**Лабораторна робота №1**

# **Тема:** системи керування версіями та їх призначення в WEB-програмуванні.

# **План роботи:**

* розглянути, що таке системи керування версіями;
* ознайомитись з існуючими системи;
* навчитися працювати з GIT репозиторіями (наприклад github, bitbucket, gitlab).

# **Теоретичні відомості:**

Система керування версіями - програмний інструмент для керування версіями одиниці інформації: початкового коду програми, скрипту, WEB-сторінки чи сайту, 3D-моделі, текстового документу тощо.

Система керування версіями — інструмент, який дозволяє одночасно, не заважаючи один одному, проводити роботу над груповими проектами.

Системи керування версіями зазвичай використовуються при розробці програмного забезпечення для відстеження, документування та контролю над поступовими змінами в електронних документах: у сирцевому коді застосунків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють декілька людей.

**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.



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.



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.

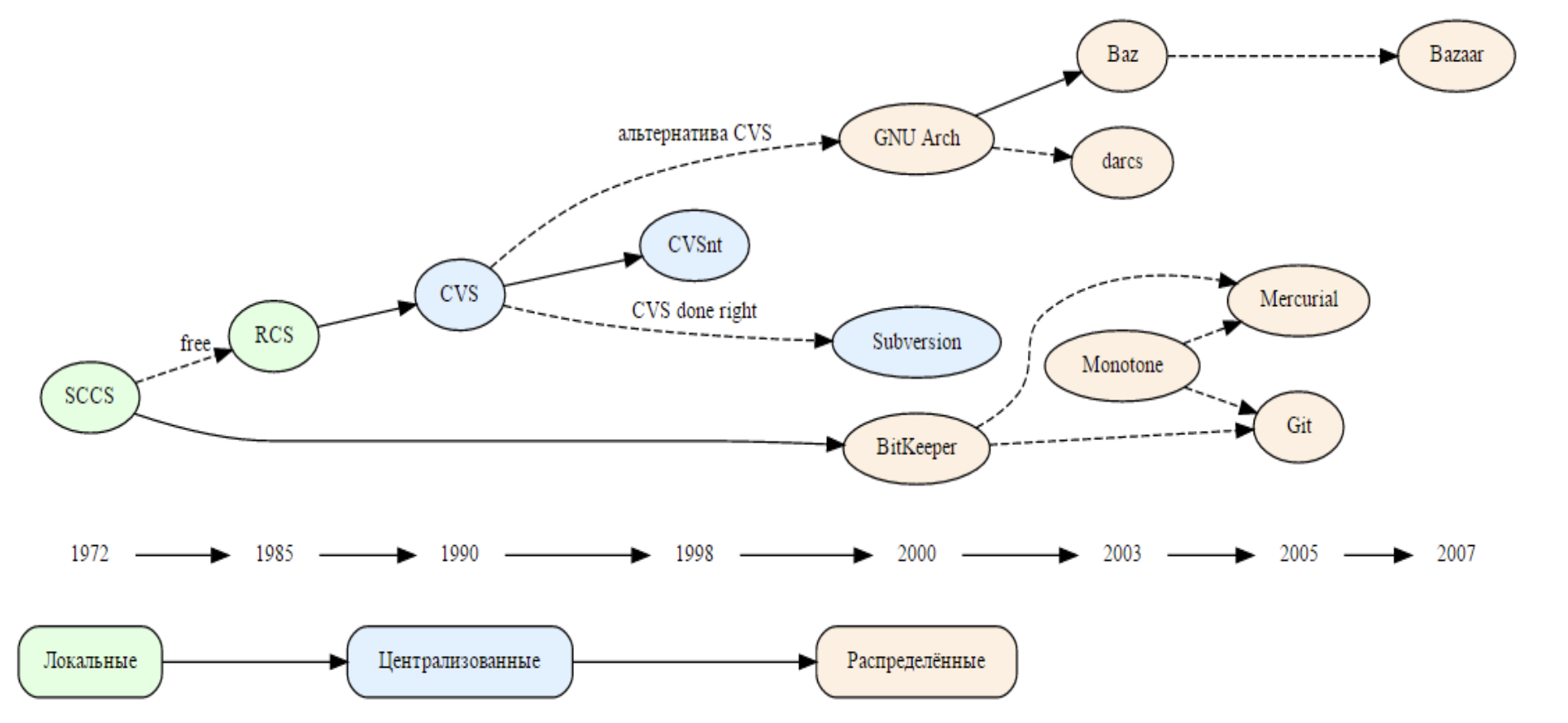
Але цей підхід також має деякі серйозні недоліки. Найбільш очевидним є єдина точка відмови, яким є централізований сервер. Якщо сервер виходить з ладу протягом години, то протягом цієї години ніхто не може співпрацювати або зберігати зміни над якими вони працюють під версійним контролем. Якщо жорсткий 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 (розподілена) 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 репозиторія: 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.



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.

**Популярні системи контроля версій.**



**CVS** є найпопулярнішою і широко застосовуваною системою контролю версій на сьогоднішній день. Після випуску в 1986 році вона швидко стала загальноприйнятим стандартом. CVS придбала популярність завдяки простій системі підтримки файлів і ревізій в актуальному стані.

**Переваги:**

* Це перевірена часом система, яка використовується більше трьох десятиліть;
* Існує багато IDE, які використовують CVS.

**Недоліки:**

* Переміщення або перейменування файлів не включається в оновлення версії;
* Надання символічних посилань на файли пов'язано з деякими ризиками безпеки;
* Відсутність підтримки атомарних операцій може привести до пошкодження вихідного коду;
* Повільні операції встановлення міток і розгалуження;
* Слабка підтримка довічних файлів.

**SVN.** Ще одна поширена система управління версіями. Більшість проектів з відкритим вихідним кодом і великі платформи, такі як Ruby, Python Apache, використовують SVN. Через величезної популярності існує безліч версій і доступних IDE.

**Переваги:**

* Нова і значно поліпшена система, заснована на CVS;
* Допускає атомарні операції;
* Операції в гілці проекту малозатратних;
* Доступні різні плагіни IDE.

**Недоліки:**

* Видає помилки при перейменуванні файлів і каталогів;
* Недостатньо команд для управління репозиторієм;
* SVN працює повільніше в порівнянні з іншими системами управління версіями.

**GIT.** Завдяки розподіленої формі управління без необхідності використання оригінального програмного забезпечення багато проектів з відкритим вихідним кодом надають перевагу GIT.

**Переваги:**

* Майже всі негативні риси CVS / SVN усунені;
* Висока швидкість роботи розподіленої системи контролю версій;
* Легкість проведення різних операцій з гілками проекту;
* Користувачі можуть отримати доступ до повного дереву історії в режимі офлайн;
* Пропонує високо розподілену однорангову модель.

**Недоліки:**

* Високий поріг входження для користувачів SVN;
* Обмежена підтримка Windows в порівнянні з Linux.

**Mercurial.** Вважається ефективної для великих проектів, в яких бере участь багато розробників і дизайнерів. Mercurial - це високопродуктивна система, яка пропонує оптимальну швидкість. Вона також відома своєю простотою і докладною документацією.

**Переваги:**

* Низький поріг входження в порівнянні з Git;
* Детальна документація;
* Розподілена модель;
* Високопродуктивна система з відмінною швидкістю.

**Недоліки:**

* Не можна об'єднати дві батьківські гілки;
* Заснована на розширеннях, а не сценаріях;
* Недостатньо гнучка, щоб виконувати операції за замовчуванням.

**Робота с GIT.**

Git - розподілена система керування версіями файлів та спільної роботи. Проєкт створив Лінус Торвальдс для керування розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано (англ. Junio C. Hamano). Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів.

Прикладами проєктів, що використовують Git, є ядро Linux, Android, LibreOffice, Cairo, GNU Core Utilities, Mesa 3D, Wine, багато проєктів з X.org, XMMS2, GStreamer, Debian DragonFly BSD, Perl, Eclipse, GNOME, KDE, Qt, Ruby on Rails, PostgreSQL, VideoLAN, PHP, One Laptop Per Child (OLPC), АБІС Koha, GNU LilyPond та ELinks і деякі дистрибутиви GNU/Linux.

Репозиторій Git - це каталог файлової системи, в якому знаходяться файли конфігурації сховища, файли журналів, що зберігають операції, що виконуються над репозиторієм, індекс, що описує розташування файлів, і сховище, що містить власне файли. Структура сховища файлів не відображає реальну структуру репозиторія файлового дерева, вона орієнтована на підвищення швидкості виконання операцій з репозиторієм. Коли ядро ​​обробляє команду зміни (неважливо, при локальних змінах або при отриманні патча від іншого вузла), воно створює в сховище нові файли, що відповідають новим станам змінених файлів. Ніякі операції не змінюють вмісту вже існуючих в сховище файлів.

Для створення Git-сховища ви можете використовувати два основні підходи. По-перше, імпорт Git вже існуючого проекту або директорії. По-друге, клонування існуючого сховища з іншого сервера.

В обох випадках ви отримаєте готовий до роботи Git репозиторій на вашому комп'ютері.

**Встановлення програмного забезпечення GIT.**

<https://git-scm.com/downloads>

**Створення сховища в існуючій директорії.**

**Для Linux:**

cd /home/user/my\_project

**Для macOS:**

cd /Users/user/my\_project

**для Windows:**

cd C:/Users/user/my\_project

**А потім виконайте команду:**

git init

Ця команда створює в поточному каталозі нову підпапку з ім'ям .git, що містить всі необхідні файли сховища - структуру Git-сховища. На цьому етапі ваш проект ще не знаходиться під версійність контролем.

Якщо ви хочете додати під версійний контроль існуючі файли (на відміну від порожнього каталогу), вам варто додати їх в індекс і здійснити перший комміт змін. Досягти цього ви зможете запустивши команду **git add** кілька разів, вказавши індексовані файли, а потім виконавши git commit:

git add name\_of\_file.txt

git add LICENSE

git commit -m 'Init project'

**Клонування існуючого сховища.**

Для отримання копії існуючого Git-сховища, наприклад, проекту, в який ви хочете внести свій внесок, необхідно використовувати команду **git clone**. При виконанні git clone з сервера забирається (pulled) кожна версія кожного файлу з історії проекту. Фактично, якщо серверний диск вийде з ладу, ви можете використовувати будь-який з клонів на будь-якому з клієнтів, для того, щоб повернути сервер в той стан, в якому він знаходився в момент клонування (ви можете втратити частину серверних хуков (server-side hooks) і т.п., але всі дані, поміщені під версійність контроль, будуть збережені.

Клонування сховища здійснюється командою **git clone <url>**. Наприклад, якщо ви хочете клонувати бібліотеку libgit2, ви можете зробити це в такий спосіб:

git clone <https://github.com/libgit2/libgit2>

**Розгалуження в Git.**

Майже кожна система контролю версій (СКВ) в якійсь формі підтримує розгалуження. Використовуючи розгалуження, ви відхиляєтеся від основної лінії розробки і продовжуєте роботу незалежно від неї, не втручаючись в основну лінію. У багатьох СКВ створення гілок - це дуже витратний процес, часто вимагає створення нової копії директорії, що може зайняти багато часу для великого проекту.

Розгалуження Git дуже простим: операція створення гілки виконується майже миттєво, перемикання між гілками, як правило, також швидко. На відміну від багатьох інших СКВ, Git заохочує процес роботи, при якому розгалуження і злиття виконується часто, навіть по кілька разів на день. Розуміння і володіння цією функціональністю дає вам унікальний і потужний інструмент, який може повністю змінити звичний процес розробки.

Гілка в Git - це простий переміщуваний вказівник на один з таких коммітов. За замовчуванням, ім'я основної гілки в Git - master. Як тільки ви почнете створювати комміти, гілка master буде завжди вказувати на останній комміт. Кожен раз при створенні коммітов покажчик гілки master буде пересуватися на наступний Комміт автоматично.

**Збереження даних в GIT.**

Для того, щоб зберегти дані в репозиторій потрібно виконати наступні команди:

git add name\_of\_file.txt

git commit -m 'Init project'

git push origin master

**Віддалені репозиторії GIT.**

**GitHub** — один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome).

Сервіс безкоштовний для проєктів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL), а для окремих індивідуальних проєктів пропонуються різні платні тарифні плани.

**Bitbucket** — веб-сервіс для хостингу проектів на базі систем керування версіями: Mercurial та Git (з жовтня 2011). Bitbucket надає як безкоштовні так і платні послуги. Є аналогом GitHub, однак, на відміну від GitHub, який до січня 2019 року зберігав файли безкоштовних профілів лише у відкритому доступі, Bitbucket від самого початку дозволяв безкоштовно створювати приватні репозиторії з можливістю спільної роботи з файлами до 5-ти користувачів. Bitbucket інтегрований з іншими програмними продуктами Atlassian, такими як, JIRA, Confluence, Bamboo та HipChat.

**GitLab** - компанія, що пропонує схожі з GitHub користувацькі послуги із додатковими перевагами, як то приватні репозиторії для безкоштовних підписників. Також суттєвою перевагою є можливість розгорнути систему на сторонніх серверах.

# **Завдання лабораторної роботи:**

1. Створити публічний GIT репозиторій в одному з наведених вище сервісів (github, bitbucket, gitlab);
2. Ініціалізувати даний GIT репозиторій на власному комп’ютері;
3. Додати файл lab1.txt до даного репозиторію та зберегти зміні на віддаленому сервері.

# **Список літератури:**

* <https://git-scm.com/book/uk/v2>
* <https://en.wikipedia.org/wiki/Version_control_system>
* <https://github.com/>
* <https://bitbucket.org/>
* <https://gitlab.com/>