# 12.2. Системи контролю версій

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

Контроль версій, також відомий як керування вихідним кодом, - це практика відстеження змін програмного коду та керування ними. Системи контролю версій – це програмні інструменти, які допомагають командам розробників керувати змінами у вихідному коді, працювати швидше та ефективніше. Системи контролю версій найбільш корисні командам DevOps, оскільки допомагають скоротити час розробки та збільшити кількість успішних розгортань.

*DevOps (Development Operations) - це набір методів та інструментів, які дозволяють автоматизувати та інтегрувати процеси команд розвитку та ІТ-команд. Особлива увага в DevOps приділяється розширенню можливостей команд, їх взаємодії та співпраці, а також автоматизації технологій.*

Програмне забезпечення контролю версій відстежує всі зміни, що вносяться в код, у спеціальній базі даних. При виявленні помилки розробники можуть повернутися назад і виконати порівняння з попередніми версіями коду для виправлення помилок, зводячи до мінімуму проблеми для всіх учасників команди.

Майже у всіх програмних проектах вихідний код є цінним ресурсом, який потрібно берегти. Для більшості команд розробників програмного забезпечення вихідний код – це репозиторій знань та розуміння проблемної галузі, які вони скрупульозно збирали та вдосконалювали. Контроль версій захищає вихідний код від катастрофічних збоїв, випадкових погіршень, спричинених людським фактором, а також від непередбачених наслідків.

Розробники програмного забезпечення, які працюють у командах, постійно пишуть новий вихідний код та змінюють існуючий. Код проекту, програми або програмного компонента зазвичай організований у вигляді структури папок або "дерева файлів". Один розробник у команді може працювати над новою можливістю, а інший у цей час змінювати код для виправлення незв'язаної помилки, тобто кожен розробник може вносити свої зміни в кілька частин дерева файлів.

Контроль версій допомагає командам вирішувати подібні проблеми шляхом відстеження кожної зміни, внесеної кожним учасником, та запобігати виникненню конфліктів при паралельній роботі. Зміни, внесені в одну частину програмного забезпечення, можуть бути несумісними із змінами, внесеними іншим розробником, який працював паралельно. Така проблема має бути виявлена та вирішена згідно з регламентом, не створюючи перешкод для роботи решти команди.

Крім того, під час розробки програмного забезпечення будь-яка зміна може сама по собі призвести до появи нових помилок, і новому програмному забезпеченню не можна довіряти доти, доки воно не пройде тестування. Ось чому процеси тестування та розробки йдуть разом, поки нова версія не буде готова.

Хороше програмне забезпечення для керування версіями працює на будь-якій платформі і не змушує розробника використовувати певну операційну систему чи ланцюжок інструментів. Хороші системи управління версіями забезпечують плавний і безперервний процес внесення змін до коду і не вдаються до громіздкого та незручного механізму блокування файлів, який дає зелене світло одному розробнику, але блокує роботу інших.

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.

### Переваги систем 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 безлічі 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.

Розробляти програмне забезпечення можна і без керування версіями, але такий підхід наражає проект на величезний ризик, і жодна професійна команда не порекомендує застосовувати його. 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 ми зосередимося на одній: системі Git.

## Управління вихідним кодом

Керування вихідним кодом (SCM) використовується для відстеження змін у репозиторії вихідного коду. Рішення SCM відстежують поточну історію змін у базі коду та допомагають вирішувати конфлікти при злитті оновлень від кількох учасників. Термін SCM має синонімічне поняття — «контроль версій».

У міру зростання кількості рядків коду в проекті та кількості співробітників, які беруть участь у розробці програмного забезпечення, зростають зокрема витрати на комунікацію та складність управління. SCM - найважливіший інструмент для зниження організаційного навантаження, викликаного зростаючими витратами на розробку.

### Важливість інструментів керування вихідним кодом

Коли кілька розробників працюють у загальній базі коду, часто вони вносять зміни у загальну частину коду. Окремі розробники можуть працювати над функціями, які здаються ізольованими, тоді як вони використовують загальний модуль коду. Тому розробник 1, який працює над функцією 1, може внести зміни і пізніше дізнатися, що розробник 2, що працює над функцією 2, вніс зміни, що конфліктують.

До впровадження SCM такий сценарій вважався страшним. Розробники могли безпосередньо редагувати текстові файли та переміщувати їх у віддалені розташування за допомогою FTP або інших протоколів. Розробник 1 міг внести зміни, а розробник 2 - ненавмисно зберегти свої зміни поверх коду розробника 1 і знищити його. Управління вихідним кодом (SCM) виступає як механізм захисту від такого сценарію і називається контролем версій .

Разом із SCM з'явилися засоби контролю версій, за допомогою яких можна запобігти втраті напрацювань через перезапис конфліктуючих змін. Вони дозволяють відстежувати зміни, внесені кожним розробником, виявляти конфліктні області та запобігати перезапису. Потім SCM повідомляє розробникам про конфлікти, щоб вони могли безпечно вивчити та усунути їх.

Цей основний механізм запобігання конфліктам має побічний ефект: комунікація в команді розробників починає протікати в пасивному режимі. Команда може переглядати та обговорювати поточну роботу, яка відстежується у SCM (програмне рішення також відстежує всю історію змін у базі коду). Завдяки цьому розробники можуть вивчати та аналізувати правки, які могли призвести до помилок або регресії коду.

## Переваги керування вихідним кодом

Крім управління версіями, в SCM доступний набір інших корисних функцій, які роблять спільну розробку коду зручнішим. Як тільки SCM починається хронологічне відстеження всіх змін у проекті, створюється докладна історія його життєвого циклу. За допомогою цієї історії можна скасовувати зміни у базі коду. Рішення SCM дозволяє миттєво повернути базу коду до стану на попередній час. Завдяки цій вкрай корисній можливості набагато простіше запобігати регресії під час оновлень і виправляти помилки.

Архів SCM із усіма змінами, внесеними протягом життєвого циклу проекту, містить цінні записи для складання приміток до версії проекту. Упорядкований журнал SCM з актуальними даними можна використовувати як примітки до релізу. У ньому доступна зрозуміла та прозора історія роботи над проектом, якою можна поділитися з кінцевими користувачами або з командами, які не задіяні у розробці.

Завдяки SCM команда може знизити накладні витрати на комунікацію та швидше випускати релізи. Розробка без SCM йде повільніше, оскільки учасникам доводиться докладати додаткових зусиль та планувати етапи таким чином, щоб не змішувати роботу над різними релізами. За допомогою SCM розробники можуть самостійно працювати над окремими гілками функцій, а потім виконувати їх злиття.

Самі по собі рішення SCM - це серйозна підмога для команд розробників, за допомогою якої вони можуть скоротити витрати на розробку та ефективно використати трудові ресурси. Сучасна технологія ПО немислима без використання SCM. Контролем версій користуються професіонали , тому вам варто впровадити його у своїй команді.

## Рекомендації щодо керування вихідним кодом

### Робіть комміти частіше

Нічого не варто зробити коміт. Робіть їх частіше, щоб фіксувати оновлення на основі коду. Кожен коміт — це моментальний знімок стану, до якого можна повернути базу коду. Завдяки частим комітам у вас буде багато можливостей повернути чи скасувати роботу. Щоб внести ясність до журналу розробки, можна об'єднати групу коммітів в один коміт за допомогою команди rebase.

### Переконайтеся, що ви працюєте з останньою версією

SCM дозволяє швидко отримувати оновлення від кількох розробників. Локальна копія бази коду може швидко відстати від глобальної копії. Перед внесенням оновлень обов'язково виконуйте команду git pull або fetch для отримання актуального коду. Це допоможе уникнути конфліктів під час злиття.

### Залишайте докладні коментарі

Кожен коміт забезпечується записом у журналі, який відповідає коментарю, вказаному під час створення комміту. Важливо залишати в журналі коммітів описові коментарі, які пояснюють суть зміни, щоб за ними було ясно, які зміни містяться в коміті та навіщо вони були внесені. З коментарів складається канонічна історія розвитку проекту, і вони залишаються доступними для перегляду майбутнім учасникам.

### Переглядайте зміни перед виконанням комміту

У SCM є розділ проіндексованих файлів, який можна використовувати для збирання групи правок, а також для перегляду змін та керування ними перед створенням комміту. Таким чином, розділ проіндексованих файлів виконує роль буферної зони та допомагає уточнити вміст комміту.

### Використовуйте гілки

Розгалуження - це потужний механізм SCM, за допомогою якого розробники можуть створювати окремі напрямки розробки. Розгалуження не пов'язане з якими істотними витратами, отже використовуйте його частіше. Завдяки гілкам кілька розробників можуть паралельно працювати над різними напрямами (зазвичай пов'язані з різними функціями продукту). Після завершення роботи над гілкою її поєднують з основним напрямом розробки.

### Погодьте робочий процес

Завдяки вбудованим можливостям SCM фахівці можуть по-різному брати участь у роботі. Важливо, щоб команди виробили спільні моделі співробітництва, оскільки від робочих процесів SCM залежать шаблони та процеси злиття гілок. Якщо команда не влаштовує загальний робочий процес, можуть виникнути проблеми в комунікації, коли потрібно об'єднати гілки.

## Висновки

SCM є найціннішим інструментом для розробки сучасного програмного забезпечення. Його використовують найкращі команди розробників програмного забезпечення, і вам точно варто взяти з них приклад. SCM відрізняється простотою налаштування нових проектів та високою окупністю інвестицій. Компанія Atlassian пропонує одні з найкращих у світі інструментів для інтеграції SCM , які допоможуть розпочати роботу.

# 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. Система Git була спочатку розроблена в 2005 Лінусом Торвальдсом - творцем ядра операційної системи Linux. Git застосовується для управління версіями в рамках колосальної кількості проектів з розробки програмного забезпечення, як комерційних, так і з відкритим вихідним кодом. Система використовується безліччю професійних розробників програмного забезпечення. Вона чудово працює під управлінням різних операційних систем і може застосовуватися з безліччю вбудованих середовищ розробки (IDE).

Git – система управління версіями з розподіленою архітектурою. На відміну від колись популярних систем на кшталт CVS і Subversion (SVN), де повна історія версій проекту доступна лише в одному місці, у Git кожна робоча копія коду сама по собі є репозиторієм. Це дозволяє всім розробникам зберігати історію змін у повному обсязі.

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

### Продуктивність

Git показує дуже високу продуктивність у порівнянні з безліччю альтернатив. Це можливо завдяки оптимізації процедур фіксації коммітів, створення гілок, злиття та порівняння попередніх версій. Алгоритми Git розроблені з урахуванням 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.

Разом про те розподілена архітектура системи як така забезпечує істотний приріст продуктивності.

Розглянемо приклад: розробник Еліс змінює вихідний код. Вона додає функцію для майбутньої версії 2.0, після чого робить коміт та супроводжує зміни описом. Потім вона розробляє іншу функцію та робить ще один комміт. Зрозуміло, ці зміни зберігаються у вигляді окремих робочих елементів. Потім Еліс перемикається на гілку, що відповідає версії 1.3 того ж ПЗ - так вона зможе виправити баг, що зачіпає цю конкретну версію. Це потрібно, щоб команда Еліс могла випустити версію 1.3.1 із виправленнями до завершення роботи над версією 2.0. Потім Еліс повернеться до гілки версії 2.0 і продовжить роботу над відповідними функціями. Всі ці дії можна виконати без доступу до мережі, тому система Git відрізняється швидкістю і надійністю, навіть якщо працювати в літаку. Коли Еліс буде готова відправити всі внесені зміни до віддаленого репозиторію, їй залишиться лише виконати команду push.

### 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.

У деяких інших системах керування версіями немає захисту від таємного внесення змін. Це може стати серйозною загрозою інформаційній безпеці в будь-якій організації, яка займається розробкою програмного забезпечення.

### 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.

На відміну від SVN, 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.

## Контроль версій за допомогою Git

Git – це найкраще рішення для більшості команд розробки ПЗ. Очевидно, оцінку слід проводити з урахуванням конкретних вимог. Ми лише хочемо перерахувати основні причини, з яких команди вважають за краще використовувати Git.

### Чудові характеристики

Функціональність, продуктивність, безпека та гнучкість Git задовольняють вимоги більшості команд та розробників. Ці якості системи докладно описані вище. При порівнянні системи з більшістю альтернатив багато команд приходять до висновку, що Git має значні переваги.

### Git - визнаний стандарт

Git є найпопулярнішим інструментом свого класу і тому привабливий з низки причин. В Atlassian управління вихідним кодом проектів практично завжди здійснюється в Git.

Багато професійних розробників вже отримали досвід використання Git, а випускники вищих навчальних закладів часто знайомі лише з цією системою. У деяких організаціях працівникам потрібне навчання при переході на Git з інших систем управління версіями, проте багато розробників (як штатних, так і майбутніх співробітників) вже мають необхідні знання.

Однак привабливість Git обумовлена не лише високою популярністю серед розробників. У системі також передбачена інтеграція різних інструментів та сервісів, включаючи інтегровані середовища розробки та власні інструменти Atlassian. До останніх входить настільний клієнт для розподілених систем управління версіями Sourcetree , система відстеження завдань і проектів Jira , а також сервіс розміщення коду Bitbucket .

Розробникам-початківцям, які хочуть придбати цінні навички роботи з інструментами розробки ПЗ, слід вивчити Git як одну з систем управління версіями.

### Git - це якісний проект з відкритим кодом

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. У сфері розробки програмного забезпечення з відкритим вихідним кодом Git безперечно виступає головним наступником успішних систем управління версіями попередніх поколінь, таких як SVN і CVS.

### Критика Git

Git нерідко критикують за складність освоєння: одні терміни можуть бути незнайомі новачкам, а інші мати інше значення. Так, поняття revert (повернення до попередньої версії) у Git має інший сенс, ніж у SVN та CVS. Тим не менш, Git - дуже потужна система, що пропонує користувачам широкі можливості. Їхнє вивчення займе якийсь час, проте засвоєні навички допоможуть учасникам команди працювати набагато швидше.

Команди, які перейшли на Git з нерозподіленої системи керування версіями, можуть захотіти й надалі користуватися центральним репозиторієм. Незважаючи на розподілену архітектуру, Git припускає можливість створення класичного репозиторію, де зберігаються всі зміни проекту. При цьому в Git продуктивність розробників не залежить від доступності та продуктивності "центрального" сервера. Кожному користувачу доступна повна копія репозиторію, в якій він може переглядати всю історію проекту навіть у періоди відсутності з'єднання з мережею та перебоїв у системі. Розподілена архітектура та гнучкість Git дозволяють учасникам проекту працювати у зручному ритмі та користуватися унікальними перевагами, про які вони могли не підозрювати раніше.

Тепер ви розібралися в основах управління версіями, отримали уявлення про Git і дізналися, чому командам розробки програмного забезпечення варто користуватися цією системою. Тепер можна перейти до вивчення переваг, які Git може надати у масштабах організації.

## Навіщо вашій організації потрібне рішення Git

Git для розробки Git для маркетингу Git для керування продуктами Git для дизайнерів Git для служби підтримки Git для служби персоналу Git для управління бюджетом

Перехід із централізованої системи управління версіями на Git змінює підхід команди до розробки програмного забезпечення. А якщо ваша компанія покладається на своє програмне забезпечення для критично важливих програм, зміна процесу розробки вплине на весь бізнес.



У цій статті ми обговоримо, як Git приносить користь усім в організації - від команди розробників до відділу маркетингу. До кінця цієї статті ви побачите, що Git призначається не просто для гнучкої розробки програмного забезпечення , а для гнучкого ведення бізнесу.

## Git для розробки

### Робочий процес із функціональними гілками

Одна з найбільших переваг Git – можливість розгалуження. На відміну від централізованих систем управління версіями, у Git гілки не вимагають багато ресурсів і легко об'єднуються. Це дозволяє реалізувати робочий процес із функціональними гілками, який так популярний серед користувачів Git.



Функціональні гілки служать ізольованим середовищем кожної зміни у базі коду. Коли розробник хоче розпочати роботу над частиною проекту — неважливо, великою чи маленькою, — він створює нову гілку. Завдяки цьому в головній гілці завжди зберігається висока якість коду, прийнятна для розгортання робочого середовища.

Використання функціональних гілок не тільки надійніше, ніж пряме редагування остаточного коду, але й дає організаційні переваги: процес розробки можна узгодити з беклогом agile . Наприклад, можна реалізувати політику, в якій кожен завдання Jira виконуватиметься в окремій функціональній гілці.

### Розподілена розробка

У SVN кожен розробник отримує робочу копію, яка посилається єдиний центральний репозиторій. Git - це розподілена система управління версіями. Замість робочої копії кожен розробник отримує свій локальний репозиторій із повною історією коммітів.



Завдяки локальній доступності всієї історії Git працює швидко, оскільки створення коммітів, перевірки попередніх версій файлу чи обчислення різниці між коммітами не потрібне мережне підключення.

Розподілена технологія також спрощує масштабування команди. Якщо хтось зіпсує робочу гілку в SVN, інші розробники не зможуть вносити зміни, доки вона не буде виправлена. У Git такого блокування немає. Усі можуть спокійно працювати у своїх локальних репозиторіях.

Розподілена технологія, як і функціональні гілки, підвищує надійність середовища. Навіть якщо розробник раптом знищить свій репозиторій, він може просто клонувати чийсь і знову розпочати роботу.

### Пул-реквести

Багато інструментів керування вихідним кодом, такі як Bitbucket , доповнюють базову функціональність Git запитами pull. За допомогою запиту pull можна попросити іншого розробника поєднати одну з ваших гілок зі своїм репозиторієм. Завдяки цьому керівникам легше відстежувати роботу над проектом, а розробникам – отримувати відгуки про свій код, перш ніж інтегрувати його з рештою бази коду.



Оскільки запити pull — це по суті потоки коментарів до функціональної гілки, вони надзвичайно універсальні. Наприклад, якщо розробник стикається з важким завданням, він може відкрити запит pull, щоб попросити допомоги у решти команди. А для молодших розробників запит pull є формальною перевіркою коду, завдяки чому вони можуть не побоюватися, що занапастить весь проект.

### Спільнота

Сьогодні Git розглядається як система контролю версій за умовчанням у різних областях. Якщо у вашій команді використовують Git, швидше за все вам не доведеться навчати нових співробітників робочому процесу, тому що вони і так будуть знайомі з розподіленою розробкою.



Крім того, Git дуже люблять на проектах із відкритим вихідним кодом. Це означає, що ви легко можете використовувати сторонні бібліотеки і заохочувати інших робити форки вашого відкритого вихідного коду.

### Прискорений цикл релізів

Функціональні гілки, розподілена розробка, запити pull та стабільна спільнота зрештою прискорюють цикл релізів. Всі ці можливості сприяють роботі з методології agile , у рамках якої розробникам рекомендується часто ділитися невеликими змінами. Такі зміни просуваються конвеєром розгортання швидше, ніж монолітні релізи, які характерні для централізованих систем управління версіями.



Неважко здогадатися, що Git відмінно підходить для середовищ із безперервною інтеграцією та постачанням. Хуки Git дозволяють запускати скрипти за певних подій усередині репозиторію, що дає можливість автоматизувати розгортання настільки, наскільки ви цього захочете. Можна навіть збирати або розгортати код із певних гілок на різних серверах.

Наприклад, можна налаштувати Git для розгортання останнього комміту з гілки розробки на тестовий сервер відразу після об'єднання з запитом pull. Автоматизація складання та оцінка з боку колег дає максимальну впевненість як код при перенесенні з середовища розробки в проміжне і потім робоче середовище.

## Git для маркетингу

Щоб зрозуміти, як використання Git впливає на маркетингову діяльність компанії, уявіть, що на найближчі кілька тижнів ваша команда розробників запланувала три окремі зміни:

* Вся команда закінчує нову функцію, над якою працювала останні 6 місяців.
* Мері впроваджує не пов'язану з цим невелику функцію, яка впливає лише на клієнтів.
* Рік вносить вкрай необхідні оновлення в інтерфейс користувача.

Якщо ви практикуєте традиційний процес розробки на основі централізованої системи контролю версій, всі ці зміни, швидше за все, виявляться в одному релізі. Відділ маркетингу може зробити лише один анонс, у якому приділить основну увагу нової функції та фактично проігнорує маркетинговий потенціал двох інших оновлень.

З більш коротким циклом розробки, якому сприяє Git, розбити зміни на окремі релізи буде значно простіше. Так маркетологи зможуть доводити до клієнтів більше інформації та робити це частіше. У наведеному вище сценарії відділ маркетингу може розробити три кампанії - по одній для кожної окремої функції - націлені на цілком конкретні сегменти ринку.



Наприклад, можна підготувати велику піар-акцію по новій функції, запис у корпоративному блозі та коротку розсилку по функції Мері, а також кілька статей з теорії дизайну інтерфейсу від Ріка як гостьові публікації в сторонніх блогах. Кожен із цих заходів можна синхронізувати з окремим релізом.

## Git для керування продуктами

У галузі управління продуктом Git має майже такі самі переваги, як і у сфері маркетингу. Чим частіше виходитимуть релізи, тим частіше ви отримуватимете відгуки від клієнтів і тим швидше ви випускатимете оновлення за цими відгуками. Замість того, щоб чекати на реліз ще 8 тижнів, можна відправити рішення клієнтам відразу після того, як розробники напишуть код.



Робочий процес із функціональними гілками забезпечує гнучкість і за зміни пріоритетів. Припустимо, ви пройшли половину циклу релізу і захотіли відкласти одну функцію для іншої, критичної за часом. Ніяких проблем. Перша функція може залишатися у своїй гілці, поки в інженерів не звільниться час, щоб повернутися до неї.

Та ж функціональність спрощує управління інноваційними проектами, бета-тестами та швидкими прототипами у вигляді незалежних баз коду.

## Git для дизайнерів

Функціональні гілки добре підходять для швидкого прототипування. Неважливо, хочуть дизайнери інтерфейсу реалізувати зовсім новий сценарій користувача або просто замінити деякі значки. Переключившись на нову гілку, вони без загрози для існуючих функцій зможуть поекспериментувати зі змінами в ізольованому середовищі та подивитися, як зміни виглядатимуть у реальному робочому екземплярі продукту.



Інкапсульовані таким чином зміни інтерфейсу простіше показувати зацікавленим особам. Наприклад, якщо технічний директор хоче подивитися, над чим працює команда дизайнерів, йому достатньо перейти на відповідну гілку.

Запити pull дозволяють зробити ще один крок уперед та надати зацікавленим сторонам місце для організованого обговорення нового інтерфейсу. Дизайнери можуть вносити будь-які необхідні зміни і відповідні коміти з'являться в запиті pull. Таким чином, кожен може брати участь у процесі ітерації.

Мабуть, головна перевага прототипування за допомогою гілок полягає в тому, що воно дозволяє однаково легко інтегрувати зміни до робочого середовища або відкинути їх. Ніхто не змушує робити ні те, ні інше. Завдяки цьому дизайнери та розробники інтерфейсу можуть експериментувати та доводити до клієнтів лише найкращі ідеї.

## Git для служби підтримки

Служба підтримки клієнтів часто дивиться на оновлення з іншого погляду, ніж менеджери з продуктів. Клієнти звертаються до служби підтримки проблем. Якщо проблема викликана програмним забезпеченням вашої компанії, виправлення бага потрібно випустити якнайшвидше.

З оптимізованим циклом розробки Git виправлення багів можна не відкладати до наступного монолітного релізу. Розробник може випустити патч і відправити його безпосередньо до робочого середовища. Чим швидше виправляються помилки, тим більшими будуть клієнти і тим менше буде повторних заявок до служби підтримки. Замість того, щоб твердити: «Вибачте, ми зараз цим займемося», — у службі підтримки клієнтів зможуть відповідати: «Ми вже все виправили!»

## Git для служби персоналу

Процес розробки програмного забезпечення певною мірою впливає на те, кого ви наймаєте. Наймати інженерів, знайомих із вашими технологіями та робочими процесами, завжди корисно, проте використання Git дає й інші переваги.

Співробітники тягнуться в компанії, які надають можливості кар'єрного зростання, а досвід роботи з Git як у великих, так і в маленьких організаціях принесе користь будь-якому програмісту. Вибираючи Git як систему управління версіями, ви вирішуєте залучати прогресивних розробників.

## Git для управління бюджетом

Git – це насамперед ефективність. Розробники зможуть заощаджувати час на всьому: від передачі комітів через мережу до трудових витрат на інтеграцію змін до централізованої системи контролю версій. У Git ефективніше працюють навіть молодші розробники, оскільки вони можуть працювати у безпечному середовищі. Усе це впливає прибуток вашого технічного відділу.

## Висновки

Важливо розуміти, ця ефективність сягає далеко за межі команди розробників. Відділ маркетингу не витрачає сили на підготовку матеріалів з непопулярних функцій. Дизайнери мають можливість тестувати нові інтерфейси на реальному продукті з невеликими накладними витратами. Служба підтримки може негайно реагувати на скарги клієнтів.

Гнучка методологія розробки вимагає оперативно виявляти перспективні ідеї, вкладати сили в успішну роботу і витрачати час на нежиттєздатні задуми. Git збільшить результати всіх видів діяльності вашої компанії та забезпечить більш ефективну роботу кожного відділу.

Більш детальну інформацію про встановлення та використання Git дивитися тут

<https://www.atlassian.com/ru/git/tutorials/learn-git-with-bitbucket-cloud>

# GitHub і його відмінність від Git

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

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 купу 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. До речі, деякі умільці примудряються збирати в GitHub цілі бібліотеки книг і статей, а не програмістські ліби :)

І так, якщо ви незадоволені якимись фічами в улюбленій відкритій програмі і вона викладена на GitHub, ви завжди можете прийти і посваритися в коментарях до проекту :) А найкраще оформити issue (ми розповімо, що це) і самостійно пофіксувати проблему на радість усім користувачам. Не забувайте і дякувати авторам класних відкритих проектів — донатам і просто теплим словам. Їм буде дуже приємно.

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

А ще не варто плутати GitHub та Git. GitHub — лише одна з реалізацій системи контролю версій Git (тільки погляньте на повний список Git-клієнтів з графічним інтерфейсом), 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).

Ну як, звучить круто? Тоді приступайте до нашого гайду про те, як користуватися GitHub, щоб у всьому розібратися і взагалі зрозуміти, чи він вам потрібен прямо зараз.

## Для яких цілей використовують GitHub

Безумовно, GitHub потрібний не всім. Припустимо, ви ще тільки вчитеся кодувати або повільно робите невеликий проект для особистого користування - і вас влаштовує зберігання проекту на локальній машині. Може, зараз ви просто навчаєте мову, яка вам подобається, і на даному етапі не хочете хапатися за все відразу.

В першу чергу GitHub необхідний проектам із частими оновленнями, безліччю версій, великою кількістю файлів, необхідністю синхронізації розробки та зручного розгортання.

Хоча і тут є винятки - навіть деякі з ключових розробників ядра Linux досі обмінюються оновленнями коду за допомогою електронної пошти та архівів із файлами :)

І справді, є безліч 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 або «ВКонтакті» (милицево, так, але цілком реально ).

А ще можна накидати список змін у нотатках у телефоні/на холодильнику текстом у приватному Telegram-каналі. Можна деплоїти проект за допомогою простого скачування та розпакування ZIP-архіву з файлами вашої програми (особливо якщо мета – просто показати програму другові чи дівчині, якій ви прийшли «допомогти з ноутбуком» ^\_^). Зрештою, можна повідомляти про баги у вашому улюбленому фреймворку спільноті анонімів у паблиці VK — обурюватися разом дуже весело.

Всі ці способи по-своєму хороші, але для роботи в ІТ потрібно звикати до GitHub: це стандарт індустрії.

Цікавий факт: нещодавно з'явилася російська альтернатива GitHub під назвою GitFlic. Команда сервісу заявила, що має намір дати "новий імпульс розробці вітчизняних операційних систем, програм, додатків та серверних рішень". Серед можливостей, що чіпляють, — інтеграція з Telegram.

Більшість необхідних можливостей GitHub безкоштовна, хоча платформа має і платні тарифи — проте з ними можна довго не стикатися, бо безкоштовних звичайному розробнику вистачає з лишком.

## Основні концепції GitHub

Початківців інтерфейс GitHub може збивати з пантелику: за роки з моменту створення майданчик обросла безліччю інструментів, але головне залишається незмінним.

Майже вся термінологія успадковується у Git. 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.

Як заливати файли, створювати репозиторії та проводити інші операції, ми розглянемо у наступному розділі, так що не лякайтеся згадки різних дій у визначеннях термінів – все покажемо з картинками :)

### Репозиторій

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

Репозиторії можуть бути публічними та приватними, у 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 бажання 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), прочитати як застосувати все це на практиці дивитися тут <https://skillbox.ru/media/code/chto-takoe-github-i-kak-im-polzovatsya/>

# Bitbucket

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

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 тут (https://bitbucket.org/product/ru/guides).

Bitbucket стає все більш популярним. У нього, як і в його конкурентів, 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. При цьому до безкоштовної версії входить доступ до необмеженої кількості приватних репозиторіїв — місць, де зберігаються та підтримуються дані.

Оновлення підписки на сервіс збільшує кількість хвилин складання та обсяг сховища великих файлів, включених у пакет: 3$ (приблизно 220 руб.) на місяць для стандартного облікового запису та 6$ (приблизно 440 руб.) для преміум-версії. У сервісу GitHub є корпоративна версія; її аналог - преміум у Bitbucket - коштує 21 $ на місяць (1550 руб.).

## Інші 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 ваш синтаксис, щоб знайти визначення, що відповідають вашому запиту, а не просто імена змінних. Це робить пошук швидшим.

**Wiki-функція.** Кожен bitbucket репозиторій (репозиторій) може мати свою власну вікі. Ця функція включає всю необхідну інформацію та нотатки щодо роботи з платформою. Крім того, ви можете використовувати вікі для постійного розміщення документації ПЗ.

**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 ви використовуєте їх разом, можете 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 ми пропонуємо 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:

## Основні поняття

Нижче наведено важливі поняття та терміни, які варто знати, щоб ефективно працювати з Bitbucket. Деякі з цих понять запозичені з Git, інші застосовуються виключно у зв'язку з Bitbucket.

### Branch Гілка

Гілка є окремим напрямом розробки. Гілки виступають як абстрактне уявлення для процесу редагування/індексації/комміту. Можна розглядати їх як спосіб запросити новий робочий каталог, розділ проіндексованих файлів та історію проекту. Нові комміти записуються в історію поточної гілки, що призводить до утворення роздоріжжя в історії проекту.

### Fork (відгалуження)

Щоб не використовувати один репозиторій на сервері як центральну базу коду, можна скористатися відгалуженнями (форками), щоб у кожного розробника був репозиторій на сервері. Таким чином, у кожного автора буде не один, а два репозиторії Git: один закритий локальний і один відкритий на сервері.

### HEAD

Вказівник на поточний знімок у Git. По суті, команда git checkout просто оновлює покажчик HEAD, щоб він посилався на вказану гілку або коміт. Коли HEAD вказує на гілку, Git мовчить, але при спробі переключитися на коміт система перетворюється на стан detached HEAD (відкріплений покажчик HEAD).

### Master (Головна Гілка)

Гілка за замовчуванням. При кожному створенні репозиторію Git створюється гілка main; вона ж стає активною гілкою.

### Pipelines

Bitbucket Pipelines – це комплексний сервіс CI/CD, вбудований у Bitbucket. З його допомогою ви можете виконувати автоматичне складання, тестування і навіть розгортання коду, використовуючи конфігураційний файл з вашого репозиторію.

### Pipes

Bitbucket Pipes – це короткі фрагменти коду, які можна поміщати у конвеєр Pipeline для виконання складних та ефективних операцій. Фрагменти Pipes полегшують побудову ефективних автоматизованих робочих процесів CI/CD та дозволяють швидко налагодити роботу.

### Project (Проект)

Проект є вмістищем репозиторіїв. Завдяки проектам, в які організовуються репозиторії, командам простіше зосередитися на досягненні мети, роботі над продуктом чи процесом. Проекти можуть бути помітні для всіх або тільки для певних людей.

### Pull requests (Пул реквест)

Запити pull полегшують спільну роботу розробників у Bitbucket. Вони забезпечують зручний веб-інтерфейс для обговорення пропонованих змін до їх включення до офіційного проекту.

### Робочий каталог

Каталог містить файли, над якими ви працюєте. У ньому зазвичай міститься вміст гілки комміта, на який вказує HEAD, і всі локальні зміни, які були здійснені, але ще не підтверджені за допомогою комміту.

Детальну інформацію по роботі з Bitbucket читати тут

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