**Лабораторна робота №1**

з курсу «**Проектування інформаційних систем**»

на тему: «**Системи контролю версій SVN,GIT**»

Виконала:

студентка групи ДА-61

Онищенко Єлизавета

**Мета роботи:** за допомогою системи контролю версій завантажити коди програми у репозіторій. Відтворити типовий цикл розробки програмного забезпечення з використанням системи контролю версій.

**Задача:**

1. Вивчити основні команди роботи з репозіторіями.

2. Завантажити код програми у репозіторій.

3. Показати основний цикл роботи з програмним кодом за допомогою системи

контролю версій.

**Короткі 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 пізнише.

Основною відмінністю від інших систем (таких як Subversion та подібних їй) є те, як Git сприймає дані. Концептуально, більшість СКВ зберігають інформацію як список файлових редагувань. Ці інші системи (CVS, Subversion, Perforce, Bazaar тощо) розглядають інформацію як список файлів та змін кожного з них протягом деякого часу (це зазвичай називають **оснований на дельтах** контроль версій).

Git не оброблює та не зберігає свої дані таким чином. Замість цього, Git сприймає свої дані радше як низку знімків мініатюрної файлової системи. У Git щоразу, як ви створюєте коміт, тобто зберігаєте стан вашого проекту, Git запам’ятовує як виглядають всі ваші файли в той момент і зберігає посилання на цей знімок. Для ефективності, якщо файли не змінилися, Git не зберігає файли знову, просто робить посилання на попередній ідентичний файл, котрий вже зберігається. Git вважає свої дані більш як **потік знімків**.

Більшість операцій у Git потребують лише локальних файлів та ресурсів для здійснення операцій — немає необхідності в інформації з інших комп’ютерів вашої мережі. Якщо ви звикли до ЦСКВ, де більшість операцій обтяжені такими мережевими запитами, то цей аспект може привести вас до думки, що боги швидкості наділили Git неземною силою. Через те, що повна історія проекту знаходиться на вашому локальному диску, більшість операцій здійснюються майже миттєво.

Будь-що в Git, перед збереженням, отримує контрольну суму, за якою потім і можна на нього посилатися. Таким чином, неможливо змінити файл чи директорію так, щоб Git про це не дізнався. Цей функціонал вбудовано в систему на найнижчих рівнях і є невід’ємною частиною її філософії. Ви не можете втратити інформацію при передачі чи отримати пошкоджений файл без відома Git.

Механізм, який використовується для цього контролю, називається хеш SHA-1. Він являє собою 40-символьну послідовність цифр та перших літер латинського алфавіту (a-f) і вираховується на основі вмісту файлу чи структури директорії в Git. SHA-1 хеш виглядає це приблизно так:

24b9da6552252987aa493b52f8696cd6d3b00373

При роботі з Git ви повсюди зустрічатимете такі хеші, адже Git постійно їх використовує. Фактично, Git зберігає все не за назвою файлу, а саме за значенням хешу його змісту.

**Створення Git-репозиторія**

Зазвичай Git репозиторій отримують одним з двох способів:

1. Беруть локальну директорію, що наразі не під контролем версій, та перетворюють її на сховище Git, або
2. Звідкілясь **клонують** існуючий Git репозиторій.

У будь-якому разі ви отримуєте на локальній машині готове до роботи Git сховище.

### **Клонування існуючого репозиторія**

Якщо ви бажаєте отримати копію існуючого Git репозиторія — наприклад, проекту, в якому ви хочете прийняти участь — вам потрібна команда git clone. Якщо ви знайомі з іншими СКВ, наприклад Subversion, ви помітите, що команда називається "clone" (клонувати), а не "checkout" (перевірити).

## Запис змін до репозиторія

Пам’ятайте, що кожен файл вашої робочої директорії може бути в одному з двох станів: **контрольований** (tracked) чи **неконтрольований** (untracked). Контрольовані файли — це файли, що були в останньому знімку. Вони можуть бути не зміненими, зміненими або індексованими. Якщо стисло, контрольовані файли — це файли, про які Git щось знає.

Неконтрольовані файли — це все інше, будь-які файли у вашій робочій директорії, що не були у вашому останньому знімку та не існують у вашому індексі. Якщо ви щойно зробили клон репозиторія, усі ваші файли контрольовані та не змінені, адже Git щойно їх отримав, а ви нічого не редагували.

По мірі редагування файлів, Git бачить, що вони змінені, адже ви їх змінили після останнього коміту. Впродовж роботи ви вибірково індексуєте ці змінені фали та потім зберігаєте всі індексовані зміни, та цей цикл повторюється.

### Перевірка статусу ваших файлів

Щоб дізнатись, в якому стані ваші файли, варто скористатись командою git status.

### Контролювання нових файлів

Щоб почати контролювати новий файл, вам треба використати команду git add.

### Ігнорування файлів

Буває, що у вас є клас файлів, що ви не хочете щоб Git їх автоматично індексував чи навіть відображав як неконтрольовані. Зазвичай це автоматично згенеровані файли, наприклад файли лоґів або файли вироблені вашою системою збірки. У таких випадках, ви можете створити файл .gitignore, що містить взірці, яким відповідають ці файли.

### Видаляємо файли

Щоб видалити файл з Git, вам треба прибрати його з контрольованих файлів (вірніше, видалити його з вашого індексу) та створити коміт. Команда git rm це робить, а також видаляє файл з вашої робочої директорії, щоб наступного разу він не відображався неконтрольованим.

## Перегляд історії комітів

Після того як ви створили декілька комітів, або якщо ви зробили клон репозиторія з існуючою історією комітів, ви напевно забажаєте дізнатись, що було відбувалося. Найбільш могутньою утилітою для цього є команда git log.

## Взаємодія з віддаленими сховищами

Задля співпраці з будь-яким проектом Git, вам необхідно знати, як керувати віддаленими сховищами. Віддалені сховища — це версії вашого проекту, що розташовані в Інтернеті, або десь у мережі. Їх може бути декілька, кожне зазвичай або тільки для читання, або для читання та змін. Співпраця з іншими вимагає керування цими віддаленими сховищами, надсилання (pushing) та отримання (pulling) даних до та з них, коли ви хочете зробити внесок.

### Дивимось на ваші сховища

Щоб побачити, які віддалені сервера ви налаштували, ви можете виконати команду git remote. Вона виводить список коротких імен кожного віддаленого сховища, яке ви задали. Якщо ви отримали своє сховище клонуванням, ви маєте побачити хоча б origin — таке ім’я Git дає серверу, з якого ви зробили клон

### Додавання віддалених сховищ

Ми згадували і навіть продемонстрували, як команда git clone неявно додає віддалене сховище origin.

### Отримання (fetching) та затягування (pulling) з ваших віддалених сховищ

Як ви щойно побачили, щоб отримати дані з ваших віддалених проектів, ви можете виконати:

$ git fetch <remote>

Ця команда заходить на віддалений проект та забирає звідти усі дані, котрих у вас досі нема. Після цього, у вас будуть посилання на всі гілки з того сховища, які ви можете зливати або оглядати в будь-який час.

### Надсилання змін до ваших віддалених сховищ

Коли ви довели свій проект до стану, коли хочете ним поділитись, вам треба надіслати (push) ваші зміни нагору (upstream). Це робиться простою командою: git push <назва сховища> <назва гілки>. Якщо ви бажаєте викласти свою гілку master до вашого серверу origin (клонування зазвичай налаштовує обидва імені для вас автоматично), ви можете виконати наступне для надсилання всіх зроблених комітів до сервера:

$ git push origin master

### Оглядання віддаленого сховища

Якщо ви бажаєте більше дізнатись про окреме віддалене сховище, ви можете використати команду git remote show <назва сховища>

### Перейменування та видалення віддалених сховищ

Ви можете виконати git remote rename, щоб перейменувати віддалене сховище.

### Створення нової гілки

Що відбувається, якщо ви створюєте нову гілку? Ну, це створює новий вказівник, щоб ви могли пересуватися. Припустімо, ви створюєте нову гілку під назвою testing. Ви це робите за допомогою команди git branch:

$ git branch testing

### Переключення гілок

Щоб переключитися на існуючу гілку, треба виконати команду git checkout. Переключімося на нову гілку testing:

$ git checkout testing

### Основи зливання

Допустимо, ви вирішили, що робота над №53 завершена і готова до злиття з гілкою master. Для цього ви виконаєте злиття гілки iss53 до master саме так, як раніше робили це з гілкою hotfix. Все що потрібно це перемкнутися на вашу робочу гілку і виконати команду git merge

## Управління гілками

Тепер, коли ви вже вмієте створювати гілки, зливати їх та видаляти, розгляньмо те, як ними управляти, та на інструменти, які можуть в цьому допомогти.

Команда git branch насправді вміє більше ніж просто створювати та знищувати гілки. Запустіть її без параметрів і ви побачите просто список ваших гілок:

### Довго-триваючі гілки

Оскільки Git використовує просте триточкове злиття, то зливання одної гілки в іншу протягом тривалого часу зазвичай не є складною задачею. Це означає, що ви можете мати кілька активних гілок та використовувати їх для різних стадій вашого робочого циклу; можете періодично зливати з одної гілки в іншу.

Багато розробників підтримують з Git такий процес, коли тільки в master є стабільна версія коду — найімовірніше того, що був чи буде запроваджений у виробництво. Також вони мають паралельні гілки develop чи next, які використовуються для тестування стабільності — це не обов’язково постійно стабільні гілки, але, як тільки вони стабілізовуються, їх можна зливати з master. Також практикується мати тематичні гілки (коротко-термінові, як iss53, що ви створили раніше) та зливати їх, коли вони готові, проходять всі тести, та не привнесуть нових помилок.

## Віддалені гілки

Віддалені посилання — це посилання (вказівники) у ваших віддалених сховищах: гілки, теґи тощо. Для повного списку віддалених посилань виконайте git ls-remote [remote], або git remote show [remote] для детальної інформації про віддалені гілки. Проте, найпоширеніше застосування — це віддалено-відслідковувані гілки.

Віддалено-відслідковувані гілки — це вказівники на стан віддалених гілок. Локально ці вказівники неможливо змінити, але їх змінює Git, коли ви виконуєте мережеві операції, щоб вони точно відповідали стану віддаленого сховища. Вважайте їх закладками, що нагадують вам про стан віддалених репозиторіїв на момент вашого останнього зв’язку з ними.

### Відслідковувані гілки

Перемикання на локальну гілку з віддалено-відслідковуваної автоматично створює так звану відслідковувану гілку “tracking branch” (а гілка, за якою вона стежить називається “upstream branch”). Відслідковувані гілки — це локальні гілки, що мають безпосередній зв’язок з віддаленою гілкою. Якщо ви знаходитеся на відслідковуваній гілці і потягнете зміни, виконуючи git pull, Git відразу знатиме з якого сервера брати та з якої гілки зливати зміни.

### Витягування змін

Команда git fetch, під час виконання, отримує всі оновлення, яких ви ще не маєте, але, зовсім не змінює вашу робочу директорію. Вона просто отримує дані для того, щоб ви могли самотужки злити зміни. Існує команда git pull, яка, по своїй суті та в більшості випадків, є послідовним виконанням команд git fetch та git merge. Якщо у вас є відслідковувана гілка, як ми показували у попередній секції, — створена та самостійно налаштована, чи як результат clone чи checkout, команда git pull буде звертатися до відслідковуваних сервера та віддаленої гілки, отримувати оновлення і тоді робити спробу зливання.

### Видаляння віддалених гілок

Уявіть, що ви закінчили з віддаленою гілкою — тобто всі співпрацівники завершили свій вклад у нову функціональність та гілку було злито з віддаленою master (чи яка там у вас стабільна лінія коду). Для видалення віддаленої гілки користуйтеся опцією --delete до git push. Щоб видалити вашу serverfix виконайте таке:

$ git push origin --delete serverfix

Основні команди:

**Створення нового репозиторія - git init**

**Перевірка статусу – git status**

**Переміщення - git add**

**Внесення коміту – git commit**

**Підключення до дистанційного репозиторія – git remote add**

**Завантаження но сервер – git push**

**Клонування репозиторія – git clone**

**Отримання змін з серверу – git pull**

**Створення нової гілки – git branch**

**Перехід між гілками – git checkout**

**З'єднання гілок – git merge**

**Висновок:**

В даній лабораторній роботі було досліджено можливості GIT. В результаті виконання роботи було завантажено коди програми у репозиторій за допомогою системи контролю версій.