab Flow [online]. GitLab, 2023 [cit. 2023-03-05]. Dostupné z: https://docs.gitlab.com/ee/topics/gitlab_flow.html
14. VIRTANEN, Tommi. Git for Computer Scientists. Eagain [online]. 2007 [cit. 2023-03-05]. Dostupné z: https://eagain.net/articles/git-for-computer-scientists/
15. Git for Windows [online]. 2023 [cit. 2023-01-14]. Dostupné z: https://gitforwindows.org/
16. MACKAL, Matt. Towards a Better SCM: Revlog and Mercurial. Mercurial SCM [online]. 2006 [cit. 2023-03-04]. Dostupné z: https://www.mercurial-scm.org/wiki/Presentations?action=AttachFile&do=get&target=ols-mercurial-paper.pdf
17. Mercurial SCM [online]. 2023 [cit. 2023-03-04]. Dostupné z: https://www.mercurial-scm.org/guide
18. DONOVAN, Ryan. Beyond Git: The other version control systems developers use. StackOverflow [online]. 2023 [cit. 2023-03-06]. Dostupné z: https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/
19. Gitea: Git with a cup of tea [online]. Gitea, 2023 [cit. 2023-04-18]. Dostupné z: https://github.com/go-gitea/gitea
20. Caddy-gitea [online]. 42wim, 2023 [cit. 2023-04-18]. Dostupné z: https://github.com/42wim/caddy-gitea

21. TECHKNOWLOGICK. Open source sustainment and the future of Gitea [online]. [cit. 2023-05-14]. Dostupné z: https://blog.gitea.io/2022/10/open-source-sustainment-and-the-future-of-gitea/
22. SourceTree: Free Git GUI for Mac and Windows [online]. Atlassian, 2023 [cit. 2023-01-14]. Dostupné z: https://www.sourcetreeapp.com/
23. GitHub Desktop: Simple collaboration from your desktop [online]. GitHub, 2023 [cit. 2023-01-14]. Dostupné z: https://desktop.github.com/
24. GitKraken Legendary Git Tools [online]. Axosoft, 2023 [cit. 2023-01-14]. Dostupné z: https://www.gitkraken.com/
25. TortoiseGit: Windows Shell Interface to Git [online]. TortoiseGit, 2023 [cit. 2023-01-14]. Dostupné z: https://tortoisegit.org/
26. Git Extensions: Standalone UI tool for managing git repositories [online]. GitHub Pages, 2023 [cit. 2023-03-16]. Dostupné z: https://gitextensions.github.io/
27. Git-ftp: uploads to FTP servers the Git way [online]. 2022 [cit. 2023-02-27]. Dostupné z: https://github.com/git-ftp/git-ftp
28. Git-it: Desktop App for Learning Git and GitHub [online]. 2023 [cit. 2023-01-15]. Dostupné z: https://github.com/jlord/git-it-electron
29. LearnGitBranching [online]. 2023 [cit. 2023-01-15]. Dostupné z: https://github.com/pcottle/learnGitBranching
30. Oh My Git!: An interactive Git learning game! [online]. 2022 [cit. 2023-01-16]. Dostupné z: https://github.com/git-learning-game/oh-my-git
31. GitHub Minesweeper: Let a bot teach you the GitHub Flow. *Profy* [online]. 2021 [cit. 2023-03-17]. Dostupné z: https://profy.dev/project/github-minesweeper
32. Digitální ekonomika v číslech [online]. Český statistický úřad, 2022 [cit. 2023-02-27]. Dostupné z: https://www.czso.cz/csu/czso/digitalni-ekonomika-v-cislech-2022#
33. GitLab for the education industry [online]. 2023 [cit. 2023-03-11]. Dostupné z: https://about.gitlab.com/solutions/education/
34. GitHub Classroom [online]. 2023 [cit. 2023-03-12]. Dostupné z: https://classroom.github.com/

10 Seznam příloh

Příloha 1 – Příprava Git serveru

Příloha 2 – Alternativní postupy pro vybraná aplikační prostředí

Příloha 3 – Soubor HTML pro ukázkové řešení výuky

Příloha 4 – Soubor HTML pro ukázkové řešení domácího úkolu

Příloha 5 – Soubor HTML pro ukázkové řešení společné práce studentů

Příloha 1: Příprava Git serveru

Instalace a nastavení Git na Linux serveru

Postup základní konfigurace a nastavení systému Git na vlastním Linux serveru (distribuce Debian/Ubuntu) je následující:

1) Otevřeme terminál na Linux serveru. Git se instaluje pomocí příkazu:
   `sudo apt-get install git`
2) Nastavíme základní konfiguraci Git pomocí příkazů:
   `git config --global user.name "Vaše jméno"`
   `git config --global user.email "vaše@emailová.adresa"`
3) Vytvoříme prázdný adresář pro naše Git repozitáře. Například:
   `mkdir /srv/git`
4) Přejdeme do adresáře Git repozitářů:
   `cd /srv/git`
5) Vytvoříme nový prázdný Git repozitář pomocí příkazu:
   `git init --bare <název-repozitáře>.git`
6) Nastavíme přístupová práva (vlastník může číst, zapisovat, spouštět) pro Git repozitář pomocí příkazu:
   `chmod -R 755 <název-repozitáře>.git`

Git repozitář na serveru je tímto připraven k použití.

Řešení pomocí GitHub/GitLab

Nechceme-li spravovat vlastní server, můžeme využít poskytovatelů hostingových služeb pro správu verzí, jako je GitHub nebo GitLab.

Nastavení vzdáleného repozitáře ve službě GitLab

Nejprve přejdeme na domovskou stránku poskytovatele služeb GitLab (www.gitlab.com) a vytvoříme si účet. Po přihlášení do našeho účtu provedeme následující kroky:

1) Na hlavní obrazovce s rozcestníkem jednotlivých možností vybereme možnost „Create a project“.

2) Jelikož nemáme k dispozici šablonu ani existující projekt, zvolíme možnost „Create blank project“.
3) Budeme přesměrováni na stránku pro vytvoření nového projektu, kde vyplníme jméno projektu a volitelně popis projektu.
4) Zvolíme úroveň viditelnosti repozitáře – na výběr jsou možnosti „private“ (soubory budou přístupné pouze pro členy projektu) a „public“ (soubory budou viditelné pro všechny uživatele služby GitLab).
5) V případě, že chceme, aby repozitář již při vytvoření obsahoval soubor s delším popisem našeho projektu, zaškrtneme možnost „Initialize repository with a README“.
6) Klikneme na tlačítko „Create project“. Tím je náš repozitář vytvořen a je možné jej stáhnout do lokálního prostředí (cestu nezbytnou pro klonování získáme přes tlačítko „Clone“ na stránce s repozitářem).

Nastavení vzdáleného repozitáře ve službě GitHub

Přejdeme na stránku poskytovatele služeb GitHub (www.github.com) a vytvoříme si účet. Po přihlášení do našeho účtu provedeme následující kroky, které jsou obdobné jako u služby GitLab:

1) V pravém horním rohu obrazovky najdeme ikonku „+“ a vybereme možnost „New repository“.
2) Na stránce s konfigurací repozitáře nastavíme jméno projektu, viditelnost a volitelně zaškrtneme vytvoření souboru README. Pokud je to nezbytné, můžeme rovněž vybrat typ licence, pod kterým bude kód repozitáře dostupný pro veřejnost.
3) Vybereme možnost „Create repository“.
4) Repozitář je vytvořen a lze jej využít k další práci.

Instalace a nastavení Git na straně klienta

Po nastavení serveru je nezbytné nakonfigurovat systém Git ještě na klientských stanicích. K tomu lze využít následující postup:

1) Otevřeme terminál na klientském zařízení.
2) Stáhneme instalační soubor pro Git z oficiálních stránek (https://git-scm.com/download/).
3) Spustíme instalační soubor a nainstalujeme Git s výchozími nastaveními.
4) Otevřeme příkazový řádek a nastavíme jméno a email, které budou spojovány se změnami, které v Gitu uživatel daného zařízení provede:
   ```
   git config --global user.name "jméno"
   git config --global user.email "emailová@adresa"
   ```
5) Poté lze vytvořit první repozitář pomocí příkazu „`git init`“. Tento příkaz vytvoří nový prázdný repozitář v aktuálním adresáři.
6) Pokud chceme naklonovat existující repozitář, použijeme příkaz „`git clone <url>`“. URL může být adresa vzdáleného repozitáře na serveru nebo cesta k lokálnímu repozitáři.

Nyní je již možné začít pracovat s repozitářem pomocí příkazů „`git add, git commit, git push`“ a dalších.

Příloha 2: Alternativní postupy pro vybraná aplikační prostředí

Alternativní postupy pro stažení zadání se serveru

Klonování repozitáře v GUI SourceTree

V rámci výuky lze s výhodou využít grafického uživatelského rozhrání SourceTree, které bylo představeno v oddílu 5.1.1. Postup pro naklonování repozitáře do lokálního adresáře je v doplňku SourceTree následující:

1) Otevřeme aplikaci SourceTree a klikneme na tlačítko „Clone“.
2) Zadáme URL repozitáře, který chceme klonovat, do pole „Source Path / URL“.
3) Vybereme adresář, do kterého chceme repozitář klonovat (pole „Destination Path“).
4) Klikneme na tlačítko „Clone“.

SourceTree stáhne obsah repozitáře a vytvoří jeho lokální kopii. Pro vytvoření nové větve stačí kliknout pravým tlačítkem myši na název repozitáře a zvolit možnost „Create Branch“. Tím je nová větev vytvořena a můžeme začít pracovat na řešení zadané úlohy.

Klonování repozitáře v aplikaci PhpStorm

Aplikace PhpStorm je jedním z vývojových prostředí pro programování webových aplikací. Prostředí obsahuje i integrované funkce pro ovládání systému Git. V aplikaci PhpStorm probíhá klonování repozitáře takto:

1) Otevřeme PhpStorm a klikneme na tlačítko „Check out from Version Control“ v úvodním okně nebo najdeme příslušnou položku v menu: „File – New – Project from Version Control – Git“.
2) V novém dialogovém okně vyplníme adresu repozitáře v poli „Git Repository URL“ a klikneme na tlačítko „Clone“.
3) Nyní vytvoříme vlastní větev projektu kliknutím na tlačítko „Git“ v pravém dolním rohu PhpStormu, kde vybereme možnost „Branches“.
4) Klikneme na tlačítko „New Branch" a zadáme název nové větve.
5) Nově vytvořenou větev si stáhneme pomocí tlačítka „Checkout“ vedle jejího názvu.
6) Vyčkáme, dokud PhpStorm nezkopíruje obsah nové větve do lokálního adresáře.

Po dokončení klonování se zobrazí nabídka, zda chceme otevřít nový projekt v PhpStormu. Zvolíme možnost „Yes“.

Klonování repozitáře v aplikaci Visual Studio Code

Visual Studio Code je dalším z populárních vývojových prostředí, ve kterém lze programovat v celé řadě jazyků. Vzdálený repozitář si ve vývojovém prostředí Visual Studio Code naklonujeme pomocí následujících kroků:

1) Otevřeme aplikaci Visual Studio Code a zvolíme v menu „File“ možnost „Open Folder“ (otevřít složku).
2) Vybereme umístění, kam chceme repozitář naklonovat.
3) Otevřeme terminál v aplikaci Visual Studio Code, a to buď přes menu „Terminal - New Terminal“, nebo klávesovou zkratkou „Ctrl + Shift + ’“ .
4) V terminálu zadáme příkaz „`git clone`“ s adresou vzdáleného repozitáře.

Po dokončení klonování se nově vytvořený adresář automaticky otevře v aplikaci Visual Studio Code. Pro vytvoření vlastní větve projektu zadáme do terminálu příkaz „`git branch`“ s názvem nové větve. Po přepnutí do nové větve (příkaz „`git checkout`“) můžeme pracovat.

Alternativní postupy pro řešení zadané úlohy

Popis metodiky práce pomocí GUI SourceTree

Postup práce s GUI SourceTree je po naklonování repozitáře následující:

1) Vytvoříme si novou větev pro naše změny kliknutím na tlačítko „Create a new branch“ v horní části okna.
2) Provedeme změny (vyřešíme zadání), které budeme ukládat do nové větve.
3) K zapsání změn musíme opět nejprve přidat soubory k danému zápisu (*commitu*) do *staging area*. To lze udělat kliknutím na tlačítko „Stage all“ v dolní části okna.
4) Vytvoříme zápis s popisem změn kliknutím na tlačítko „Commit“.

Popis metodiky práce v aplikaci PhpStorm

Pro řešení zadané úlohy a odevzdání řešení do vzdáleného repozitáře v prostředí PhpStorm lze využít následující postup:

1) Klikneme na tlačítko „VCS“ v horní nástrojové liště a z nabídky vybereme možnost „Git“.
2) Otevřeme okno „Git Branches“ kliknutím na ikonu v pravém dolním rohu obrazovky (lze využít i klávesové zkratky „Ctrl + Shift + A“) a vyhledáme větev, do které chceme nahrát své změny.
3) Pokud již větev existuje, dvojklikem na její název se do ní můžeme přepnout (*checkout*). Pokud neexistuje, klikneme na tlačítko „New Branch“ a novou větev si vytvoříme.
4) Provedeme požadované změny (vyřešíme zadanou úlohu).
5) Klikneme na tlačítko „Commit“ v dolním panelu PhpStormu tak, aby se zobrazilo okno „Commit Changes“.
6) Ujistíme se, že jsou vybrány všechny soubory, které chceme nahrát do repozitáře a napíšeme vysvětlující zprávu do pole „Commit Message“.
7) Klikneme na tlačítko „Commit and Push“ v dolním rohu okna, abychom odeslali své změny do repozitáře.

Po dokončení odeslání změn bude řešení distribuováno do existujícího repozitáře na serveru.

Popis metodiky práce v aplikaci Visual Studio Code

Postup pro řešení zadané úlohy a jejího odeslání do repozitáře je v této aplikaci následující:

1) Otevřeme projekt v aplikaci Visual Studio Code.
2) Klikněte na tlačítko „Source Control“ v levé liště.
3) Vybereme ikonu „Create a new branch“ a zadáme název pro naši novou větev.
4) V aplikaci Visual Studio Code provedeme potřebné změny, jako jsou úpravy kódu, přidání nebo odebrání souborů.
5) V seznamu změn na levé straně obrazovky vybereme soubory, které chceme přidat ke commitu, a klikneme na tlačítko „Stage Changes“.
6) Do textového pole „Message“ napíšeme popis provedených změn.
7) Vybereme tlačítko „Commit“.

Alternativní postupy pro odevzdání řešení na server

Nahrání změn do Git repozitáře – GUI SourceTree

Odevzdání práce je v GUI SourceTree otázkou několika málo kliknutí:

1) Otevřeme repozitář v aplikaci Sourcetree a přepneme se do vlastní větve, ve které máme uloženy změny.
2) Zkontrolujeme, zda jsou změny správně zapsány do commitů.
3) Stiskneme tlačítko „Push“ v horní liště, vybereme správnou větev a vše potvrdíme opět tlačítkem „Push“ v dialogovém okně.

Nahrání změn do Git repozitáře – PhpStorm

Jak jsme si ukázali v oddílu 6.1.2, v aplikaci PhpStorm lze provést commit společně s přenesením dat na server (*push*). Pokud se nám to nehodí, lze operaci push provést i zvlášť:

1) Otevřeme projekt v aplikaci PhpStorm a přepneme se do vlastní větve.
2) Zkontrolujeme, zda jsou změny správně zapsány do commitů.
3) Klikneme na tlačítko „VCS“ v horní liště a vybereme možnost „Git – Push“, zvolíme správnou větev a volbu potvrdíme.

Nahrání změn do Git repozitáře – Visual Studio Code

Odevzdání řešení ve Visual Sutio Code se podobá postupu při použití systému Git bez doplňků:

1) Otevřeme projekt v aplikaci Visual Studio Code a přepneme se do vlastní větve.
2) Zkontrolujeme, zda jsou změny správně zapsány do commitů.
3) Otevřeme terminál a zadáme příkaz „`git push origin "nazev_vetve"`“.

Příloha 3: soubor index.html pro ukázkové řešení výuky

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>pozicování v css</title>  <!-- nezapomenout vyplnit -->
  <style type="text/css">     /* komentar v css sekci */
    *{
        margin: 0px;   /* resetovani prednastavenych stylu prohlizece*/
        padding: 0px;
        position: relative;
        box-sizing: border-box; /* zrušeni přičitáni paddingu a borderu k
velikosti */
      }

      #hlavicka, #telicko, #paticka {
        margin: 0px auto;
        width: 960px;
      }

      #kontainer-1 {
        background: rgb(7, 74, 161);
        padding-top: 30px;
      }

       #kontainer-2 {
        background: rgb(230, 230, 230);
      }

       #kontainer-3 {
        background: rgb(60, 60, 60);
        padding-bottom: 40px;
      }
```

```
#hlavicka {
background: rgb(5, 113, 255);
border: solid 1px rgb(205 201 201);
border-bottom: none;
}

#hlavicka h1 {
   padding: 20px 0px;
   text-align: center;
}

#menu {
  background: rgb(7, 93, 207);
}

#menu ul {
 display: flex;
}

#menu ul li{
 display: inline-block;
}

#menu ul li a {
   text-decoration: none;
   height: 28px;
   width:  80PX;
   display: block;
   border: solid 1px rgb(1, 67, 155);
   text-align: center;
   line-height: 25px;
   background: rgb(7, 93, 207);
   color: white;
}
```

```
    #menu ul li a:hover {
      background: green;
      color: white;
    }

    #telicko {
       background: white;
       border-left: solid 1px rgb(160, 159, 159);
       border-right: solid 1px rgb(160, 159, 159);
    }

    #paticka {
     height: 100px;
     background: grey;
     text-align: center;
     padding: 40px;
     border: solid 1px black;
     border-top: none;
   }
</style>
</head>
<body>
  <div id="kontainer-1">
      <div id="hlavicka">
          <div id="menu">
              <ul>
                <li>
                   <a href="/index.html">odkaz 1</a>
                </li>
                <li>
                   <a href="/index.html">odkaz 2</a>
                </li>
                <li>
```

```
                    <a href="/index.html">odkaz 3</a>
                  </li>
                </ul>
            </div>
            <h1>Nadpis stránky</h1>
        </div>
    </div>
    <div id="kontainer-2">
        <div id="telicko">
<!--
1.    Vytvořte obsah rozdělený na dvě části s šířkou 60 a 40 % šířky stránky.
2.    Do části o velikosti 60 % vložte obrázek s nastavením obtékání textu zprava. Vložte libovolný text o vhodné délce tak, aby se zobrazoval vpravo vedle obrázku.
3.    Do části o velikosti 40 % vložte nadpis a libovolný text o vhodné délce.
Jednotlivé body zadání vždy verzujte a uveďte v commitech vysvětlující komentář.
-->
        </div>
      </div>
    <div id="kontainer-3">
        <div id="paticka">
            copyright   &#169;
        </div>
    </div>
  </body>
</html>
```

Příloha 4: soubor index.html pro ukázkové řešení domácího úkolu

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>du - pozicování</title>
  <style type="text/css">
    *{
        margin: 0px;
        padding: 0px;
        position: relative;
        box-sizing: border-box;
     }

    body {
       background: rgb(7, 74, 161);
    }

     #main {
        margin: 0px auto;
        width: 960px;
        padding: 30px 0px;
     }

     #hlavicka {
        background: rgb(5, 113, 255);
     }

     #hlavicka h1 {
       padding: 30px 0px;
       text-align: center;
      }
```

```
#menu {
  background: rgb(7, 93, 207);
}

#menu ul {
 display: flex;
}

#menu ul li{
 display: inline-block;
}

#menu ul li a {
   text-decoration: none;
   height: 28px;
   width:  80PX;
   display: block;
   border: solid 1px rgb(1, 67, 155);
   text-align: center;
   line-height: 25px;
   background: rgb(7, 93, 207);
   color: white;
}

#menu ul li a:hover {
  background: green;
  color: white;
}

#telicko {
   background: white;
   border-left: solid 1px rgb(160, 159, 159);
   border-right: solid 1px rgb(160, 159, 159);
}
```

```
      #paticka {
       height: 100px;
       background: grey;
       text-align: center;
       padding: 40px;
      }
  </style>
  </head>
  <body>
    <div id="main">
        <div id="hlavicka">
            <div id="menu">
                <ul>
                  <li>
                     <a href="#">odkaz 1</a>
                  </li>
                  <li>
                     <a href="#">odkaz 2</a>
                  </li>
                  <li>
                     <a href="#">odkaz 3</a>
                  </li>
                  <li>
                     <a href="#">odkaz 4</a>
                  </li>
                </ul>
            </div>
            <h1>du - pozicování</h1>
        </div>
        <div id="telicko">
<!--
   1. Vytvořte obsahovou část rozdělenou na tři části o šířce 30 %, 30 %
a 40 %
```

```
   2. Do první části vložte vhodný obrázek a text o vhodné délce
   3. Do druhé části vložte vhodný obrázek a bodový seznam
   4. Do poslední části vložte vhodný nadpis a text o vhodné délce
   Jednotlivé kroky vždy verzujte a uveďte v commitu vhodný komentář.
-->
        </div>
        <div id="paticka">
            copyright   &#169;
        </div>
    </div>
  </body>
</html>
```

Příloha 5: soubor index.html pro ukázkové řešení společné práce

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>společný projekt</title>
  <style type="text/css">
    *{
        margin: 0px;
        padding: 0px;
        position: relative;
        box-sizing: border-box;
      }

      #hlavicka, .telicko, #paticka {
        margin: 0px auto;
        width: 960px;

      }
      .telicko {
        min-height: 450px;
        padding-top: 136px;
      }

      #kontainer-1 {
        background: rgb(7, 74, 161);
        padding-top: 30px;
        z-index: 1;
        position: fixed;
        width: 100%;
      }

       #kontainer-2, #kontainer-3, #kontainer-4, #kontainer-5 {
```

```
  background: rgb(230, 230, 230);
}

#kontainer-6 {
  background: rgb(60, 60, 60);
  padding-bottom: 40px;
  z-index: 1;
  position: fixed;
  bottom: 0px;
  width: 100%;
}

#hlavicka {
  background: rgb(5, 113, 255);
  border: solid 1px rgb(205 201 201);
  border-bottom: none;
}

#hlavicka h1 {
   padding: 20px 0px;
   text-align: center;
 }

#menu {
  background: rgb(7, 93, 207);
}

#menu ul {
 display: flex;
}

#menu ul li{
 display: inline-block;
}
```

```
#menu ul li a {
   text-decoration: none;
   height: 28px;
   width:  80PX;
   display: block;
   border: solid 1px rgb(1, 67, 155);
   text-align: center;
   line-height: 25px;
   background: rgb(7, 93, 207);
   color: white;
}

#menu ul li a:hover {
  background: green;
  color: white;
}

#telicko {
   background: white;
   border-left: solid 1px rgb(160, 159, 159);
   border-right: solid 1px rgb(160, 159, 159);
}


#paticka {
 height: 100px;
 background: grey;
 text-align: center;
 padding: 40px;
 border: solid 1px black;
 border-top: none;
}
```

```
  </style>
  </head>
  <body>
    <div id="kontainer-1">
        <div id="hlavicka">
            <div id="menu">
                <ul>
                  <li>
                     <a href="#">odkaz 1</a>
                  </li>
                  <li>
                     <a href="#">odkaz 2</a>
                  </li>
                  <li>
                     <a href="#">odkaz 3</a>
                  </li>
                  <li>
                     <a href="#">odkaz 4</a>
                  </li>
                </ul>
            </div>
            <h1>Nadpis stránky</h1>
        </div>
    </div>
<!--        Společná tvorba jednostránkového webu.
```

Sekce 1:

1. Vytvořte obsah pro stránku „Úvod“ v sekci 1.
2. Obsahovou část rozdělte na tři části a do každé části vložte obrázek a text vztahující se k tématu.
3. Upravte odkaz v menu tak, aby odkazoval na id sekce-1 a zobrazoval popis „Úvod“.

Sekce 2:

1. Vytvořte obsah pro stránku „O nás“ v sekci 2.

2. Obsahovou část rozdělte na dvě části a do každé části vložte vhodný obsah (o nás).

3. Upravte odkaz v menu tak aby odkazoval na id sekce-2 a zobrazoval popis „O nás“.

Sekce 3:

1. Vytvořte obsah pro stránku „Portfolio“ v sekci 3.

2. Obsahovou část rozdělte na čtyři části a do každé části vložte vhodný obsah (portfolio).

3. Upravte odkaz v menu tak aby odkazoval na id sekce-3 a zobrazoval popis „Portfolio“.

Sekce 4:

1. Vytvořte obsah pro stránku „Kontakt“ v sekci 4.

2. Obsahovou část rozdělte na dvě části a do první části vložte vhodný obsah (kontakty). Do druhé části vložte Google mapku vhodné velikosti.

3. Upravte odkaz v menu tak aby odkazoval na id sekce-4 a zobrazoval popis „Kontakt“.

Jednotlivé kroky vždy verzujte a v commitu uveďte vhodný komentář.

-->

```
    <div id="kontainer-2">
        <div id="sekce-1" class="telicko">
          <!--  obsah sekce 1  -->

        </div>
    </div>
        <div id="kontainer-3">
        <div id="sekce-2" class="telicko">
          <!--  obsah sekce 2  -->

        </div>
    </div>
        <div id="kontainer-4">
        <div id="sekce-3" class="telicko">
```

```
            <!--  obsah sekce 3  -->

        </div>
    </div>
        <div id="kontainer-5">
        <div id="sekce-4" class="telicko">
            <!--  obsah sekce 4  -->

        </div>
    </div>
    <div id="kontainer-6">
        <div id="paticka">
            copyright   &#169;
        </div>
    </div>
  </body>
</html>
```