УДК 37.031.4

Гриценко Валерій Григорович
кандидат педагогічних наук, доцент, докторант
Інститут інформаційних технологій і засобів навчання НАПН України, м. Київ, Україна
grycenko@ukr.net

Подолян Оксана Миколаївна
кандидат фізико-математичних наук, старший викладач кафедри автоматизації та комп'ютерно-інтегрованих технологій
Черкаський національний університет імені Богдана Хмельницького, м. Черкаси, Україна
ompodolyan@mail.ru

ВИКОРИСТАННЯ СИСТЕМИ УПРАВЛІННЯ ВЕРСІЯМИ GIT ДЛЯ ОРГАНІЗАЦІЇ КОМАНДНОЇ РОБОТИ НАД ІТ ПРОЕКТОМ

Анотація. Досліджено засади використання систем управління версіями і здійснено пошук інструментарію щодо їх реалізації. З цією метою було проведено аналіз й обґрунтовано вибір програмного середовища Git. Визначено особливості використання системи управління версіями Git під час реалізації ІТ проектів. Описано процедуру й основні кроки створення робочої версії проекту. Досліджено, визначено й запропоновано технологію взаємодії команди розробників програмного проекту, що використовують систему управління версіями Git для організації спільної роботи. На прикладі реалізації авторського проекту інформаційно-аналітичної системи тестового контролю знань показано практичну сторону організації командної роботи розробників з використанням засобів системи управління версіями Git.

Ключові слова: Git; система управління версіями; репозиторій; ІТ проект; ІТ фахівці; команда розробників.

1. ВСТУП

Нинішній стрімкий розвиток інформаційних технологій потребує підготовки фахівців цієї сфери, здатних швидко реагувати й адаптуватися до всіх змін і вимог задля виконання різноманітних професійних завдань. Однією з найпоширеніших форм співпраці ІТ фахівців є командна робота над створенням проектів. Така форма організації роботи є досить ефективною й продуктивною, оскільки дозволяє розв'язувати складні і громіздкі завдання, які не зможуть виконати вчасно і якісно поодинці навіть високопрофесійні фахівці [1]. Тому набуття умінь здійснювати розробку певних частин проекту паралельно з іншими учасниками сприятиме отриманню досвіду майбутніми ІТ фахівцями щодо ефективності їх подальшої участі у розробці програмного забезпечення будь-якої складності.

Для організації ефективної командної роботи над ІТ проектом потрібні відповідні, зокрема, програмні засоби.

Аналіз останніх досліджень і публікацій. Різним аспектам теорії і технології командної роботи присвячені дослідження вітчизняних і зарубіжних учених, зокрема, основний понятійний апарат визначено у роботах Д. Катценбаха і Д. Сміта, психологічним проблемам управління командою присвячено праці О. Бандурки і Є. Ходаківського, вивченню методів командного підходу в управлінні присвячено роботи І. Мазура, В. Шапіро, М. Уолтон і П. Шолтерса, виявленню способів підвищення ефективності діяльності команд приділяли увагу Д. Грейсон і К. О'Делл, розробку ідеології навчання у команді започаткували Р. Славін, Р. Джонсон, Д. Джонсон та С. Каган.

©В.Г.Гриценко,О.М.Подолян, 2014.

Проте наявні дослідження не повністю враховують особливості організації індивідуально-групового навчання під час командної реалізації ІТ проектів у нинішніх швидкоплинних умовах. За цих обставин постійно продукується проблема пошуку технологічних рішень і розробки методичних прийомів щодо організації командної роботи і навчання, механізмів і закономірностей міжособистісної взаємодії майбутніх ІТ фахівців.

Метою нашої роботи є аналіз можливостей використання систем управління версіями під час розробки ІТ проектів; пошук та обґрунтування вибору інструментарію що забезпечує бездоганну взаємодію учасників команди ІТ розробників; дослідження технології використання запропонованого інструментарію у процесі командної реалізації ІТ проектів.

2. ПОСТАНОВКА ЗАДАЧІ

На нашу думку, одним із різновидів індивідульно-групової роботи студентів у команді може слугувати їх безпосередня участь у реалізації відносно великого проекту в розрізі виконання випускних робіт. Такий підхід дозволяє організувати індивідуальну роботу в команді, зокрема, у процесі розробки певної інформаційної системи (наприклад, у нашому розгляді — системи тестового контролю знань), яка виконується за всіма правилами командної роботи.

Зважаючи на те, що розробка певного програмного продукту може займати доволі тривалий проміжок часу, то саме командна робота дозволяє скоротити термін розробки проекту [3].

Донині було зроблено багато спроб створити спеціальне програмне забезпечення, яке б дало можливість прискорити і спростити розробку майбутніх програмних продуктів. І як показав їх аналіз, більшість сучасних рішень стосовно управління вихідним кодом програмних продуктів містять в собі системи управління версіями [4]. Вбачаємо за потрібне дослідити такого роду системи задля визначення можливостей їх використання для організації командної роботи над ІТ проектами.

3. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1. Засади використання систем управління версіями

Насамперед, з'ясуємо що таке *система управління версіями* (СУВ). *Система управління версіями* — це система, що зберігає зміни в одному або декількох файлах так, щоб потім, за потреби, можна було відновити відповідні старі версії.

Системи управління версіями дозволяють зберігати попередні версії файлів і завантажувати їх у разі потреби. Вони зберігають повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки. Місце зберігання даних файлів називають *репозиторієм*. У середині кожного репозиторію можуть бути створені паралельні лінії розробки — *гілки*.

Гілки, зазвичай, використовують для зберігання експериментальних, незавершених та повністю робочих версій проекту. Більшість СУВ дозволяють кожному з об'єктів присвоювати мітки (*теги*), за допомогою яких можна формувати нові гілки і репозиторії.

Використання СУВ є вкрай важливими для роботи над великими проектами, до виконання яких одночасно задіяна велика кількість розробників. Використання СУВ створює низку додаткових можливостей:

– створення різних варіантів одного документу;
– документування всіх змін (коли, ким було змінено/додано, хто який фрагмент змінив);
– реалізація функції контролю доступу користувачів до файлів, передбачена можливість його обмеження;
– створення документації проекту з поетапним записом змін залежно від версії;
– додавання пояснень до змін і їх документування.

Досить часто для управління версіями користувач копіює файли проекту в інший каталог (назва каталогу, зазвичай, відповідає поточній даті). Звісно, популярність такого підходу спричинена його простотою, але, на жаль, він найчастіше дає збої. Адже дуже легко забути, у якому каталозі знаходиться потрібна інформація, змінити не той файл або скопіювати і перезаписати файли не туди, куди потрібно. Задля розв'язання цієї проблеми розроблялися локальні СУВ з простою базою даних, у якій зберігаються всі зміни потрібних файлів (рис. 1).

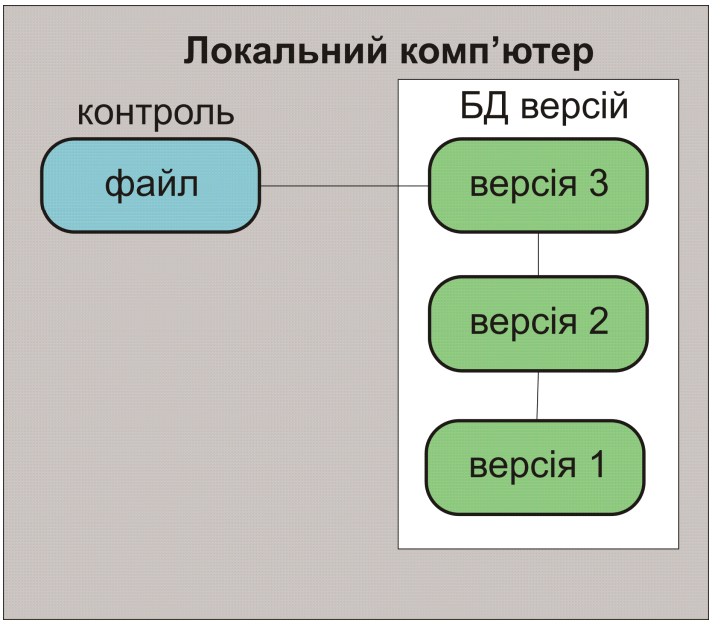

Рис. 1. Схема локальної СУВ

Однією з найбільш часто вживаних СУВ даного типу є RCS (Revision Control System), яка до цих пір широко використовується. Навіть у сучасній операційній системі Mac OS X утиліта RCS встановлюється разом з Developer Tools. Використання цієї утиліти полягає у роботі з наборами *патчів* між парами змін, які зберігаються у спеціальному форматі на диску. *Патч* — це файл, що описує відмінність між файлами попередньої і поточної версій. Така технологія дозволяє перестворити будь-який файл у будь-який момент часу, послідовно накладаючи патчі.

Іншою важливою проблемою виявилась потреба забезпечення співпраці розробників, що працюють віддалено за іншими комп'ютерами. Для її розв'язання були створені централізовані системи управління версіями (ЦСУВ). У таких системах, зокрема, CVS, Subversion і Perforce, є центральний *сервер*, на якому зберігаються всі файли, що відслідковуються, і група *клієнтів*, котрі отримують копії файлів з нього. Упродовж багатьох років такий підхід був стандартом управління версіями (рис. 2).

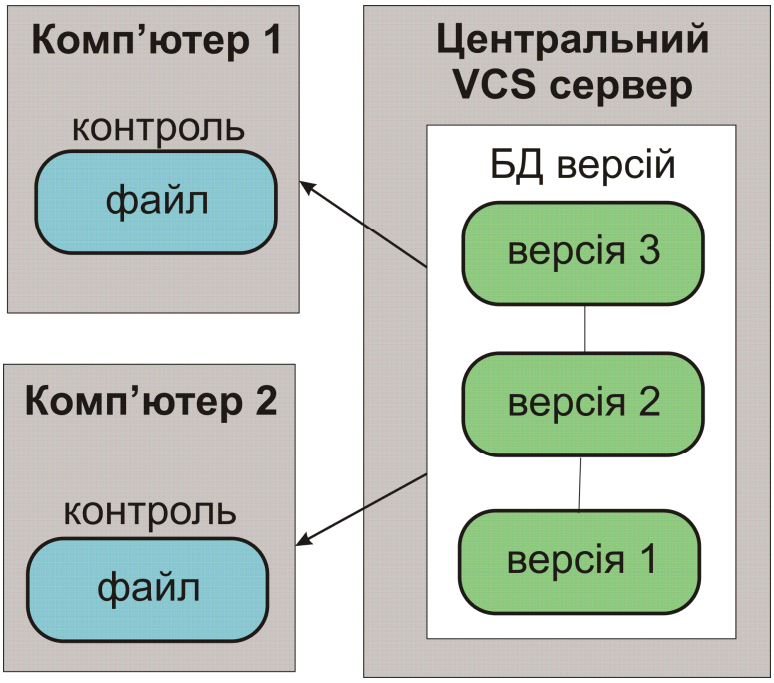

Рис. 2. Схема централізованого управління версіями

Зазначений підхід має низку переваг, особливо над локальними СУВ. Наприклад, коли між виконавцями проекту чітко розподілені завдання й обов'язки, то адміністратори легко можуть контролювати всіх учасників, усі процеси розробки проекту, і, попри це, адмініструвати ЦСУВ набагато простіше, ніж локальні бази на кожному клієнті. Однак, у разі використання такого підходу виникає низка суттєвих недоліків. Найбільш очевидний — централізований сервер є уразливою частиною всієї системи. Якщо сервер на певний час вимикається, то упродовж цього часу розробники не можуть взаємодіяти між собою, зберігати нові версії файлів проекту. Якщо ж пошкоджується диск із центральною базою даних і відсутні резервні копії, то втрачається абсолютно все — уся історія проекту, окрім, хіба що, декількох робочих версій, що збереглися на комп'ютерах розробників.

Зарадити проблемі може використання розподілених систем управління версіями (РСУВ) для командних розробок. У таких системах як Git, Mercurial, Bazaar або Darcs клієнти не просто отримують останні версії файлів, а повністю копіюють репозиторій. Тому у випадку, коли виходить з ладу сервер, через який організовувалась спільна робота над проектом, будь-який клієнтський репозиторій можна скопіювати знову на сервер, щоб відновити базу даних. Робота в РСУВ організована так, що кожного разу, коли клієнт отримує свіжу версію файлів, створюється повна копія всіх даних (рис. 3).

Попри це, у більшості такого типу систем передбачена можливість взаємодії з декількома віддаленими репозиторіями, завдяки цьому, можна одночасно у різних площинах працювати з різними групами розробників у межах одного проекту. Зокрема, в одному проекті можна одночасно організовувати декілька типів робочих процесів, що є неможливим у централізованих системах.

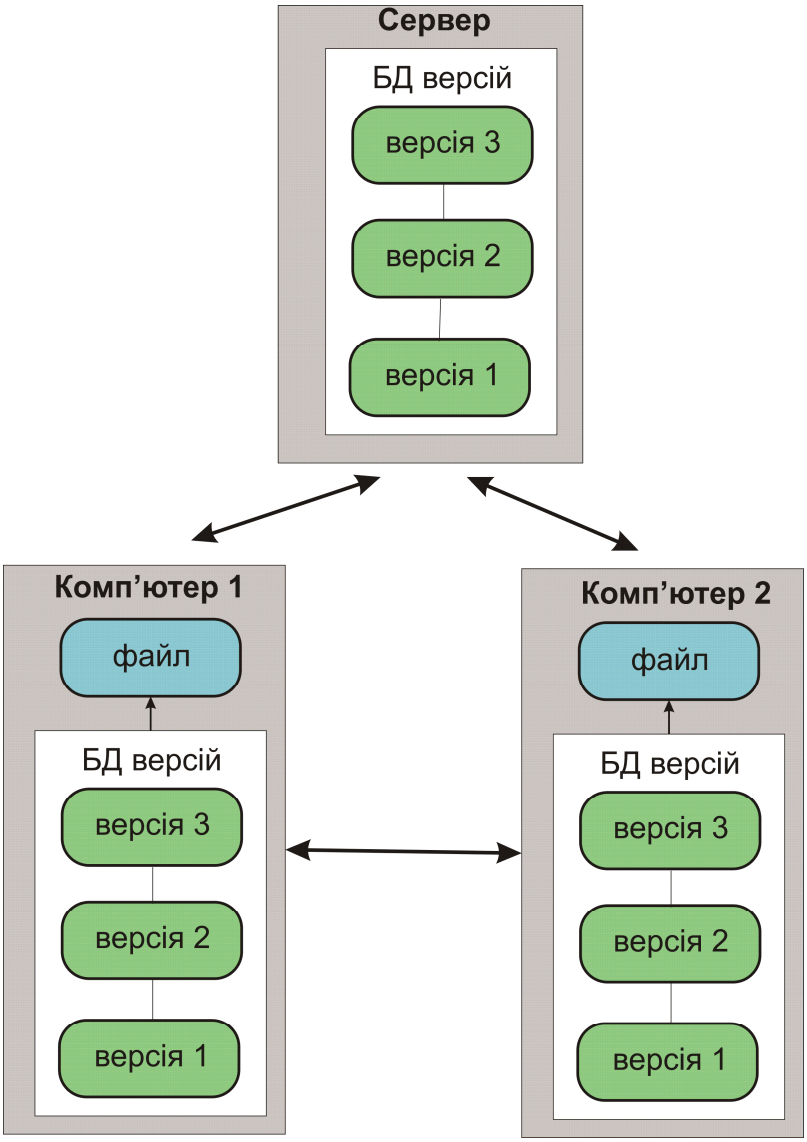

Рис. 3. Схема розподіленої системи управління версіями

Однією із СУВ, що визнана ІТ фахівцями доволі ефективною для створення великих проектів, а також має потужну систему розгалуження, є Git (розроблена Лінусом Торвальдсом у 2005 році) [5]. Відповідно до документації даного програмного продукту:

Git — це швидка, масштабована, розподілена система управління версіями з надзвичайно різноманітним набором команд, які забезпечують виконання основних операцій з репозиторієм, а також повний доступ до його внутрішньої структури.

До основних переваг Git порівняно із CVS можна віднести такі:

– розгалуження створюється швидко й легко;
– підтримується автономна робота, локальні фіксації змін можуть бути надіслані пізніше;
– фіксації змін атомарні і поширюються на весь проект, а не на окремий файл;
– кожне робоче дерево містить сховище з повною історією проекту;
– жодне зі сховищ за своєю суттю не є більш важливим, ніж будь-яке інше.

На сьогодні використання системи Git докорінно змінило підходи розробників щодо процесів розгалуження і злиття, які, до речі, є основними у роботі СУВ.

Основними елементами Git є репозиторій, гілки — головні і додаткові, а також основні команди й операції.

Репозиторій — це спеціальне сховище, у якому зберігаються всі файли разом з історією їх зміни та іншою службовою інформацією.

У системі Git репозиторій — це сховище, яке завжди знаходиться поряд з робочою директорією проекту в директорії *.git*.

У процесі реалізації проекту колективом розробників, передбачається існування головного спільного репозиторію, який, зазвичай, розміщується на сервері доступному для всіх його учасників.

Слід зазначити, що у системі Git, окрім головного, кожен учасник проекту може мати власний репозиторій. Кожен користувач звертається до центру, але окрім двосторонньої взаємодії з центром, кожен розробник може брати до уваги зміни на інших вузлах, так утворюючи субкоманду. Зокрема, застосування такого підходу може бути доречним під час роботи кількох розробників над реалізацією певної великої функції, перш ніж передати цю функцію до головного репозиторію. Відповідно до рис. 4 утворено субкоманди з таких учасників, як Розробник 1 і Розробник 4, Розробник 1 і Розробник 2, Розробник 3 і Розробник 2. З технічної точки зору це не що інше, як використання Розробником 1 віддалено репозиторію Розробника 4, і навпаки [9].

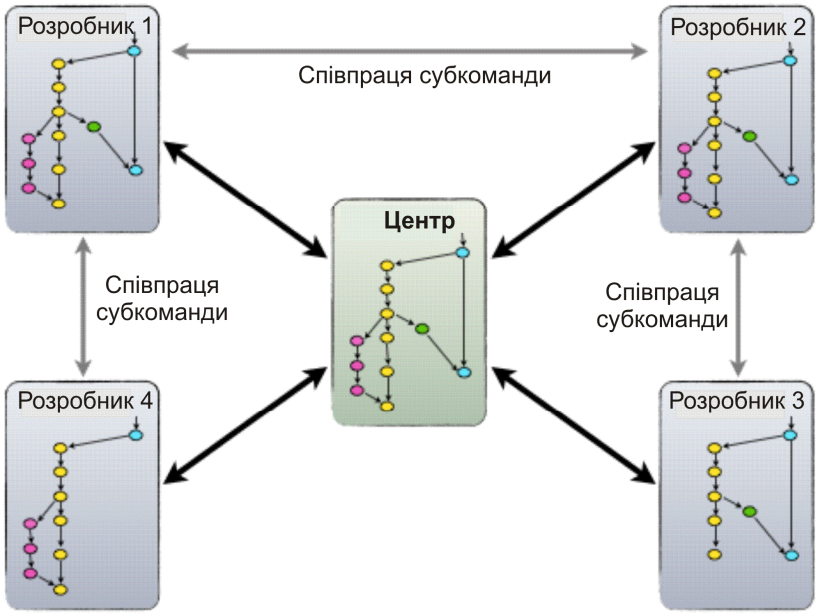

Рис. 4. Модель співпраці учасників субкоманди

3.2. Особливості використання системи управління версіями Git

Для створення репозиторію Git можна використовувати один із двох існуючих підходів [7]. Перший підхід полягає в імпорті у Git вже існуючого проекту чи каталогу. Другий реалізується шляхом клонування вже існуючого репозиторію із сервера.

Для початку використання Git з уже існуючим проектом, слід перейти до проектного каталогу й у командному рядку ввести команду:

```
$ git init
```

Ця команда створить у поточному каталозі новий підкаталог з ім'ям .git, що міститиме всі необхідні файли репозиторію — основу репозиторію Git.

Для внесення до репозиторію вже існуючих файлів слід їх спочатку проіндексувати, а потім виконати першу фіксацію змін. Цей процес забезпечить послідовне виконання кількох команд git add, що вказують на файли, які повинні бути проіндексовані, а потім на завершення зафіксованими командою commit:

```
$ git add *.c
$ git add README
$ git commit -m 'initial project version'
```

Комміт (commit) —— це збереження у репозиторії змін у програмному коді.

У разі виникнення потреби отримання копії вже існуючого репозиторію Git, на приклад проекту, до виконання якого слід долучити ще одного учасника, слід скористатися командою git clone. У процесі виконання цієї команди кожна версія кожного файлу з історії проекту забирається (pulled) із сервера. Наприклад, якщо потрібно зробити клон бібліотеки Ruby Git, відомої як Grit, то виконати це можна так:

```
$ git clone git://github.com/schacon/grit.git
```

Ця команда дає вказівку системі Git створити каталог з іменем «grit», ініціалізувати в ньому каталог .git, завантажити всі потрібні дані для цього репозиторію і створити (checks out) робочу копію останньої версії.

Як і будь-яка СУВ, Git надає можливість створювати розгалуження. Розгалуження означає, що можливе відхилення від основної лінії розробки і продовження роботи без втручання до основної лінії [9]. *Гілка* у системі Git — це звичайний рухомий вказівник на один зі створених коммітів.

Існує кілька підходів до використання системи Git, серед яких, найвдалішою, на наш погляд, є так звана концепція gitflow. Саме тому у подальшому нашому розгляді будемо дотримуватись цієї концепції.

Центральний репозиторій працює з двома паралельними між собою гілками: master і develop (рис. 5). Кожна гілка розглядається як основна, а вихідний код продукту буде відображати стан з внесеними змінами, що буде основою для подальших змін.

Коли вихідний код на робочій гілці (develop) досягає стабільної версії і приймається рішення про готовність його до використання, усі внесені зміни передаються головній гілці (master) і позначаються номером версії. Отже, кожного разу, коли зміни записуються у головну гілку, можна говорити про створення фактично нової, стабільної версії проекту. Слід зазначити, що потрібно доволі виважено ставитися до виконання цих дій, оскільки використання Git-сценарію для автоматичного створення й оновлення версії програмного забезпечення буде відбуватися кожного разу, коли відбуватиметься звернення до головної гілки.

Поряд з основними гілками — master і develop, можливе (рекомендоване для gitflow) існування також додаткових гілок, які забезпечуватимуть паралельну розробку проекту декількома членами команди, полегшуватимуть відслідковування певних негараздів, здійснюватимуть локальну підготовку до розробки версій, а також сприятимуть швидкому усуненню поточних проблем, які виникатимуть у процесі розробки проекту. На відміну від головних, такі гілки завжди матимуть визначений часовий термін існування і після завершення своєї актуальності будуть видалені.

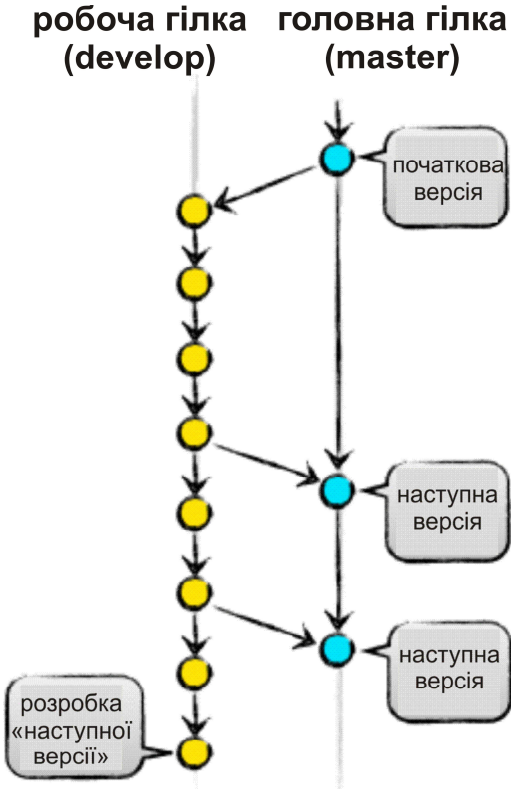

Рис. 5. Схема взаємодії робочої і головної гілок проекту

У процесі розробки складного проекту можуть використовуватися різні види додаткових гілок, зокрема: тематичні гілки (feature), гілки версій (release) та гілки помилок (hotfix). Кожна з цих гілок має спеціальне призначення і розробники проектів обов'язково мають притримуватись певного набору правил щодо їх визначення, зокрема, яка з гілок є першоджерелом, а яка — результатом злиття інших, тощо. З технічної точки зору це звичайні Git гілки, а їх класифікації буде визначатися метою їх використання.

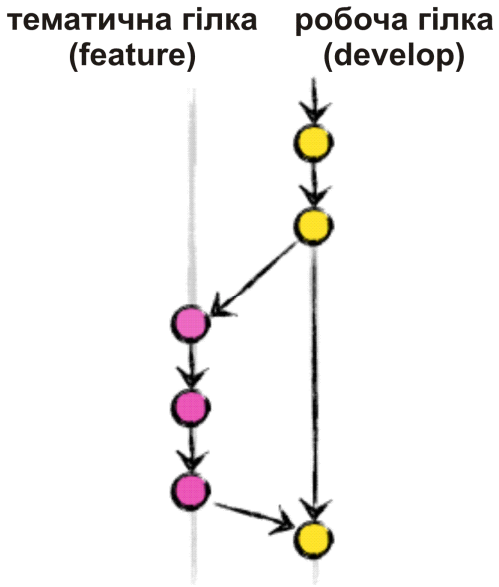

Рис. 6. Схема утворення тематичної гілки

Тематичні гілки є відгалуженням від робочої гілки (develop) і після завершення розробки визначеного функціоналу на них повинні бути злиті саме з робочою гілкою. Такі гілки можна називати будь-якими іменами, окрім master, develop, release-*, or hotfix-*.

Тематичні гілки використовуються для розробки нових, специфічних функцій майбутньої версії проекту. Основною особливістю даних гілок є те, що вони існують доки триває розробка певної функції проекту, а потім відбудеться їх злиття з гілкою develop (у випадку додавання нової функції до нової версії проекту) або і взагалі будуть знищені (у випадку невдалого експерименту їх застосування). Тематичні гілки існують лише в репозиторію гілки develop.

Коли починається робота над новою функцією проекту, то відбувається відвітлення від гілки develop. Процес створення тематичної гілки відбувається так:

```
$ git checkout -b myfeature develop
```

Отже, відбулося створення і перехід на нову гілку "myfeature". Після завершення роботи на цій гілці вона зіллється з гілкою develop:

```
$ git checkout develop // з'єднання з гілкою 'develop'
$ git merge --no-ff myfeature // злиття змін гілки myfeature з поточною develop
$ git branch -d myfeature // видалення гілки  myfeature
$ git push origin develop // синхронізація даних з віддаленим репозиторієм
```

Використання параметра `--no-ff` забороняє виконання операції Fast Forward, тобто простого переміщення вказівника гілки по дереву, навіть якщо це можливо [8]. Це дозволяє уникнути втрат інформації про існування тематичних гілок і згрупувати разом усі компоненти, з яких складатиметься новоутворена функція проекту. Порівняємо два способи процесу злиття гілок.

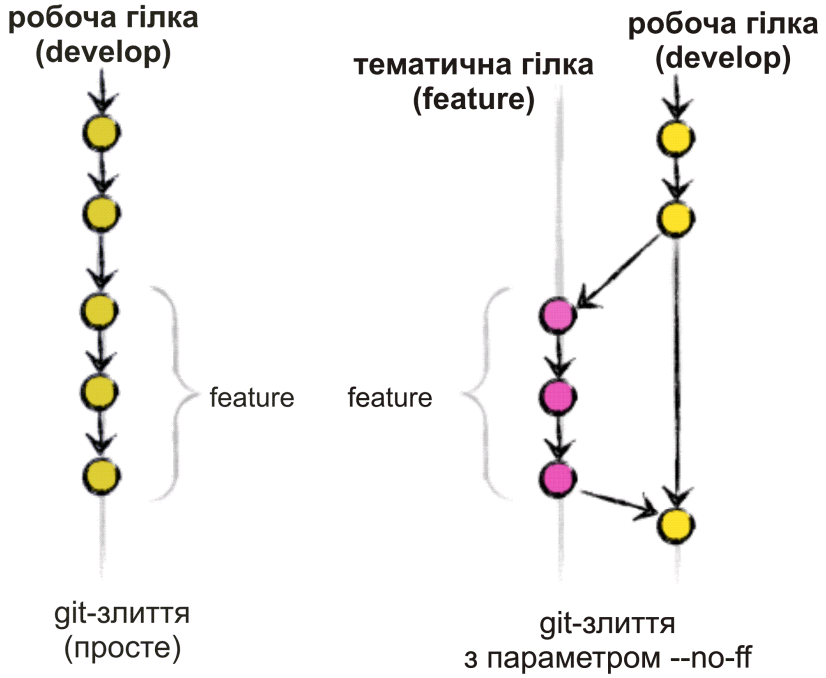

Рис. 7. Схема злиття гілок

У першому випадку (ліва частина рис. 7) з історії проекту неможливо виокремити ті об’єкти, які утворювали нову функцію, а, отже, у разі потреби її вдосконалення, слід буде ретельно переглядати весь журнал повідомлень. Утім, процес повернення до всієї тематичної гілки (тобто до групи фіксацій) буде дуже складним. Уникнути зазначених проблем можна за умови використання параметра `--no-ff` (права частина рис. 7). Звісно, такий підхід призведе до створення декількох додаткових коммітів (містять інформацію про злиття, а в разі виникнення конфліктів злиття міститимуть також зміни файлів, які призвели до них), але це не завадить проекту.

Гілки версій призначені для підготовки нової версії проекту. Вони дозволяють за мить до завершення роботи над поточною версією визначати місця підключення до головної гілки, а також надають можливість виконувати незначні правки й готувати дані про версію (номер, дата створення тощо). Виконання такого виду операцій на гілці версії дозволяє значно спростити розгалуження головної гілки develop. Відгалуження гілки версії від робочої гілки develop відбуватиметься за умови отримання бажаного стану нової версії проекту на робочій гілці.

Гілки помилок функціонально дуже схожі на гілки версій, оскільки також використовуються для підготовки нової версії проекту, але створюються вони спонтанно. Утворюються гілки помилок у разі потреби виконання миттєвих дій задля усунення небажаних станів робочої версії проекту. Тобто, якщо потрібно виправити критичну помилку у робочій версії проекту, то створюється відгалуження гілки помилок від головної гілки master, на тому її етапі, що відповідає робочій версії проекту в цей момент часу. Суттю використання таких гілок є те, що у разі виникнення непередбачуваних помилок робота членів команди на головній гілці develop не буде зупинена, а помилки будуть швидко знайдені і виправлені.

3.3. Технологія командної роботи над проектом системи тестового контролю знань

Вище розглянуті особливості використання СУВ Git визначили достатню функціональність цього середовища і переконують нас у можливості його успішного використання як інструменту командної роботи над проектом. Як приклад розглянемо розробку інформаційно-аналітичної системи тестового контролю знань у рамках Git-розгалуження. Беззаперечно, що розробка будь-якого проекту повинна бути чітко спланована, розділена на етапи й основні блоки, з яких складатиметься майбутня система тестування. До таких основних блоків віднесемо такі модулі: управління користувачами, банк тестів, обробка результатів тестування. Кожен із модулів можна доповнювати, розширювати, додаючи нові можливості і функції, але при цьому потрібно ретельно і виважено здійснювати створення нової версії програмного продукту, без втрат інформації на попередніх кроках розробки.

Важливим етапом командної роботи є обрання так званого лідера команди (team leader), тобто людини, яка керуватиме повністю процесом розробки проекту, контролюватиме процеси розгалуження і злиття гілок тощо. Тому бажано, щоб таким лідером був не студент, а викладач, тобто керівник дипломних робіт або адміністратор, який координує роботу всіх проектів кафедри. Саме лідер команди (адміністратор) починає роботу над проектом зі створення початкового комміту — створюється гілка master, і початкових налаштувань системи — створюється гілка develop. Далі, визначавшись з основними модулями, які необхідно розробити, завдання видається студентам-дипломникам, кожен з яких працює окремо, створюючи свою тематичну (feature) гілку.

Першим було розроблено блок керування користувачами, що включав у себе такі етапи: створення списку користувачів, створення структури бази даних, створення системи управління користувачами. Усе це було розроблено на тематичній гілці — feature/users_manage з подальшим її злиттям з гілкою develop.

Наступним кроком була розробка блоків, що дозволяють створювати сховища завдань, редактор завдань та каталог завдань. Ці модулі було розроблено на тематичній гілці — feature/task_catalogue, яка також була злита з гілкою develop.

Далі, модулі класифікації тестів, перегляду результатів тестування і власне самого тестування розроблялися на тематичній гілці — feature/test_processing, що також була злита з основною гілкою develop.

Об’єднання вище описаних трьох гілок дає змогу підготувати першу, так звану незавершену, версію продукту — release/0.1. Виправлення всіх помилок злиття на гілці версій (release) дозволяє отримати першу стабільну версію розробленої системи тестування на гілці master — версія 0.1.

У ході першого використання розробленої системи тестування виникли нові завдання, які значно покращать функціональність даного програмного продукту. У результаті чого було додано ще декілька модулів, зокрема модуль імпорту тестів, що розроблявся на гілці feature/test_importer, модуль редагування тестів, що розроблявся на гілці feature/advanced_editor, модуль математичного редактора, що розроблявся на гілці feature/math_editor. Об’єднання цих трьох тематичних гілок утворить нову версію програмного продукту — release/0.2. Виправлення помилок злиття на гілці версій (release) дозволяє отримати другу стабільну версію розробленої системи тестування на гілці master — версія 0.2.

Подальше вдосконалення системи, виправлення виникаючих помилок злиття на гілці версій дає змогу отримати наступні версії програмного продукту — версія 0.2.1.

Отже, розробка окремих модулів, з яких складається система тестування відбувається на окремих тематичних гілках, окремими студентами в рамках виконання їхніх дипломних робіт. Це дозволяє виконувати роботу незалежно від інших учасників проекту, значно спрощує процес злиття розроблених модулів в єдине ціле, оскільки цим займається лише адміністратор. Що ж до структурної схеми проекту, то саме використання тематичних гілок дозволяє значно спростити гілку develop, а на головній гілці master розміщувати лише стабільні версії програмного продукту. Загальну схему розробки системи тестування з використання Git-розгалуження представлено на рис. 8.

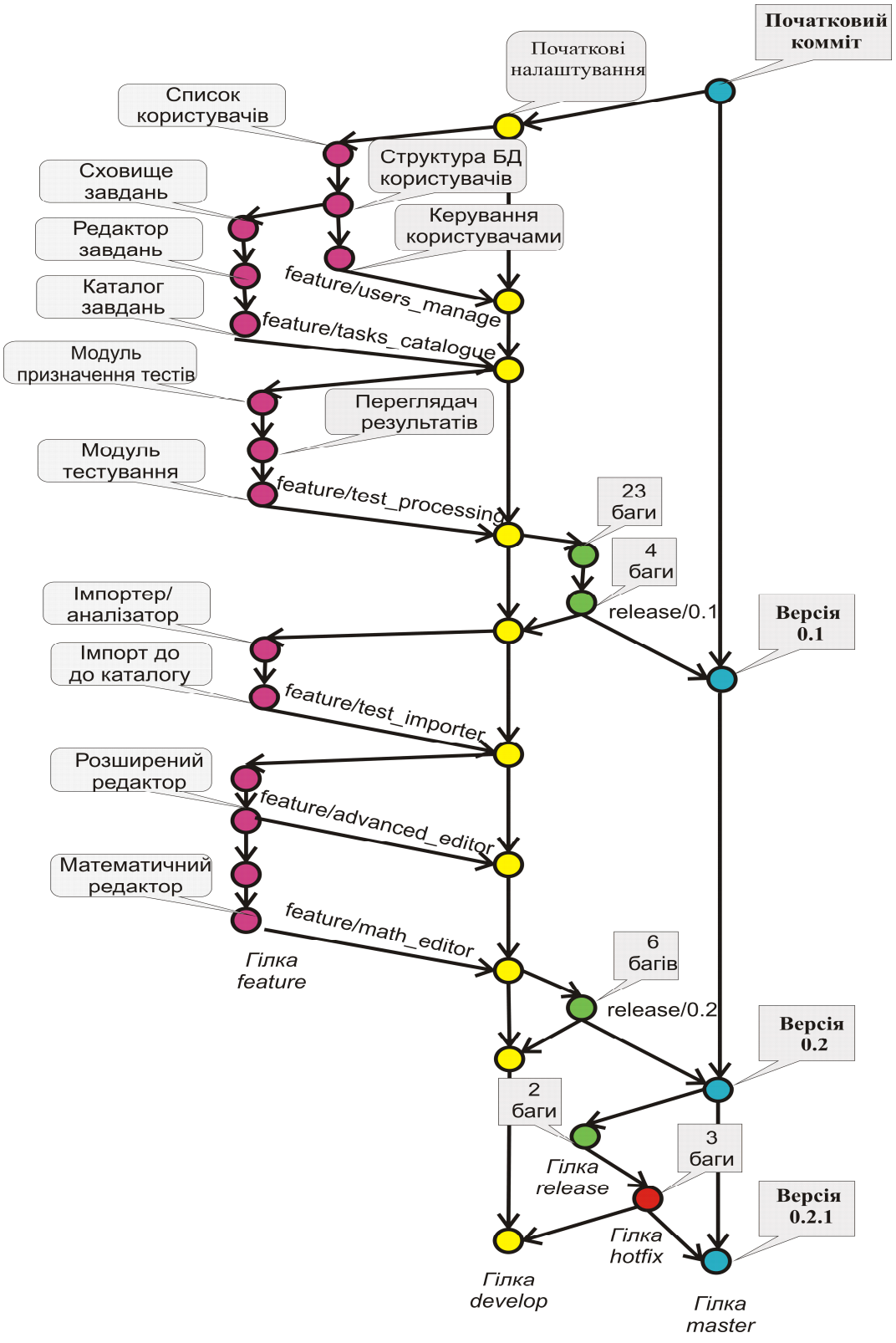

Рис. 8. Загальна схема розробки проекту системи тестового контролю

4. ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Підсумовуючи вище сказане, можна стверджувати, що проста й водночас ефективна система віддаленого управління версіями Git дозволяє організувати й

забезпечити виконання дипломних робіт як складових частин більш складного програмного проекту. Навички командної роботи, отримані студентами-дипломниками в процесі виконання подібних проектів, дозволять їм у майбутній професійній діяльності легко адаптуватися до роботи в колективі. Адже уміння працювати в команді є необхідними для випускника сучасного ВНЗ незалежно від отриманого фаху, але особливо важливі для менеджерів і ІТ фахівців, оскільки сфера їх професійної діяльності вимагає об'єднання в команди для генерації нових ідей, створення нових проектів і технологій, а також продукування ефективних рішень.

Перспективи подальших досліджень вбачаємо у визначенні особливостей використання системи управління версіями Git як засобу конфігураційного управління, що забезпечуватиме супровід програмного продукту упродовж усього його життєвого циклу, зокрема, створення резервних копій програмного коду і даних, контроль програмного коду, опису проекту і супровідної документації тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Скібіцька Л. І. Організація праці менеджера / Л. І. Скібіцька. — К., 2010. — 346 с.
2. Горбовий А. Ю. Упровадження командної форми роботи в систему управління організацією / А. Ю. Горбовий, О. М. Степанюк // Науковий вісник Волинського національного університету імені Лесі Українки, 2009. — С. 54–59.
3. Карамушка Л. М. Особливості розвитку психологічної готовності аспірантів технічних університетів до роботи в команді / [Л. М. Карамушка, О. А. Філь, П. П. Блешмудт, І. Г. Васильєва] // Теорія і практика управління соціальними системами. — 2010. — № 3. —С. 74–83.
4. Киричек Г. Г. Модель оцінки плагіату програмного коду на основі системи контролю версій / Г. Г. Киричек, О. О. Киричек // Восточно-Европейский журнал передовых технологий. — 2012. — № 2/2. — Вып. 56. — С. 25–28.
5. Robbins Jason. Analysis of Git and Mercurial [Заглавие с экрана] / Jason Robbins. — Режим доступу : http://code.google.com/p/support/wiki/DVCSAnalysis.
6. Scott Chacon. Pro Git // Apress. — 2013. — 294 с.
7. Моя шпаргалка по работе с Git. Записки программиста. 2011. [Електронний ресурс]. — Режим доступу : http://eax.me/git-commands/.
8. Эли М Доу. Управление исходным кодом с помощью Git. 2009. [Електронний ресурс]. — Режим доступу : http://www.ibm.com/developerworks/ru/library/l-git/.
9. Driessen V. A successful Git branching model. 2010 [Електронний ресурс]. — Режим доступу : http://nvie.com/posts/a-successful-git-branching-model/.

Матеріал надійшов до редакції 11.12.2013 р.

ИСПОЛЬЗОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ GIT ДЛЯ ОРГАНИЗАЦИИ КОМАНДНОЙ РАБОТЫ НАД ИТ ПРОЕКТОМ

Гриценко Валерий Григорьевич
кандидат педагогических наук, доцент, докторант
Институт информационных технологий и средств обучения НАПН Украины, г. Киев, Украина
grycenko@ukr.net

Подолян Оксана Николаевна
кандидат физико-математических наук, старший преподаватель кафедры автоматизации и компьютерно-интегрированных технологий
Черкасский национальный университет имени Богдана Хмельницкого, г. Черкассы, Украина
ompodolyan@mail.ru

Аннотация. Исследованы основы использования систем управления версиями и осуществлен поиск инструментария для их реализации. С этой целью был проведен анализ

и обоснован выбор программной среды Git. Определены особенности использования системы управления версиями Git при реализации ИТ проектов. Описана процедура и основные шаги создания рабочей версии проекта. Исследована, определена и предложена технология взаимодействия команды разработчиков программного проекта, с использованием системы управления версиями Git для организации совместной работы. На примере реализации авторского проекта информационно-аналитической системы тестового контроля знаний показано практическую сторону организации командной работы разработчиков с использованием средств системы управления версиями Git.

Ключевые слова: Git; система управления версиями; репозиторий; ИТ проект; ИТ специалисты; команда разработчиков.

APPLICATION OF GIT-BRANCHING FOR THE ORGANIZATION OF TEAMWORK ON IT PROJECTS

Valerii G. Grytsenko
PhD (pedagogical sciences), associate professor, doctoral
Institute of Information Technologies and Learning Tools, NAPS of Ukraine, Kyiv, Ukraine
grycenko@ukr.net

Oksana M. Podolyan
PhD (Physics and Mathematics), senior lecturer of the Department of Automation and Computer-Integrated Technologies
Cherkasy Bohdan Khmelnytsky National University, Cherkasy, Ukraine
ompodolyan@mail.ru

Abstract. In the article the basics of version control systems using are investigated and searched the tools for their implementation. For this purpose, it was made the analysis and justified the choice of programming environment Git. The features of Git version control system using in the implementation of IT projects are identified, as well as described the procedure and the basic steps to create a working version of the project. The technology of interoperability team software project using Git version control system for collaboration is investigated, identified and proposed. By the example of the implementation of the author's project of knowledge testing information-analytical system it was shown the practical side of organizing teamwork with the use of Git version control system.

Keywords: Git; version control system; IT project; IT professionals; the development team.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Skibitska L. I. The organization works manager. — K., 2010. — 346 p. (in Ukrainian)
2. Gorbovyy A. Y., Stepanjuk A. Implementation team forms of work organization in management // Scientific Bulletin of Volyn National University of Lesya Ukrainian, 2009. — P. 54–59 (in Ukrainian)
3. Karamushka L. M., Fil A., Bleshmudt P., Vasilyeva I. G. Features of psychological readiness of graduate students of technical universities to work in a team // Theory and practice of social systems. — 2010. — № 3. — P. 74–83 (in Ukrainian)
4. Kyrychek G. G., Kyrychek A. A.. Model evaluation plagiarism software code from version control systems // Journal of East- Evropeyskyy peredovыh technology. — 2012. — № 2/2. — Vol. 56. — P. 25–28 (in Ukrainian)
5. Jason Robbins. Analysis of Git and Mercurial [Online] / Jason Robbins. — Available from : http://code.google.com/p/support/wiki/DVCSAnalysis (in English)
6. Chacon Scott. Pro Git // Apress. — 2013. — 294 p.(in English)
7. My crib to work with Git. Notes of programmer, 2011. [Online]. — Available from : http://eax.me/git-commands/ (in Russian)
8. Ely M. Doe. Manage source code using Git, 2009. [Online]. — Available from : http://www.ibm.com/developerworks/ru/library/l-git/(in Russian)
9. Driessen V. Successful Git branching model. 2010 [Online]. — Available from : http://nvie.com/posts/a-successful-git-branching-model/(in English)