ЛАБОРАТОРНА РОБОТА 1.1
КЕРУВАННЯ ВЕРСІЯМИ ЗА ДОПОМОГОЮ GIT

Мета роботи: вивчити та закріпити на практиці можливості системи керування версіями, одержати практичний досвід у використанні Git та GitHub.

Обладнання:

– ПК IBM PC x86 CPU з встановленою операційною системою;

– доступ до мережі інтернет.

1.1.1 Теоретичні відомості

Система керування версіями (англ. *source code management*, SCM) — програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D моделі, текстового документу тощо.

Система керування версіями — це потужний інструмент, який дозволяє одночасно, без перешкод 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 коді додатків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють декілька людей.

Кожна версія позначається унікальною цифрою чи літерою, зміни документу занотовуються. Зазвичай також зберігається автор зробленої зміни та її час.

Інструменти для контролю версій входять до складу багатьох інтегрованих середовищ розробки.

Системи керування версіями існують двох основних типів: з централізованим сховищем та розподіленим.

Централізовані системи контролю версії. Централізована система контролю версії (клієнт-серверна) — система, дані в якій зберігаються в єдиному «серверному» сховищі. Весь обмін файлами відбувається з використанням центрального сервера. Є

можливість створення та роботи з локальними репозиторіями (робочими копіями).

Переваги:

– загальна нумерація версій;
– дані знаходяться на одному сервері;
– можлива реалізація функції блокування файлів;
– можливість керування доступом до файлів.

Недоліки:

– необхідність мережевого з'єднання для оновлення робочої копії чи збереження змін.

До таких систем відносять Subversion, Team Foundation Server.

Розподілені системи контролю версії. Розподілена система контролю версії (англ. Distributed Version Control System, DVCS) — система, яка використовує замість моделі клієнт-сервер, розподілену модель зберігання файлів. Така система не потребує сервера, адже всі файли знаходяться на кожному з комп'ютерів.

Переваги:

– кожний з розробників працює зі своїм власним репозиторієм;
– рішення щодо злиття віток приймається керівником проекту;
– немає потреби в мережевому з'єднанні.

Недоліки:

– немає можливості контролю доступу до файлів;
– відсутня загальна нумерація версії файла;
– значно більша кількість необхідного дискового простору;
– немає можливості блокування файлів.

До таких систем відносять Git, Mercurial, SVK, Monotone, Codeville, BitKeeper.

Система контролю дозволяє зберігати попередні версії файлів та завантажувати їх за потребою. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки. Місце зберігання даних файлів називають репозиторієм. В середині кожного з репозиторіїв можуть бути створені паралельні лінії розробки — гілки.

Гілки, зазвичай, використовують для зберігання експериментальних, незавершених (alpha, beta) та повністю

робочих версій проекту (final). Більшість систем контролю версії дозволяють кожному з об'єктів присвоювати теги, за допомогою яких можна формувати нові гілки та репозиторії.

Використання системи контролю версії є необхідним для роботи над великими проектами, над якими одночасно працює велика кількість розробників. Системи контролю версії надають ряд додаткових можливостей:

– можливість створення різних варіантів одного документу;

– документування всіх змін (коли і ким було змінено/додано, хто який рядок змінив);

– реалізує функцію контролю доступу користувачів до файлів. Є можливість його обмеження;

– дозволяє створювати документацію проекту з поетапним записом змін в залежності від версії;

– дозволяє давати пояснення до змін та документувати їх.

Словник основних термінів-сленгів:

– транк (trunk) — основна гілка коду;

– бранч (branch) — відгалуження;

– чекін (Check in (submit, commit)) — відправлення коду в репозиторій;

– чекаут (Check out) — одержання зміни з репозиторію;

– конфлікти — виникають, коли кілька людей правлять один і той же код, конфлікти можна вирішувати;

– патч — шматок з записаними змінами, які можна застосувати до сховища з кодом.

1.1.2 Порядок виконання роботи

Для керування версіями розробки програмного забезпечення обрано сервіс GitHub.

Переваги:

– простий спосіб реєстрації;

– адаптовані під розповсюдженні версії операційних систем клієнти;

– простий спосіб синхронізації між сервісом та локальною версією;

– підтримка багатьох мов програмування;

– можливість відновлення репозиторію після видалення.

Недоліки:

– для windows необхідний NET Framework 4.0;

– ручна синхронізація;

– для проекту що складається з декількох директорій, потрібно створити відповідно декілька репозиторіїв у системі та відслідковувати кожен з них;

– невірне відображення кирилиці після відновлення з віддаленого репозиторію.

Реєстрація у системі

1 Зайти на сайт https://github.com/.

2 Натиснути на кнопку SING UP FOR GitHub (див. рис. 1.1.1)

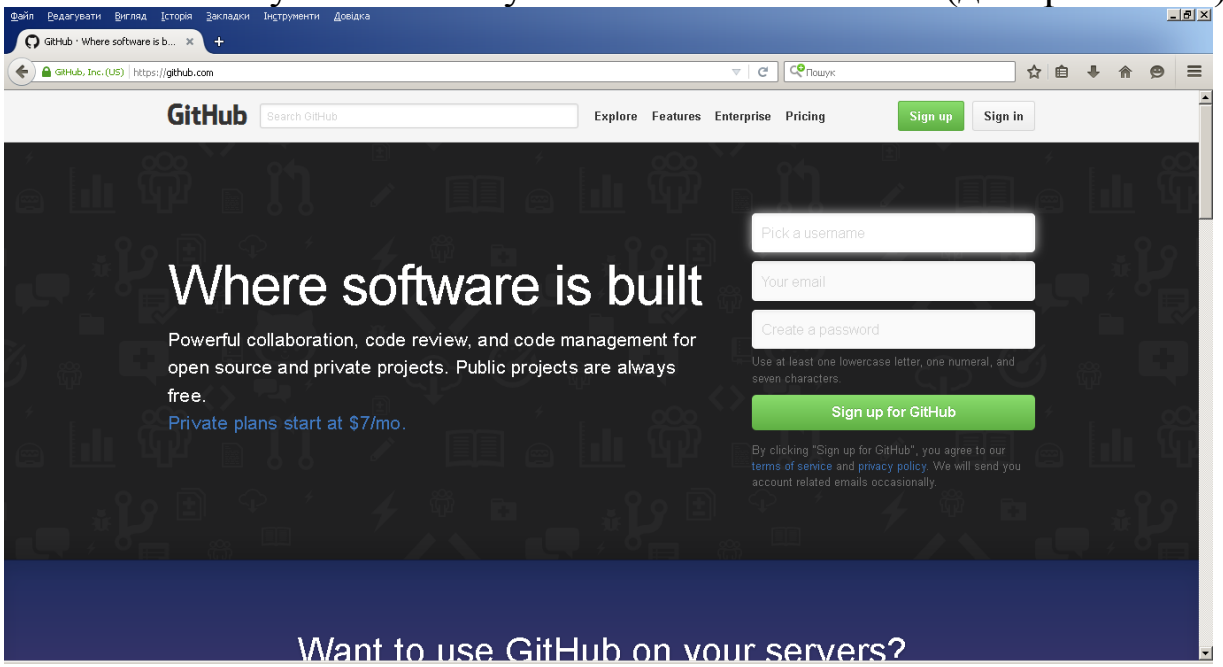

Рисунок 1.1.1 – Кнопка для реєстрації учасника в системі

3 Заповнити відповідні поля форми реєстрації (див. рис. 1.1.2)

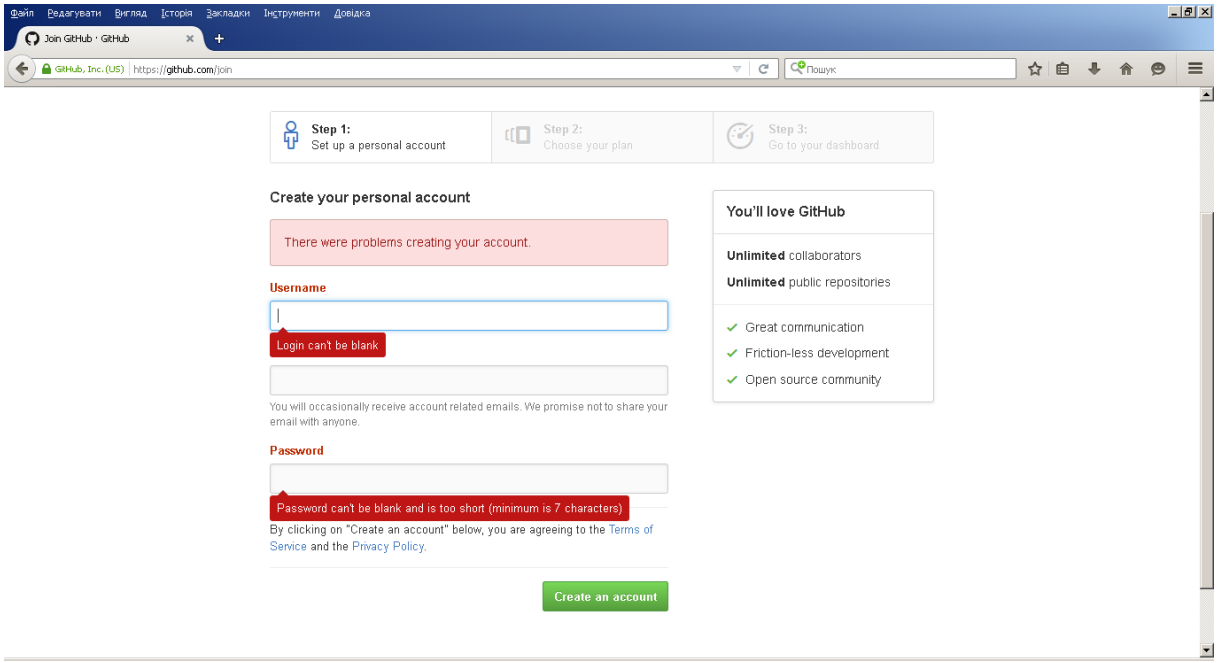

Рисунок 1.1.2 – Форма реєстрації учасника

4 Після реєстрації відкриється головна форма системи, де є можливість створити новий проект та завантажити клієнт відповідно до вашої операційної системи (див. рис. 1.1.3).

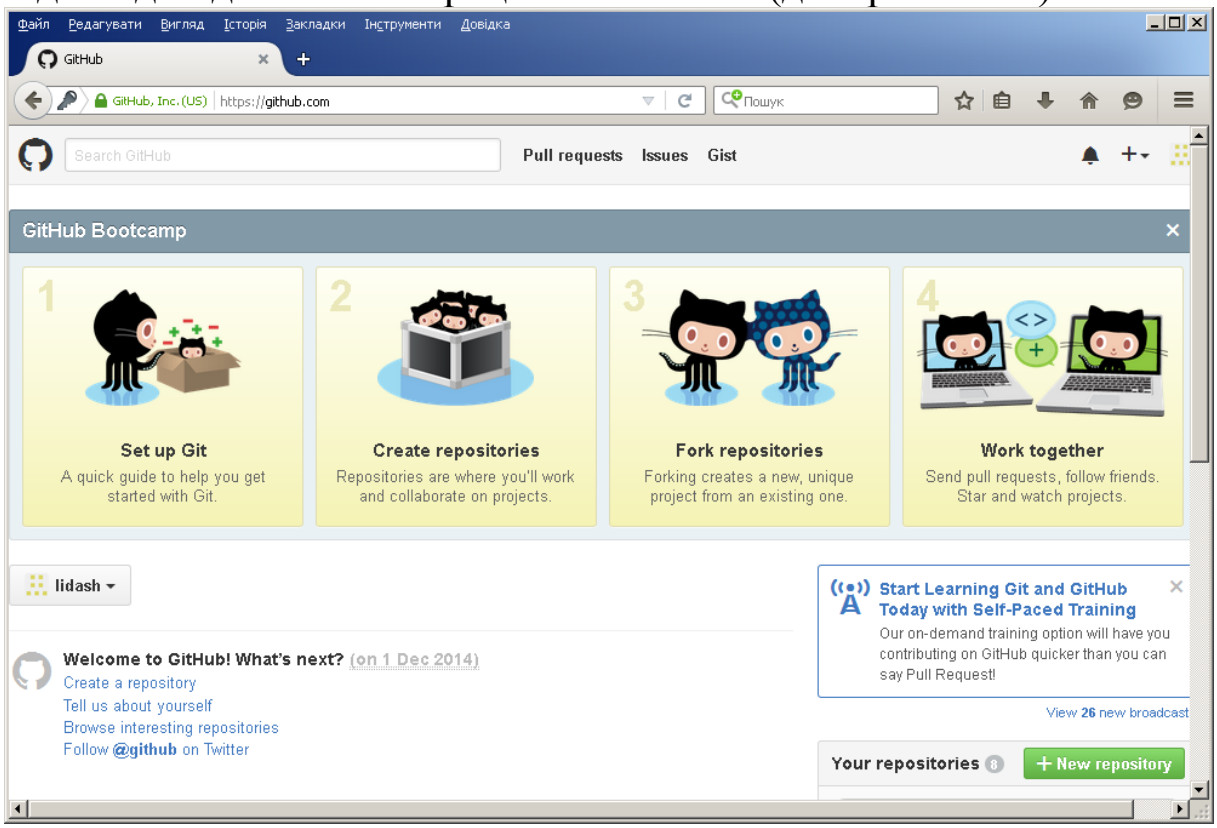

Рисунок 1.1.3 - Головна форма керування системою

5 Встановлення Git. Перед початком використання Git, необхідно встановити його на Вашому комп'ютері. Встановлення Git на Linux за допомогою бінарного пакету - через основний менеджер управління пакетами, що йде з вашим дистрибутивом (для Debian-подібного дистрибутиву, такого як Ubuntu):

```
$ sudo apt-get install git
```

Інсталяція на Windows: є декілька шляхів встановити Git під Windows. Найофіційніша збірка доступна для завантаження з сайту Git. Перейдіть до http://git-scm.com/download/win і завантаження почнеться автоматично.

6 Початкове налаштування Git. Налаштування потрібно виконати лише один раз (налаштування залишаються між оновленнями). Встановлення імені користувача та адреси електронної пошти (кожен коміт в Git використовує цю інформацію і вона включена у комміти):

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Перевірка налаштувань – виконати команду git config --list, щоб переглянути всі налаштування, котрі Git встановив.

7 Створення Git-репозіторія. Для створення Git репозиторію використовують два основних підходи. Перший приймає існуючий проект або каталог і імпортує його в Git. Другий клонує існуючий репозіторій Git з іншого сервера.

Ініціалізація репозіторія в існуючому каталозі: якщо ви починаєте використовувати Git для існуючого проекту, вам треба зайти в каталог проекту та виконати:

```
$ git init
```

Це створить новий підкаталог .git, який містить всі необхідні файли вашого репозіторія - скелет Git-репозиторія. На даний момент, у проекті ще нічого не відстежується.

Якщо необхідно додати існуючі файли під версійний контроль, необхідно проіндексувати ці файли і зробити перший коміт. Це можна зробити за допомогою декількох команд git add, що визначають файли, за якими необхідно слідкувати, після чого треба виконати git commit:

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'Перша версія проекту'
```

Клонування існуючого репозиторію: для отримання копії існуючого Git репозиторію – наприклад, проекту, в якому ви хочете прийняти участь – потрібна команда git clone. Git отримує повну копію майже всіх даних, що є у сервера. Кожна версія кожного файлу в історії проекту витягується автоматично, коли ви виконується git clone. Якщо щось станеться з диском серверу, зазвичай можна використати майже будь-який з клонів на будь-якому клієнті щоб повернути сервер до стану на момент клонування.

Щоб клонувати репозиторій треба використати команду git clone [url]. Наприклад, для клонування бібліотеки Git libgit2:

```
$ git clone https://github.com/libgit2/libgit2
```

Це створить директорію під назвою “libgit2”, проведе ініціалізацію директорії .git, забере всі дані для репозиторію, та приведе директорію до стану останньої версії.

Якщо необхідно зробити клон репозиторія в директорію з іншою назвою, можна задати її у наступному параметрі команди:

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

Ця команда робить те саме, що й попередня, тільки цільова директорія називається mylibgit.

Git має декілька різних протоколів передачі даних, які можна використовувати: Попередній https://, git:// або user@server:шлях/до/репозиторію.git, що використовує SSH протокол.

8 Створити проект з кількома файлами, синхронізувати його з сайтом. Якщо у вас вже є локальний репозиторій Git і ви хочете його викласти в загальний доступ, то спочатку необхідно створити віддалений репозиторій (наприклад на GitHub), а потім виконати команди наведені нижче, змінивши відповідно назву вашого репозиторію.

Зв’язування локального репозиторію з віддаленим:

```
$ git remote add origin https://github.com/n0tb0dy/UpRemote.git
```

Публікування вітки master у віддалений репозиторій:

```
$ git push -u origin master
```

9 Внести зміни до проекту. Відобразити зміни у проекті на сайті. Додавання всіх файлів (в тому числі і папок):

```
$ git add .
$ git commit –m ‘додавання файлів’
```

Встановлення зв'язку з віддаленим репозиторієм, обчислення локальних змін відсутніх в ньому, і їх передачі в віддалений репозиторій:

```
$ git push
```

10 Внести зміни на сайті і відобразити їх локально в проекті. Якщо ваша гілка налаштована слідкувати за віддаленою гілкою, це можна виконати з допомогою команди git pull для автоматичного отримання змін та злиття віддаленої гілки до поточної гілки (табл. 1.1.1).

11 Додати чужий проект.

Основні команди git наведено в табл. 1.1.1, дія команд в системі git показана на рис. 1.1.4.

Детальна інформація про можливості роботи з Git - https://git-scm.com/book/uk/v2, основні команди - http://www.fandroid.info/shpargalka-po-komandam-git/.

Таблиця 1.1.1 - Основні команди git

Команда	Призначення
git add	додає вміст робочої директорії в індекс (staging area) для подальшого commit
git status	показує стану файлів в робочій директорії і індексі: які файли змінені, але не додані в індекс; які очікують комітів в індексі.
git commit	бере всі дані, додані в індекс за допомогою git add, і зберігає їх зліпок у внутрішній базі даних, а потім переміщує вказівник поточної гілки на цей зліпок. Ключі: -a для додавання всіх змін в індекс без використання git add, -m для передачі повідомлення коммітів без запуску редактора.
git fetch	зв'язується з віддаленим репозиторієм і забирає з нього всі зміни, яких поки немає і зберігає їх локально.
git pull	працює як комбінація команд git fetch і git merge, тобто Git спочатку забирає зміни із зазначеного віддаленого сховища, а потім намагається злити їх з поточної гілкою.
git push	використовується для встановлення зв'язку з віддаленим репозиторієм, обчислення локальних змін відсутніх в ньому, і їх передачі в вищезгаданий репозиторій.

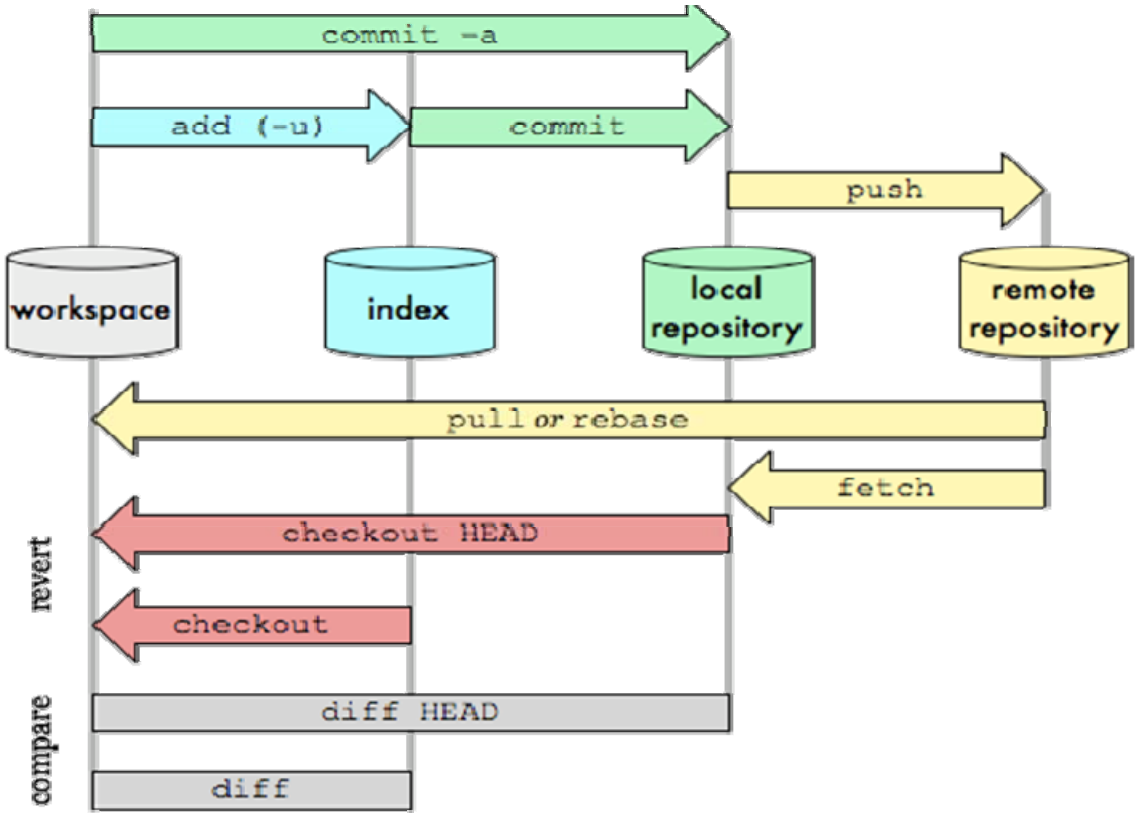

Рисунок 1.1.4 - Команди переміщення даних Git

Отже, в результаті взаємодії з системою, стало можливим відслідковувати зміни файлів проекту. Складність системи не дозволяє повністю автоматизувати внесення змін, але слід розуміти що зміни у проекті відбуваються під контролем розробника, і він самостійно може додати до списку змін внесенні ним корективи.

12 Оформити звіт.

1.1.3 Зміст звіту

Звіт повинен містити:

– титульний аркуш;

– мету роботи і завдання;

– покроковий опис роботи, копії екранів з результатами виконаної роботи;

– висновки.

Запитання для самоконтролю:

Тривалість заняття: 2 год.