ЛАБОРАТОРНА РОБОТА №8

Захист цілісності засобами контролю версій

Мета роботи

Навчитися використовувати систему контролю версій Git для захисту цілісності даних.

1 Теоретичні відомості

Система контролю версій - це система, що записує зміни у файл або набір файлів протягом деякого часу, так що ви зможете повернутися до певної версії пізніше. Як приклад, в цій книзі, для файлів, що знаходяться під контролем версій, буде використовуватися код програмного забезпечення, хоча насправді ви можете використовувати контроль версій практично для будь-яких типів файлів.

Якщо ви графічний або веб-дизайнер і хочете зберегти кожну версію зображення або макета (швидше за все, захочете), система контролю версій (далі СКВ) якраз те, що потрібно. Вона дозволяє повернути вибрані файли до попереднього стану, повернути весь проект до попереднього стану, побачити зміни, побачити, хто останній міняв щось і спровокував проблему, хто вказав на проблему і коли, та багато іншого. Використання СКВ також в цілому означає, що, якщо ви зламали щось або втратили файли, ви просто можете все виправити. Крім того, ви отримаєте все це за дуже невеликі накладні витрати.

Локальні системи контролю версій

Багато людей в якості одного з методів контролю версій застосовують копіювання файлів в окрему директорію (можливо навіть директорію з відміткою за часом, якщо вони достатньо розумні). Даний підхід є дуже поширеним завдяки його простоті, проте він, неймовірним чином, схильний до появи помилок. Можна легко забути в якій директорії ви знаходитеся і випадково змінити не той файл або скопіювати не ті файли, які ви хотіли.

Щоб справитися з цією проблемою, програмісти давно розробили локальні СКВ, що мають просту базу даних, яка зберігає всі зміни в файлах під контролем версій.

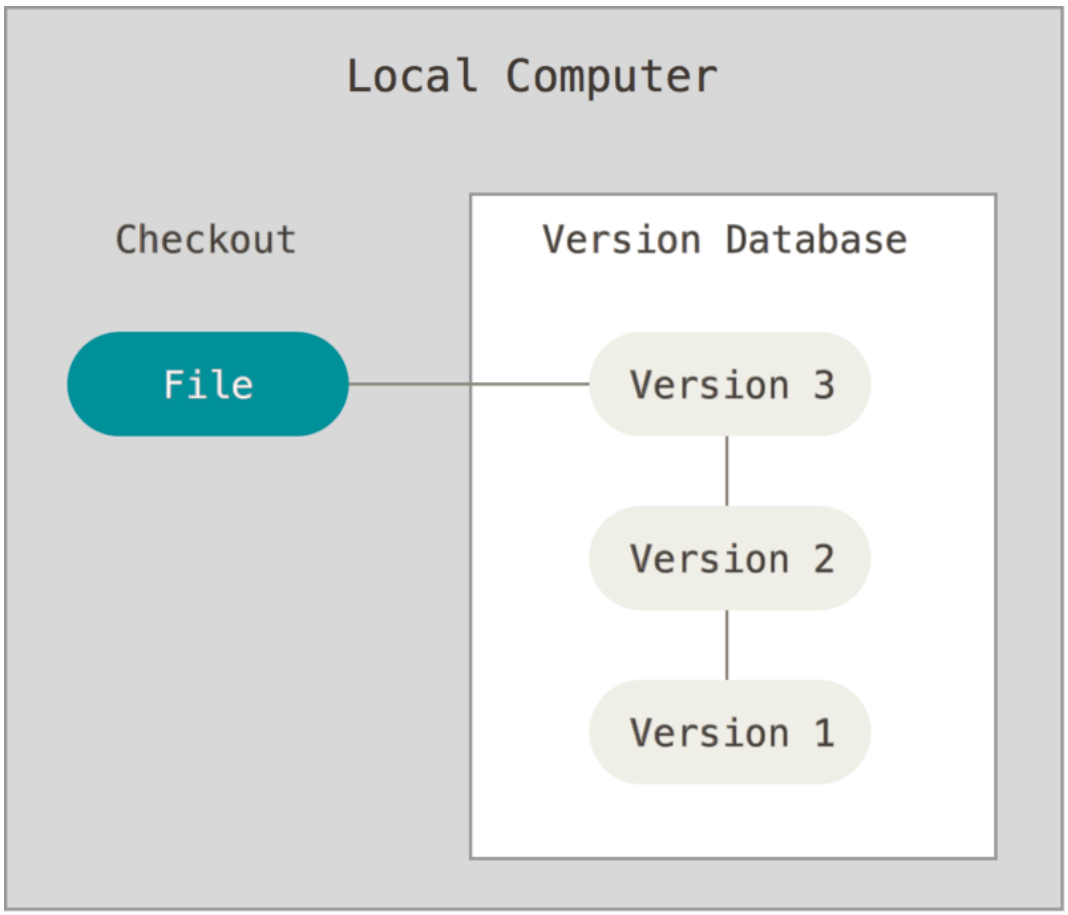


Рисунок 1 – Локальні системи контролю версій.

Одним з найбільш поширених інструментів СКВ була система під назвою RCS, яка досі поширюється з багатьма комп'ютерами сьогодні. RCS зберігає набори латок (тобто, відмінності між файлами) в спеціальному форматі на диску; він може заново відтворити будь-який файл, як він виглядав, в будь-який момент часу, шляхом додавання всіх латок.

Централізовані системи контролю версій

Наступним важливим питанням, з яким стикаються люди, є необхідність співпрацювати з іншими розробниками. Щоб справитися з цією проблемою, були розроблені централізовані системи контролю версій (ЦСКВ). Такі системи як CVS, Subversion і Perforce, мають єдиний сервер, який містить всі версії файлів, та деяке число клієнтів, які отримують файли з центрального місця. Протягом багатьох років, це було стандартом для систем контролю версій.

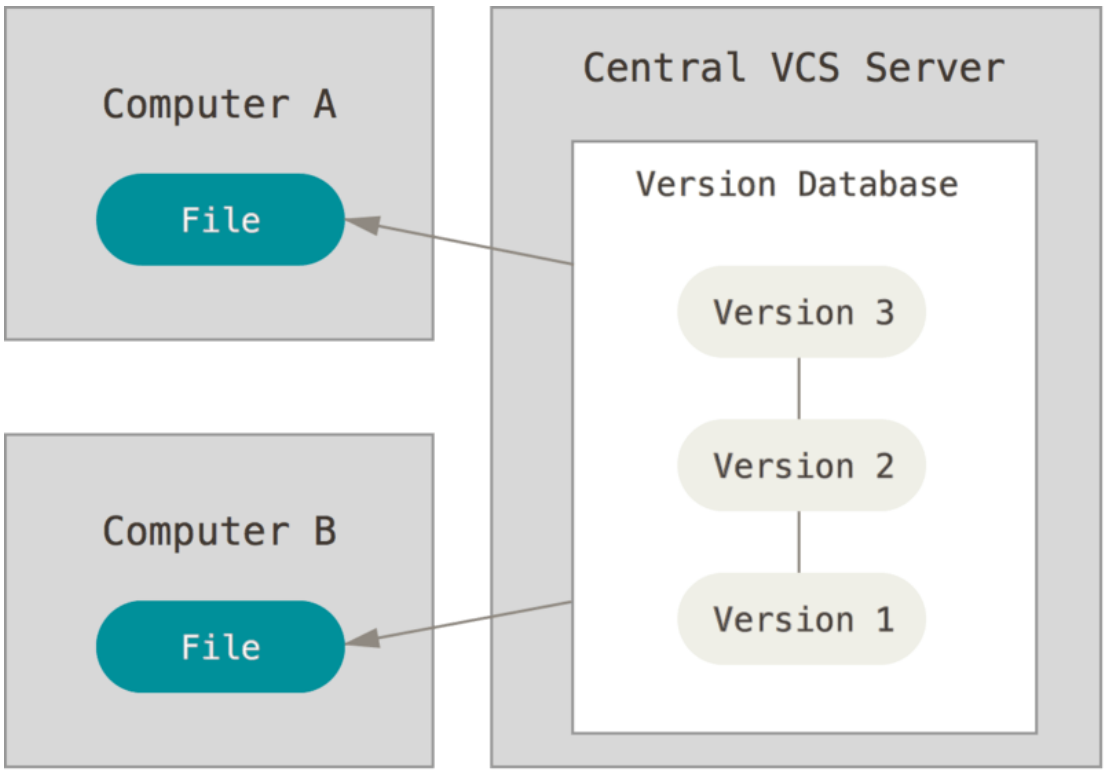


Рисунок 2 – Централізовані системи контролю версій.

Такий підхід має безліч переваг, особливо над локальними СКВ. Наприклад, кожному учаснику проекту відомо, певною мірою, чим займаються інші. Адміністратори мають повний контроль над тим, хто і що може робити. Набагато легше адмініструвати ЦСКВ, ніж мати справу з локальними базами даних для кожного клієнта.

Але цей підхід також має деякі серйозні недоліки. Найбільш очевидним є єдина точка відмови, яким є централізований сервер. Якщо сервер виходить з ладу протягом години, то протягом цієї години ніхто не може співпрацювати або зберігати зміни над якими вони працюють під версійним контролем. Якщо жорсткий диск центральної бази даних на сервері пошкоджено, і своєчасні резервні копії не були зроблені, ви втрачаєте абсолютно все — всю історію проекту, крім одиночних знімків проекту, що збереглися на локальних машинах людей. Локальні СКВ страждають тією ж проблемою — щоразу, коли вся історія проекту зберігається в одному місці, ви ризикуєте втратити все.

Децентралізовані системи контролю версій

Долучаються до гри децентралізовані системи контролю версій (ДСКВ). В ДСКВ (таких як, Git, Mercurial, Bazaar або Darcs), клієнти не просто отримують останній знімок файлів репозиторію: натомість вони є повною копією сховища разом з усією його історією. Таким чином, якщо вмирає який-небудь сервер, через який співпрацюють розробники, будь-який з клієнтських репозиторіїв може бути скопійований назад до серверу, щоб відновити його. Кожна копія дійсно є повною резервною копією всіх даних.

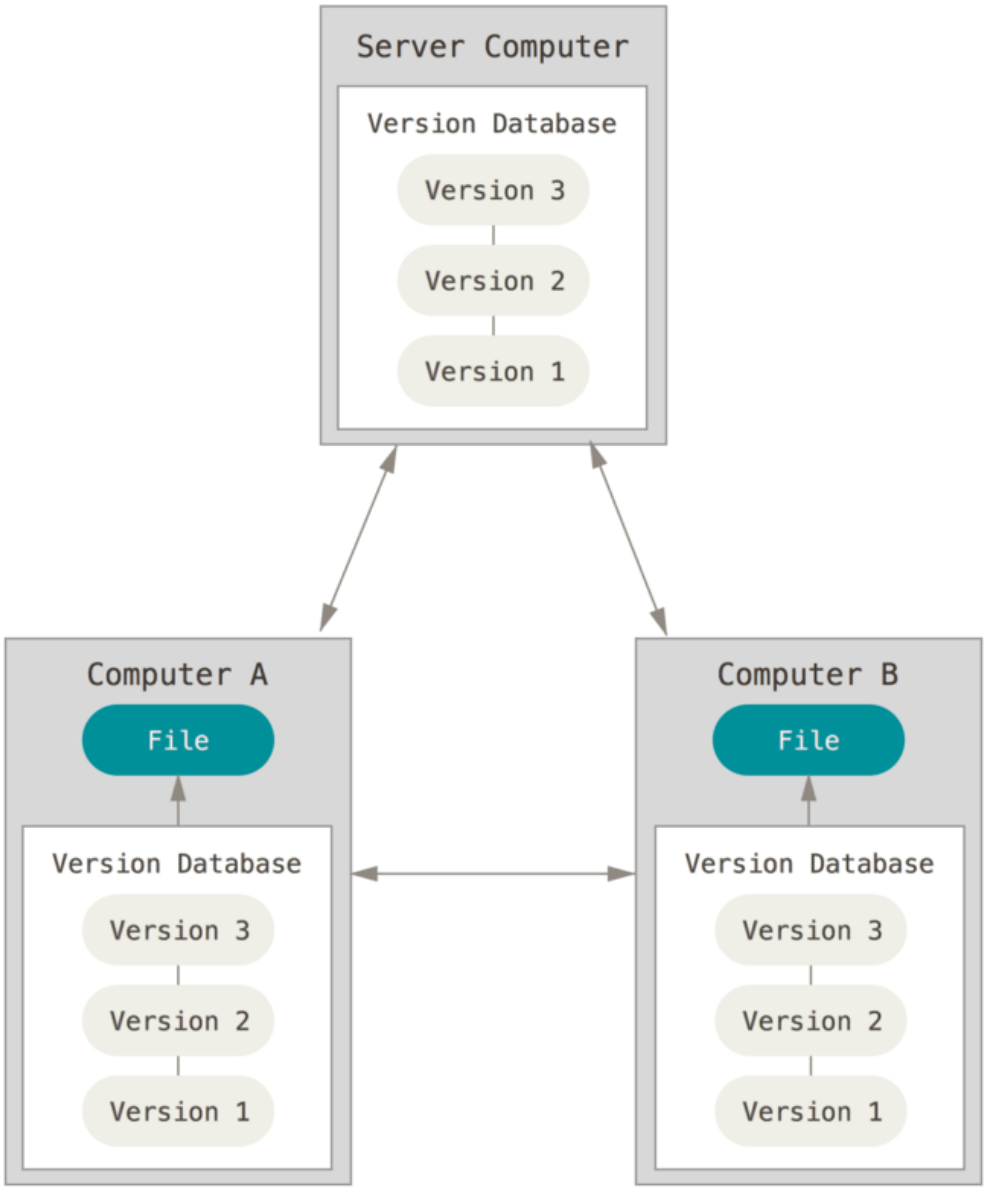


Рисунок 3 – Децентралізовані системи контролю версій

Більш того, багато з цих систем дуже добре взаємодіють з декількома віддаленими репозиторіями, так що ви можете співпрацювати з різними групами людей, застосовуючи різні підходи в межах одного проекту одночасно. Це дозволяє налаштувати декілька типів робочих процесів, таких як ієрархічні моделі, які неможливі в централізованих системах.

2 Хід роботи

1. Створити обліковий запис в системі GitHub (допустимо BitBucket, SourceForge або якійсь інші альтернативі https://itsfoss.com/github-alternatives/ , однак дана робота орієнтована на GitHub).
2. Інсталювати Git, керуючись інструкціями з https://git-

scm.com/book/uk/v2/%D0%92%D1%81%D1%82%D1%83%D0%BF-%D0%86%D0%BD%D1%81%D1%82%D0%B0%D0%BB%D1%8F%D1%86%D1%96%D1%8F-Git

3. Налаштувати свої облікові дані у системі https://git-scm.com/book/uk/v2/%D0%92%D1%81%D1%82%D1%83%D0%BF-%D0%9F%D0%BE%D1%87%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D0%B5-%D0%BD%D0%B0%D0%BB%D0%B0%D1%88%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-Git

4. Створити у GitHub відкритий репозиторій та додати до нього ліцензію для розповсюдження матеріалів з переліку, який випадає (на свій розсуд).

5. Використовуючи базові команди git http://guides.beanstalkapp.com/version-control/common-git-commands.html виконати такі дії:

5.1. Клонувати віддалений репозиторій на локальний комп'ютер.

5.2. Створити гілку в локальному репозиторії з відомостями про те, що саме Ви плануєте додавати. Аналогічно тому, як була створена гілка https://github.com/YuriiBaryshev/CyberSecurityHandbook/tree/git_init

5.3. Додати файл README.md до репозиторію (add, commit).

5.4. Змінити файл README.md відповідно до семантики мови Markdown https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

5.5. Зберегти зміни у репозиторії.

5.6. Завантажити отриманий результат на віддалену гілку (push)

5.7. Через графічний інтерфейс GitHub створити запит на злиття гілок (pull request) та призначити себе як відповідального виконавця.

5.8. Виконати автоматичне злиття нової гілки в основну (main або master), та видалити нову гілку у віддаленому репозиторії.

5.9. Скачати отримані зміни у локальний репозиторій (fetch та pull)

5.10. Створити у локальному репозиторію нову гілку (branch).

5.11. Перемкнутись на нову гілку (checkout)

5.12. Внести нові зміни у файл в новій гілці та зафіксувати ці зміни.

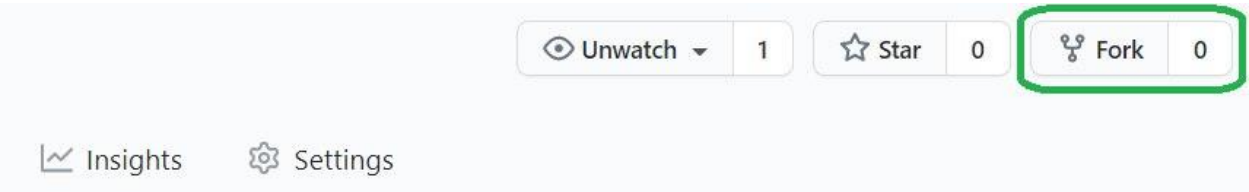
5.13. Злити у локальному репозиторії (merge) нову гілку в основну гілку.

6. Внести отримані зміни у віддалений репозиторій (push).

7. Внести зміни в проект іншого користувача. За бажанням, можна заразом зробити гарну справу і взяти участь у створенні універсального довідника для допомоги кіберзахисникам всього світу у їх нелегкій справі, зробивши форк репозиторію. https://github.com/YuriiBaryshev/CyberSecurityHandbook та розширити наявний там перелік команд git, або розпочати власну тематику (головне розподілити між собою завдання, щоб не почати розкривати одну тематику двічі).

7.1. Визначитись з репозиторієм, до якого плануєте внести зміни (репозиторій одногрупника або https://github.com/YuriiBaryshev/CyberSecurityHandbook) та зробити

форк репозиторію



7.2. Внести зміни аналогічно виконаним у кроці 6.

7.3.Надіслати запит на злиття відгалуженого репозиторію до "основного".

8. Зробити висновки, в яких зокрема відзначити враження щодо використання можливостей git для захисту цілісності даних та роботи в командах.

Джерела дослідження :

1. S. Chacon, B. Straub. Pro Git book. https://git-scm.com/book/uk/v2
2. Git шпаргалка. http://pr0git.blogspot.com/p/git.html
3. Git за полчаса: руководство для начинающих. https://proglib.io/p/git-for-half-an-hour
4. Git на практике. https://habr.com/ru/post/342116/