## 12.2. Системи контролю версій

За останні кілька десятиліть системи контролю версій VCS (Version Control Systems) стали набагато досконалішими, причому деяким це вдалося краще за інших. Системи VCS іноді називають інструментами SCM (керування вихідним кодом) або RCS (системою керування редакціями). Один з найпопулярніших на сьогоднішній день інструментів VCS називається Git. Git відноситься до категорії розподілених систем контролю версій, відомих як DVCS (ця тема буде розглянута докладніше пізніше). Git, як і багато інших популярних і доступних на сьогоднішній день системи VCS, поширюється безкоштовно та має відкритий вихідний код. Незалежно від того, яку систему контролю версій ви використовуєте та як вона називається, основні її переваги полягають у наступному.

1. Повна історія змін кожного файлу протягом тривалого періоду. Це стосується всіх змін, внесених величезною кількістю людей протягом багатьох років. Зміною вважається створення та видалення файлів, а також редагування їхнього вмісту. Різні інструменти VCS відрізняються тим, наскільки добре вони обробляють операції перейменування та переміщення файлів. В історію також повинні входити відомості про автора, дата та коментар з описом мети кожної зміни. Наявність повної історії дозволяє повертатись до попередніх версій, щоб проводити аналіз основних причин виникнення помилок та усувати проблеми у старих версіях програмного забезпечення. Якщо над програмним забезпеченням ведеться активна робота, то «старою версією» вважатимуться майже весь код цього програмного забезпечення.
2. Розгалуження та злиття. Ця можливість корисна не лише за одночасної роботи учасників команди: окремі люди також можуть отримати з неї користь та працювати над кількома незалежними напрямками. Створення «гілок» в інструментах VCS дозволяє мати кілька незалежних напрямків розробки, а також виконувати їх злиття, щоб розробники могли перевірити, що зміни, внесені в кожну з гілок, не конфліктують один з одним. Багато команд розробників програмного забезпечення створюють окремі гілки для кожної функціональної можливості, для кожного релізу або для того, і для іншого. Наявність багатьох різних робочих процесів дозволяє командам вибирати підходящий їм спосіб використання розгалуження і злиття в VCS.
3. Відстежуваність. Можливість відслідковувати кожну зміну, внесену до програмного забезпечення, і пов'язувати її з ПЗ для керування проектами та відстеження помилок, наприклад Jira , а також залишати до кожної зміни коментар з описом мети та призначення зміни може допомогти не тільки при аналізі основних причин виникнення помилок, але та при проведенні іншого аналізу. Історія з коментарями під час читання коду допомагає зрозуміти, що цей код робить і чому дія реалізована саме таким чином. Завдяки цьому розробники можуть вносити коректні та сумісні зміни відповідно до довгострокового плану розробки системи. Це особливо важливо для ефективної роботи з успадкованим кодом, оскільки дає розробникам можливість більш точно оцінити обсяг подальшої роботи.

Програмне забезпечення для керування версіями є невід'ємною частиною повсякденної професійної практики сучасної команди розробників програмного забезпечення. Окремі розробники програмного забезпечення, які звикли працювати в команді з ефективною системою управління версіями, зазвичай визнають неймовірну користь управління версіями навіть при роботі над невеликими сольними проектами. Звикли до потужних переваг систем

контролю версій, багато розробників не уявляють як працювати без них навіть у проектах, не пов'язаних з розробкою ПЗ. Таким чином, питання полягає не в тому, чи використовувати керування версіями, а в тому, яку систему керування версіями вибрати.

Серед багатьох існуючих систем управління версіями самою популярною є система Git.

## *Контроль версій за допомогою Git*

Git - абсолютний лідер за популярністю серед сучасних систем управління версіями. Це розвинений проект з активною підтримкою та відкритим вихідним кодом. Git застосовується для управління версіями в рамках колосальної кількості проектів з розробки програмного забезпечення, як комерційних, так і з відкритим вихідним кодом. Система використовується багатьма професійними розробниками програмного забезпечення. Вона працює під управлінням різних операційних систем і може застосовуватися з багатьма вбудованих середовищ розробки (IDE).

Розробка в Git орієнтована на забезпечення високої продуктивності, безпеки та гнучкості розподіленої системи.

**Продуктивність**

Git показує дуже високу продуктивність у порівнянні з іншими альтернативами. Це можливо завдяки оптимізації процедур фіксації коммітів, створення гілок, злиття та порівняння попередніх версій. Алгоритми Git розроблені з врахуванням глибокого знання атрибутів, притаманних реальних дерев файлів вихідного коду, і навіть типової динаміки змін та послідовностей доступу.

Деякі системи керування версіями керуються іменами файлів під час роботи з деревом файлів та ведення історії версій. Замість обробки назв система Git аналізує вміст. Це важливо, оскільки файли вихідного коду часто перейменовують, поділяють та змінюють місцями. Об'єктні файли репозиторію Git формуються за допомогою дельта кодування (фіксації відмінностей вмісту) та компресії. Крім того, такі файли в чистому вигляді зберігають об'єкти з вмістом каталогу та метаданими версіями.

**Безпека**

При розробці Git насамперед забезпечується цілісність вихідного коду під управлінням системи. Вміст файлів, а також об'єкти репозиторію, що фіксують взаємозв'язки між файлами, каталогами, версіями, тегами та коммітами, захищені за допомогою криптографічно стійкого алгоритму хешування SHA1. Він захищає код та історію змін від випадкових та зловмисних модифікацій, а також дозволяє простежити історію у повному обсязі. Використання Git гарантує справжність історії змін коду.

**Гнучкість**

Гнучкість – одна з основних характеристик Git. Вона проявляється у підтримці різних нелінійних циклів розробки, ефективності використання з малими та великими проектами, а також сумісності з багатьма системами та протоколами.

Система Git розрахована насамперед створення гілок і використання тегів. Тому процедури за участю гілок та тегів (наприклад, об'єднання та повернення до попередньої версії) зберігаються в історії змін. Не всі системи управління версіями мають такі широкі можливості відстеження.

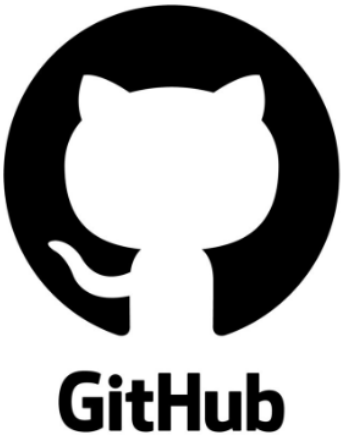
**Відкритий код**

Проект Git має відкритий вихідний код, а також активно підтримується та безперервно розвивається вже понад 10 років. Куратори проекту продемонстрували зважений та продуманий підхід до виконання вимог користувачів, регулярно випускаючи релізи для підвищення зручності та розширення функціональних можливостей системи. Якість ПЗ з відкритим вихідним кодом легко перевіряється, і багато організацій повністю довіряють таким продуктам.

Навколо Git сформувалася численна спільнота користувачів, а сам проект отримує активну підтримку з боку спільноти. Система має докладну та якісну документацію: всім бажаючим серед іншого доступні книги, навчальні посібники, спеціалізовані веб сайти, подкасти та навчальні відеоролики.

Git - це система з відкритим вихідним кодом, тому розробники любителі можуть користуватися нею абсолютно безкоштовно.

## *GitHub і його відмінність від Git*




GitHub – це хмарна платформа для хостингу IT-проектів та спільної розробки, під капотом якої знаходиться популярна система контролю версій Git, а також повноцінна соціальна мережа для розробників.

Тут можна знайти багато open-source-проектів різними мовами та взяти участь у них, розмістити своє портфоліо з прикладами коду, щоб додати посилання до резюме, підглядати у відкритих проектах цікаві архітектурні рішення, дивитися, як досвідчені розробники пишуть код, та завантажувати величезну кількість корисних у розробці та безкоштовних інструментів для розробки.

Прийшовши практично до будь-якої IT-компанії, ви зіткнетеся з тим, що код в переважній більшості випадків зберігається на GitHub. У GitHub є досить відомий конкурент - GitLab, він теж заснований на Git, але це різні платформи різних компаній, хоча їхня функціональність дуже схожа.

Не варто плутати GitHub та Git. GitHub — лише одна з реалізацій системи контролю версій Git, до якої додано багато зручних інструментів та можливостей (ті ж коментарі, issues, гіперпосилання, форматований текст тощо). Пам'ятайте, GitHub можна використовувати і без знання Git (зворотне теж правильно).

**Для яких цілей використовують GitHub**

Безумовно, GitHub потрібний не всім. В першу чергу GitHub необхідний проектам із частими оновленнями, багатьма версіями, великою кількістю файлів, необхідністю синхронізації розробки та зручного розгортання.

Є багато інших способів зберігання вихідних джерел: можна створити для них папку в розділі «Мої документи», закидати їх у хмару та підписувати версії або навіть завантажувати в «Вибране» в Telegram. Всі ці способи по-своєму хороші, але для роботи в IT потрібно звикати до GitHub: це стандарт індустрії.

**Основні концепції GitHub**

З моменту створення у GitHub з'явилося багато інструментів, але головне залишається незмінним. У Git успадковується майже вся термінологія. Основні терміни – репозиторій, гілка, комміт, форк. Вибір деяких із цих назв може здатися не дуже інтуїтивним (навіть якщо ви володієте англійською), але так вже склалося.

**Репозиторій**

Це просто коренева папка з файлами та вкладеними директоріями програми – і водночас її сторінка на GitHub. Завантажити в репозиторій можна все, що завгодно, але передбачається, що в ньому зберігаються файли з вихідним кодом і додаткові матеріали — наприклад, необхідну для GUI або верстки графіку (картинки, іконки тощо).

Репозиторії можуть бути публічними та приватними, в них можна створювати інші папки та відстежувати зміни версій. Керувати своїми репозиторіями можна прямо через інтерфейс сайту, командний рядок, десктопну програму GitHub або різні засоби розробки (IDE), що підтримують інтеграцію з сервісом.

**Гілка (branch)**

У гілки групуються зміни та оновлення - припустимо, одна головна гілка (за замовчуванням створюється main) і одна beta. Гілки незалежні одна від одної, але за бажанням їх можна об'єднувати (merge - злиття) - навіть якщо між ними є різниця в коді.

**Способи зміни репозиторію: коміт, пуш, клон, форк**

Внести зміни репозиторію можна безпосередньо або створивши копію. Саме внесення змін називається «комміт» (від англійської commit — зробити), він має тимчасову мітку і хеш-суму.

Перенесення змін-коммітів з локального репозиторію (на вашому ПК) на віддалений (remote repository, тобто в даному випадку на GitHub) називається "пуш" (push) - від англійського "штовхати" (дослівно - "проштовхувати" зміни).

Скопіювати репозиторій для внесення змін до копії можна двома основними способами:

- клонувати (clone) - тобто просто скопіювати на локальний комп'ютер чи сервер;
- або форкнути (від англійського fork - розвилка) - зробити окрему копію репозиторію (зазвичай чужого) для продовження розробки "іншим шляхом розвилки".

Якщо ви форкнули чужий проект, щоб запропонувати автору конкретні покращення, потрібно по готовності «запулити» їх у вихідний репозиторій — зробивши pull request (запит на зміни).

Це 90% потрібних фактів. Більш нудні подробиці описані в документації та розділі навчання GitHub (https://docs.github.com/en), а також у посібнику по самому Git (https://git-scm.com/book/en/v2)

## *Bitbucket*

Bitbucket – це сервіс для хостингу систем контролю версій коду (Version Control System, VCS). Через таку систему розробники відстежують зміни коду. Офіційний посібник з використання Bitbucket (https://bitbucket.org/product/ru/guides).

Bitbucket стає все більш популярним. В нього є можливість повернутися до потрібної версії коду та виправити помилки.

Особливість Bitbucket у порівнянні з його аналогом GitHub у тому, що ним можуть безкоштовно користуватися до п'яти розробників. До безкоштовної версії входить доступ до необмеженої кількості приватних репозиторіїв — місць, де зберігаються та підтримуються дані.

**Особливості Bitbucket**

**Гнучкість.** Дані в Bitbucket можна імпортувати з багатьох інших сервісів: Git, CodePlex, Google Code, SourceForge і SVN.

**Інтелектуальний семантичний пошук.** У Bitbucket зашитий семантичний пошук JQL. Він сканує синтаксис, щоб знайти визначення, що відповідають запиту, а не просто імена змінних. Це робить пошук швидшим.

**Wiki-функція.** Кожен bitbucket репозиторій  може мати свою власну вікі. Ця функція містить всю необхідну інформацію та нотатки щодо роботи з платформою.

**Інтеграція з Jira.** Jira — програмне забезпечення, яке дозволяє планувати завдання, керувати проектами та відстежувати помилки. Bitbucket легко інтегрується з ним: обидва проекти належать компанії Atlassian. Якщо їх використовувати разом, можна встановити фіксацію коду для автоматичного оновлення завдань Jira.

**Варіанти розміщення Bitbucket**

- **Cloud (Хмара).** Сервіс Bitbucket Cloud розміщується на серверах Atlassian, доступ до нього здійснюється через URL. У складі Bitbucket Cloud є ексклюзивний вбудований інструмент CI/CD - Pipelines, який дозволяє виконувати складання, тестування та розгортання прямо з Bitbucket.
- **Server (Сервер).** Bitbucket Server розміщується локально, у вашому середовищі. Bitbucket Server тісно інтегрується із Bamboo, нашим ефективним інструментом CI/CD, який дозволяє повністю автоматизувати ваш процес. Надається безстрокова ліцензія.
- **Data Center (Дата центр).** Для корпоративних клієнтів пропонується Bitbucket Data Center. Для користувачів він виглядає як один екземпляр Bitbucket Server, але він розміщується на кількох серверах у кластері вашого середовища. Тому цей сервіс має значні переваги перед Bitbucket Server:

*Джерела інформації*

https://www.atlassian.com/ru/git/tutorials/what-is-version-control

https://skillbox.ru/media/code/chto-takoe-github-i-kak-im-polzovatsya/

https://bitbucket.org/product/ru/guides/getting-started/overview

https://bitbucket.org/product/ru/guides/basics/bitbucket-interface