**Лекція №1. Системи контролю версіями**

На даній лекції ми розглянемо поняття «система контролю версій», проаналізуємо наявні системи контролю, їх історію, можливості, основні переваги та недоліки.

План

1. Поняття «система контролю версій».
2. Централізовані та розподільні системи контролю версіями.
3. Установка Git.

**Список використаних джерел**

1. https://gitscm.com/book/uk/v2/%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%9F%D1%80%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%83%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8E%D0%B2%D0%B5%D1%80%D1%81%D1%96%D0%B9 .
2. http://kissarat.blogspot.com/2013/03/blog-post.html

Що таке «система контролю версій», і чому це важливо?

Напевно, багатьом знайома ситуація, коли при роботі над проектом, виникає необхідність внести зміни, але при цьому потрібно зберегти працездатний варіант, в такому випадку, як правило, створюється нова папка, назва якої швидше за все буде «Нова папка» з доповненням у вигляді дати або невеликої позначки, в неї копіюється робоча версія проекту і вже з ним проводиться робота. Згодом кількість таких папок може значно зрости, що створює труднощі в питанні «відкату» на попередні версії, відстеження змін і т.п. Ця ситуація значно погіршується, коли над проектом працює кілька людей.

Для вирішення таких проблем, як раз, і використовується система контролю версій, вона дозволяє комфортно працювати над проектом як індивідуально, так в колективі.

Система контролю версій (VCS) – це система, що записує зміни у файл або набір файлів протягом деякого часу, так що ви зможете повернутися до певної версії пізніше. Ви можете використовувати контроль версій практично для будь-яких типів файлів.

VCS відстежує зміни в файлах, надає можливості для створення нових і злиття існуючих гілок проекту, виробляє контроль доступу користувачів до проекту, дозволяє відкочувати виправлення і визначати хто, коли і які зміни вносив в проект.

Основним поняттям VCS є репозиторій (repository) – спеціальне сховище файлів і папок проекту, зміни в яких відстежуються. У розпорядженні розробника є так звана «робоча копія» (working copy) проекту, з якою він безпосередньо працює. Робочу копію необхідно періодично синхронізувати з репозиторієм, ця операція передбачає відправку до нього змін, які користувач вніс у свою робочу копію (така операція називається commit) і актуалізацію робочої копії, в процесі якої до користувача завантажується остання версія зі сховищ (цей процес носить назву update ).

2. ***Системи контролю версій можна розділити на дві групи: розподілені і централізовані.***

Централізовані системи контролю версій є додатки типу клієнт-сервер, коли репозиторій проекту існує в єдиному екземплярі і зберігається на сервері. Доступ до нього здійснювався через спеціальну клієнтську програму. Як приклади таких програмних продуктів можна привести CVS, Subversion.

Традиційно, де-факто системою контролю версій стала CVS (Система Паралельних Версій). З усіх цих можливостей CVS підтримує тільки об'єднання змін. Це зріла і надійна система контролю версій. Багато проектів з відкритими вихідними кодами, включаючи KDE, GNOME і Mozilla використовують CVS. Більшість центрів відкритих вихідних кодів, такі як SourceForge, пропонують CVS як сервіс, тому її використовують у багатьох інших проектах.

Система CVS має архітектуру клієнт-сервер. Зазвичай сервер і клієнт з'єднуються через локальну мережу або через Інтернет, але можуть працювати і на одній машині, якщо необхідно вести історію версій локального проекту. Серверне програмне забезпечення зазвичай працює під управлінням Unix, тоді як CVS клієнти доступні у всіх популярних операційних системах. Сервер зберігає в репозиторії поточну версію (версії) проекту і історію змін, а клієнт з'єднується з ним, щоб отримати потрібну йому версію або записати нову. Отримавши з сервера потрібну версію, клієнт створює локальну копію проекту (або його частини) - так звану робочу копію. Після того як в файли, що знаходяться в робочій копії, внесені необхідні зміни, вони пересилаються на сервер.

Користувачі мають можливість порівнювати різні версії файлів, переглянути історію змін або навіть отримати історичний образ проекту на певну дату. Багато проектів формату «Відкритих ресурсів» дозволяють анонімний доступ на читання. Дана можливість передбачає, що користувачі, можуть отримати доступ до версій файлів без пароля. CVS також може містити різні гілки проекту. Наприклад, стабільна версія проекту може становити одну гілку, до якої вносяться тільки виправлення помилок, тоді як активна розробка може вестися в паралельній гілці, яка включає значні поліпшення або зміни з моменту виходу стабільної версії. Крім того, варто також зазначити, що дана система використовує механізм дельта-компресії, щоб більш ефективно ніж зберігати різноманітні версії одного і того ж файлу.

Незважаючи на свою популярність, CVS має обмеження. Наприклад, вона не підтримує перейменування файлів і директорій. Крім того, бінарні файли не обробляються досить добре. CVS – нерозподілена система і атомарні фіксації змін не підтримуються. Так як вже є кращі альтернативи, які містять більш широкий набір функцій, ви, можливо, вважатимете за краще почати новий проект використовуючи щось інше.

Як плюс, CVS дуже добре документована в своїй книзі і в багатьох онлайн посібниках. Також існують безліч графічних клієнтів і доповнень.

Subversion прагне бути кращою альтернативою CVS. Вона підтримує більшість угод CVS, включаючи велику частину набору команд, тому користувачі CVS швидко відчувають себе як вдома. Subversion пропонує багато корисних поліпшень в порівнянні з CVS: копіювання і перейменування файлів і директорій, справжні атомарні фіксації, ефективна обробка бінарних файлів, здатність мережевої роботи по HTTP (і HTTPS). Subversion також має Win32 клієнт і сервер.

Таким чином, можна виділити наступні можливості Subversion:

* Реалізовано більшість можливостей CVS;
* Відстежується історія файлів, директорій і метаданих файлів, в тому числі при перейменуванні і копіюванні;
* Публікації змін атомарні;
* Можливість організації доступу до сховища Subversion через Apache по протоколу WebDAV / DeltaV;
* Можливість установки автономного сервера Subversion з доступом за власним протоколом;
* «Дешеві» операції створення гілок і міток (потрібна невелика фіксована кількість тимчасових і дискових ресурсів);
* Багаторівнева архітектура бібліотек, спочатку розрахована на клієнт-серверну модель;
* Клієнт-серверний протокол пересилає по мережі тільки різницю між об'єктами, коли це можливо;
* Витрати ресурсів пропорційні розміру змін, а не розміром даних, які порушені змінами;
* Два можливих внутрішніх формати сховища: база даних або простий файл;
* Однаково ефективна робота і з текстовими, і з двійковими файлами;
* Висновок клієнта командного рядка однаково зручний і для читання, і для розбору програмами;
* Часткова локалізація повідомлень
* Бібліотеки для мов PHP, Python, Perl, Java;
* Можливість віддзеркалення сховища.

Subversion недавно почала бета-період, після того як була довгий час в альфа-періоді. Тому, вона ще може мати невеликі примхи і її продуктивність в деяких місцях невелика. І все-таки, вона дуже корисна для свого бета-періоду і була такою навіть в більшій частині свого альфа-періоду. Сервіс Subversion на основі HTTP (або HTTPS) важкий для розгортання, у порівнянні з іншими системами, так як він вимагає встановленої служби Apache2 зі своїм власним спеціальним модулем.

Також є «svnserver» сервер, який менш здатний, але більш простий в установці і використовує спеціальний протокол. Крім того, підтримка Subversion об'єднання змін обмежена і схожа в цьому на CVS (тобто, об'єднання гілок, де файли були переміщені не буде виконано коректно). Вона також вимоглива до ресурсів, особливо на великих операціях.

Subversion пропонує два варіанти організації репозиторіїв. Репозиторії першого типу використовують для зберігання базу даних на основі Berkeley DB, репозиторії другого типу – в звичайних файлах спеціального формату (доступ до даних організовується за допомогою власних бібліотек, без використання сторонніх баз даних). Обидва типи репозиторіїв забезпечують достатню надійність при правильній організації, кожна з них має свої переваги й недоліки. Вважається, що другий тип легше правильно налаштувати, вона вимагає меншої уваги від адміністратора. Subversion добре документована в своїй безкоштовної онлайн книзі, «Контроль Версій з Subversion». Невелика онлайн система допомоги, що поставляється з Subversion клієнтом може також бути корисною в якості довідника. Subversion має багато доповнень, але вони все ще менш зрілі, ніж їх CVS конкуренти.

Розподілені системи контролю версій (Distributed Version Control System, DVCS) дозволяють зберігати репозиторій (його копію) для кожного розробника, що працює з даною системою. При цьому можна виділити центральний репозиторій (умовно), в який будуть відправлятися зміни з локальних і, з ним же ці локальні репозиторії будуть синхронізуватися. При роботі з такою системою, користувачі періодично синхронізують свої локальні репозиторії з центральним і працюють безпосередньо зі своєю локальною копією. Після внесення достатньої кількості змін в локальну копію вони (зміни) відправляються на сервер. При цьому сервер, найчастіше, вибирається умовно, тому що в більшості DVCS немає такого поняття як «виділений сервер з центральним репозиторієм».

Велика перевага такого підходу полягає в автономії розробника при роботі над проектом, гнучкості загальної системи і підвищення надійності, завдяки тому, що кожен розробник має локальну копію центрального сховища. Дві найбільш відомі DVCS – це Git і Mercurial.

Почнемо з Mercurial, ця система являє собою вільну DVCS, яка побудована таким чином, що в ній відсутнє поняття центрального сховища, для роботи з цією VCS використовується (як правило) консольна утиліта «hg». Mercurial має всі можливості системи контролю версій, такими як розгалуження, злиття, синхронізація з іншими репозиторіями. Даний проект використовують і підтримують велику кількість великих розробників, серед них Mozilla, OpenOffice, OpenJDK і багато інших. Сам продукт написаний на мові Python і доступний на більшості сучасних операційних систем (Windows, Mac OS, Linux), також існує значна кількість утиліт з графічним інтерфейсом для роботи з Mercurial.

Розглянемо основні переваги та недоліки Mercurial.

*Переваги:*

* Швидка обробка даних.
* Кросплатформенна підтримка.
* Можливість роботи з декількома гілками проекту.
* Простота в обігу.
* Можливість конвертації репозиторіїв інших систем підтримки версій, таких як CVS, Subversion, Git, Darcs, GNU Arch, Bazaar і ін.

*Недоліки:*

* Можливі (але надзвичайно низькі) збіги хеш-коду відмінних за змістом ревізій.
* Орієнтований на роботу в консолі.

Отже, простий і відточений інтерфейс, і набір команд, можливість імпортувати репозиторії з інших систем контролю версій, - зроблять перехід на Mercurial і навчання основним особливостями безболісним і швидким. Навряд чи це займе більше кількох днів.

Надійність і швидкість роботи дозволяють використовувати його для контролю версій величезних проектів. Все це робить mercurial гідним конкурентом git'а.

Основним конкурентом Mercurial на ринку розподілених систем контролю версій є Git, який, на сьогоднішній день, виграв гонку за лідерство.

***Історія Git`a***. З лютого 2002 року для розробки ядра Linux'а більшістю програмістів стала використовуватися система контролю версій BitKeeper. Досить довгий час з нею не виникало проблем, але в 2005 році Ларі МакВоем (розробник BitKeeper'а) відкликав безкоштовну версію програми.

Розробляти проект масштабу Linux без потужної і надійної системи контролю версій - неможливо. Одним з кандидатів і найбільш підходящим проектом виявилася система контролю версій Monotоne, але Торвальдса Лінуса не влаштувала її швидкість роботи. Так як особливості організації Monatone не дозволяли значно збільшити швидкість обробки даних, то 3 квітня 2005 року Лінус приступив до розробки власної системи контролю версій – Git.

Git – це гнучка, розподілена (без єдиного сервера) система контролю версій, що дає масу можливостей не тільки розробникам програмних продуктів, але і письменникам для зміни, доповнення та відстеження зміни «рукописів» і сюжетних ліній, і вчителям для коригування та розвитку курсу лекцій, і адміністраторам для ведення документації, і для багатьох інших напрямків, що вимагають управління історією змін. У кожного розробника, який використовує Git, є свій локальний репозиторій, що дозволяє локально керувати версіями. Потім, збереженими в локальний репозиторій даними, можна обмінюватися з іншими користувачами. Часто при роботі з Git створюють центральний репозиторій, з яким інші розробники синхронізуються. Приклад організації системи з центральним репозиторієм - це проект розробки ядра Linux'a. В цьому випадку всі учасники проекту ведуть свої локальні розробки і безперешкодно викачують оновлення з центрального сховища. Коли необхідні роботи окремими учасниками проекту виконані і налагоджені, вони, після посвідчення власником центрального сховища в коректності і актуальності виконаної роботи, завантажують свої зміни в центральний репозиторій.

Наявність локальних репозиторіїв також значно підвищує надійність зберігання даних, так як, якщо один з репозиторіїв вийде з ладу, дані можуть бути легко відновлені з інших репозиторіїв.

Робота над версіями проекту в Git може вестися в декількох гілках, які потім можуть з легкістю повністю або частково об'єднуватися, знищуватися, відкочуватися і розростатися в усі нові і нові гілки проекту. Можна довго обговорювати можливості Git'а, але для стислості і більш простого сприйняття наведемо основні переваги і недоліки цієї системи управління версіями.

*Переваги:*

* Надійна система порівняння ревізій і перевірки коректності даних, засновані на алгоритмі хешування SHA1 (Secure Hash Algorithm 1).
* Гнучка система розгалуження проектів і злиття гілок між собою.
* Наявність локального сховища, що містить повну інформацію про всі зміни, дозволяє вести повноцінний локальний контроль версій і заливати в головний репозиторій тільки повністю пройшли перевірку зміни.
* Висока продуктивність і швидкість роботи.
* Зручний і інтуїтивно зрозумілий набір команд.
* Безліч графічних оболонок, що дозволяють швидко і якісно вести роботи з Git'ом.
* Можливість робити контрольні точки, в яких дані зберігаються без дельта компресії, а повністю. Це дозволяє зменшити швидкість відновлення даних, так як за основу береться найближча контрольна точка, і відновлення йде від неї. Якби контрольні точки були відсутні, то відновлення великих проектів могло б займати години.
* Широка поширеність, легка доступність і якісна документація.
* Гнучкість системи дозволяє зручно її налаштовувати і навіть створювати спеціалізовані контролю системи або призначені для користувача інтерфейси на базі git.
* Універсальний мережевий доступ з використанням протоколів http, ftp, rsync, ssh і ін.

*Недоліки*:

* Unix - орієнтованість. На даний момент відсутня зріла реалізація Git, сумісна з іншими операційними системами.
* Можливі (але надзвичайно низькі) збіги хеш-коду відмінних за змістом ревізій.
* Не відстежується зміна окремих файлів, а тільки всього проекту цілком, що може бути незручно при роботі з великими проектами, що містять безліч незв'язаних файлів.
* При початковому (першому) створенні сховища та синхронізації його з іншими розробниками, буде потрібен досить тривалий час для скачування даних, особливо, якщо проект великий, так як потрібно скопіювати на локальний комп'ютер весь репозиторій.

Отже, Git – гнучка, зручна і потужна система контролю версій, здатна задовольнити абсолютна більшість користувачів. Існуючі недоліки поступово віддаляються і не приносять серйозних проблем користувачам. Якщо ви ведете великий проект, територіально віддалений, і тим більше, якщо часто доводиться розробляти програмне забезпечення, не маючи доступу до інших розробників (наприклад, ви не хочете втрачати час при перельоті з країни в країну або під час поїздки на роботу), можна робити будь-які зміни і зберігати їх в локальному репозиторії, відкочуватися, перемикатися між гілками і т . д.). Git – один з лідерів систем контролю версій.

Серед великих проектів, в рамках яких використовується git, можна виділити ядро Linux, Qt, Android. Git вільний і розповсюджується під ліцензією GNU GPL 2 і, також як Mercurial, доступний практично на всіх операційних системах. За своїми базовими можливостями git схожий на Mercurial (і іншими DVCS), але завдяки ряду переваг (висока швидкість роботи, можливість інтеграції з іншими VCS, зручний інтерфейс) і дуже активній спільноті, сформованій навколо цієї системи, git вийшов в лідери ринку розподілених систем контролю версій. Необхідно відзначити, що незважаючи на велику популярність таких систем як git, великі корпорації, подібні Google, використовують свої VCS.

**Установка Git.**

Для установки Git під Windows необхідно попередньо завантажити дистрибутив. Для цього перейдіть на сторінку <https://git-scm.com/>

Для того щоб завантажити Git потрібно натиснути на кнопку Downloads for Windows, розташовану в правій частині вікна.

Процес подальшої установки Git виглядає так.

1. Запустити інсталяційний файл

2. Ознайомитися, якщо є бажання, з ліцензійною угодою і натиснути на кнопку Next.

3. Вибрати компоненти, які слід встановити.

4. Вказати спосіб використання Git.

У цьому вікні доступні три можливих варіанти:

Use Git from Git Bash only

Змінна PATH не змінюється і робота з Git можлива тільки через спеціалізовану оболонку, яка називається Git Bash.

Use Git from the Windows Command Prompt

В цьому випадку відбувається мінімальна модифікація змінної оточення PATH, яка дозволить працювати з Git через командну стоку Windows. Робота через Git Bash також можлива.

Use Git and optional Unix tools from the Windows Command Prompt

У змінну PATH вноситься значна кількість модифікацій, які дозволять, в рамках командного рядка Windows, використовувати як Git так і утиліти Unix, які поставляються разом з дистрибутивом Git.

Наша рекомендація: опція Use Git from the Windows Command Prompt.

5. Налаштування правил закінчення рядка.

Існує два варіанти формування кінця рядка в текстових файлах - це Windows стиль і Unix стиль. Дане вікно дозволяє вибрати одну з опцій, що визначають правило формування закінчення рядка:

***Checkout Windows-style, commit Unix-style line endings***

Checkout (операція вилучення документа зі сховища і створення робочої копії) проводиться в Windows стилі, а commit (операція відправки змін в репозиторій) в Unix стилі.

***Checkout as-is, commit Unix-style line endigns***

Checkout проводиться в тому форматі, в якому дані зберігаються в репозиторії, а commit здійснюється в Unix стилі.

***Checkout as-is, commit as-is***

Checkout і commit виробляються без додаткових перетворень.

Наша рекомендація: опція Checkout Windows-style, commit Unix-style line endings.

6. Вибір емулятора терміналу, який буде використаний з Git Bash (без змін).

7. Встановлення додаткових параметрів.

Доступні наступні параметри:

***Enable file system caching***

Включення операції кешування при роботі з файлами. Ця опція дозволить значно підвищити продуктивність.

***Enable Git Credential Manager***

Надає можливість роботи з захищеним сховищем.

***Enable symbolic links***

Активує роботу з символьними посиланнями.

Наша рекомендація: опції Enable file system caching і Enable Git Credential Manager.

8. Завершення установки

Після натискання на кнопку Install буде проведена установка Git на Windows, після закінчення установки користувач отримає відповідне повідомлення.

Потім, відкриваємо програму і проводимо налаштування :

Встановлюємо ім'я та адресу електронної пошти

Якщо ви ніколи раніше не використовували git, для початку вам необхідно здійснити установку. Виконайте наступні команди, щоб git дізнався про ваше ім'я та електронну пошту. Якщо git вже встановлено, можете переходити до розділу закінчення рядків.

git config --global user.name "Your Name"

git config --global user.email your\_email@whatever.com

**Параметри установки закінчень рядків**

git config --global core.autocrlf true

git config --global core.safecrlf true