

ЩО ТАКЕ VCS. РОБОТА З GIT

План лекції

Репозиторій та VCS

Конфігурація GIT репозиторію та основні команди

Створення та конфігурація віддаленого репозиторію, поєднання з локальним

Розгалуження та пов'язані з ним команди та процеси



Репозиторій та VCS

Репозиторій

Репозиторій - це централізоване або розподілене сховище, де зберігаються файли, дані та інформація про версії цих файлів у системах контролю версій (VCS). Це місце, де зберігається повна історія змін файлів, яка дозволяє відстежувати та контролювати розвиток проекту.



Основні аспекти репозиторію

1.Зберігання файлів: Репозиторій - це місце для зберігання різних типів файлів (тексти, зображення, вихідний код програм і т.д.), які використовуються для побудови проектів.

2.Історія змін: Всі зміни, внесені до файлів у репозиторії, зберігаються разом з інформацією про те, хто, коли і що змінив у файлі.

3.Гілки та версії: Репозиторій може мати різні гілки (branches), що дозволяє розробникам працювати паралельно над різними частинами проекту. Також у репозиторії зберігаються різні версії файлів, що дає змогу відстежувати розвиток проекту у часі.

4.Керування доступом: Деякі системи контролю версій дозволяють налаштовувати права доступу до репозиторію, визначаючи, хто може читати, записувати або видаляти файли в репозиторії.

5.Командна робота: Репозиторії дозволяють розробникам спільно працювати над проектом, здійснюючи обмін змінами та спільно вносячи виправлення або новий код.

Локальний репозиторій

Локальний репозиторій - це сховище файлів та їх історії змін, яке знаходиться безпосередньо на робочій станції розробника. Всі зміни файлів та їх версії зберігаються тільки на локальному комп'ютері, що дозволяє розробнику працювати самостійно без зв'язку з мережею або іншими розробниками. Це зручно для індивідуальних проектів або експериментів, де не потрібно спільної роботи з іншими.



Віддалений репозиторій

Віддалений репозиторій - це сховище файлів та їх історії змін, розміщене на сервері, доступ до якого можуть мати кілька розробників. Розробники спільно працюють з цим централізованим сховищем, здійснюючи коміти (збереження змін) та отримуючи оновлення з центрального репозиторію. Віддалений репозиторій забезпечує спільну роботу над проектом, можливість відстеження змін, управління версіями та спільний доступ до коду для всієї команди розробників.





Система контролю версій (VCS)

VCS (Version Control System) - це система контролю версій, яка використовується для відстеження змін у файлах і коді програмного забезпечення з часом. Основна мета VCS полягає в тому, щоб зберігати, відстежувати та координувати зміни, внесені до файлів, дозволяючи розробникам спільно працювати над проектом та відслідковувати його історію.



Ключові аспекти систем контролю версій (VCS)

- 1.Історія версій:** VCS зберігає повну історію змін файлів, що дозволяє вам переглядати, повертатися та порівнювати попередні версії. Це корисно для відстеження та розуміння того, як відбувалися зміни у файлів протягом часу.
- 2.Відгалуження (Branching) та злиття (Merging):** Системи контролю версій дозволяють створювати відгалуження (гілки) у репозиторії для вирішення різних завдань без впливу на основний код. Пізніше ці гілки можна злити разом. Це дозволяє розробникам ефективно працювати над різними функціями чи виправленнями помилок паралельно.
- 3.Робота в режимі офлайн:** В деяких системах, зокрема у розподілених VCS, розробники можуть працювати над проектом навіть у відсутність зв'язку з центральним сервером, зберігаючи повну копію репозиторію локально.
- 4.Конфлікти та управління версіями:** VCS надає засоби вирішення конфліктів, що виникають, коли дві чи більше гілок мають різні зміни в одних і тих же місцях файлів. Крім того, вони допомагають контролювати, які версії файлів використовуються в різних гілках або в різних частинах проекту.
- 5.Спільна робота та доступ до історії змін:** VCS дозволяють розробникам спільно працювати над проектами, ділитися змінами, а також відстежувати, хто і коли вніс певні зміни в код чи файли.
- 6.Автоматичне резервне копіювання:** Системи контролю версій забезпечують резервне копіювання коду та файлів, що допомагає уникнути втрати даних через випадкове видалення чи пошкодження файлів.
- 7.Аудит та безпека:** VCS дозволяють відстежувати, хто та коли здійснив зміни в коді. Це корисно для аудиту та забезпечення безпеки проектів.



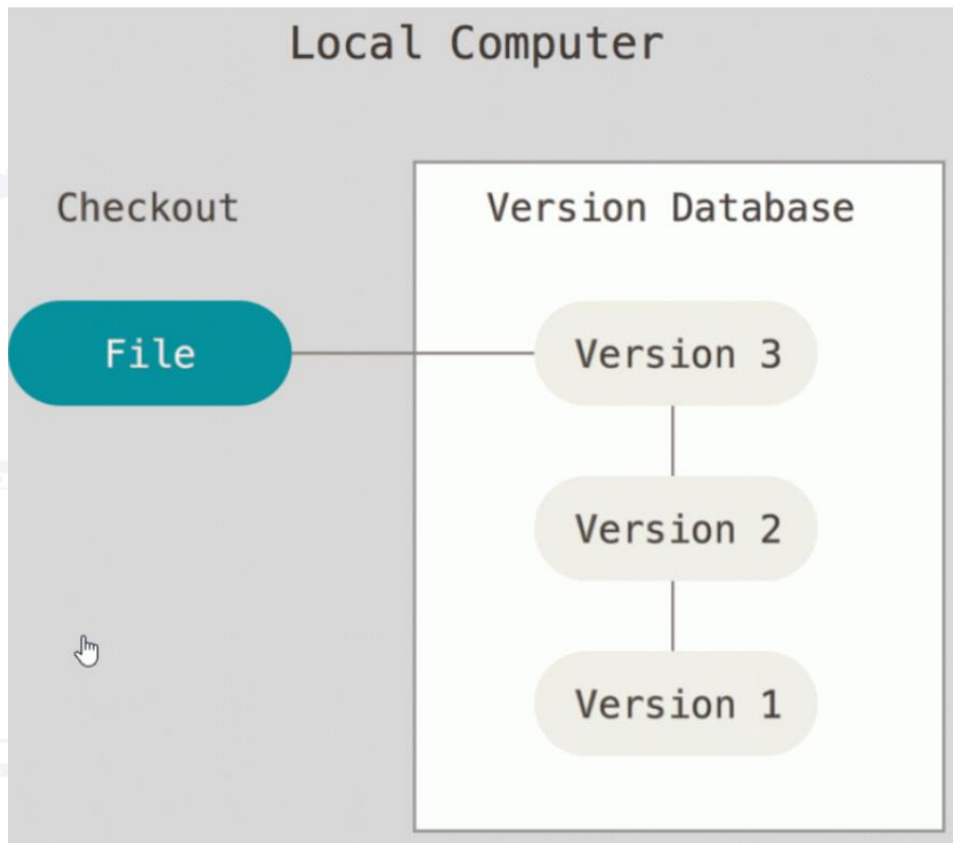
Типи VCS

1.Централізовані системи контролю версій (Centralized Version Control Systems - CVCS): У централізованих системах існує один центральний сервер, який зберігає всю інформацію про репозиторій (сховище файлів та їх версій). Розробники взаємодіють із цим центральним сервером, здійснюючи коміти (збереження змін), витягуючи оновлення та працюючи зі своїми копіями файлів. Приклади централізованих систем контролю версій включають Subversion (SVN) та CVS (Concurrent Versions System).

2.Розподілені системи контролю версій (Distributed Version Control Systems - DVCS): У розподілених системах кожен розробник отримує повну копію репозиторію. Це означає, що кожен клон репозиторію є самодостатнім і містить повну історію змін. Розробники можуть працювати локально без зв'язку з центральним сервером, що полегшує роботу в умовах обмеженої або відсутньої мережі. Приклади розподілених систем контролю версій включають Git, Mercurial.

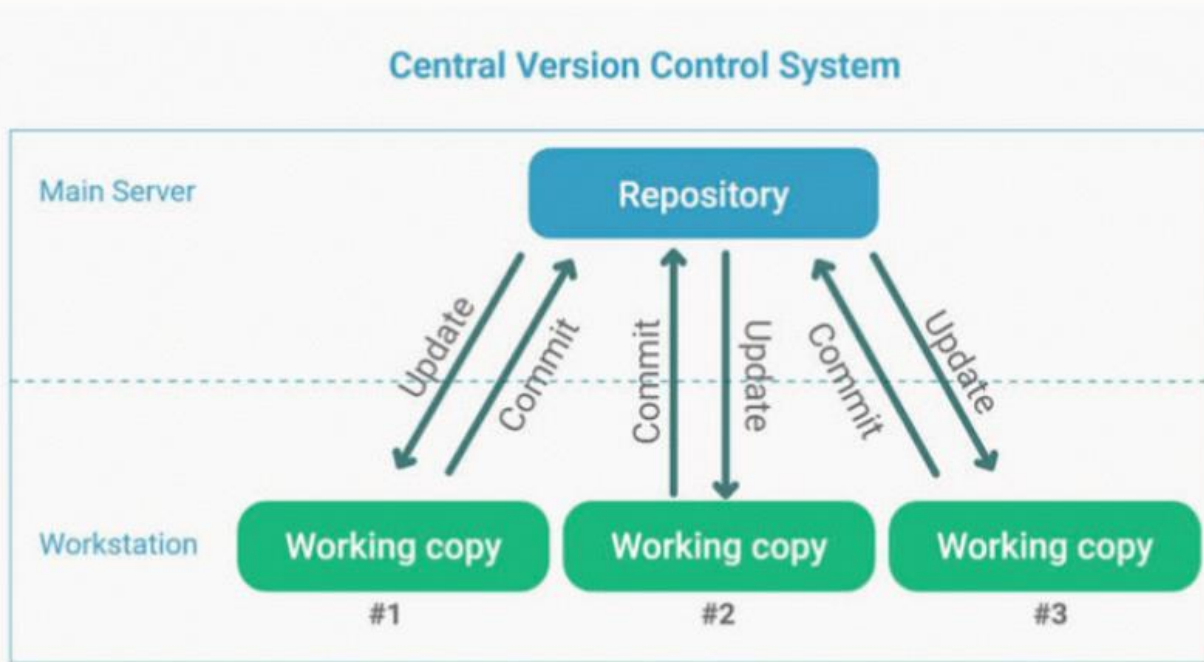
3.Локальні системи контролю версій (Local Version Control Systems - LVCS): прості системи, які використовуються для контролю версій файлів на одній конкретній робочій станції або комп'ютері. LVCS зберігає історію змін без залучення віддаленого сервера. У локальних системах контролю версій кожна зміна в файлі зберігається локально. Наприклад, один файл може мати кілька версій, які відстежуються та зберігаються прямо на тому самому комп'ютері. Такі системи не підтримують спільну роботу кількох розробників, оскільки вони призначені для використання однією особою. Хоча локальні системи контролю версій можуть бути простими у використанні, вони мають обмеження, особливо коли йдеться про спільну роботу над проектами або зберігання історії змін в розподіленій команді. Це призвело до розвитку централізованих та розподілених систем контролю версій, які надають більш широкі можливості спільної роботи та управління версіями файлів у більш складних проектах.

Основні принципи роботи локальних систем контролю версій



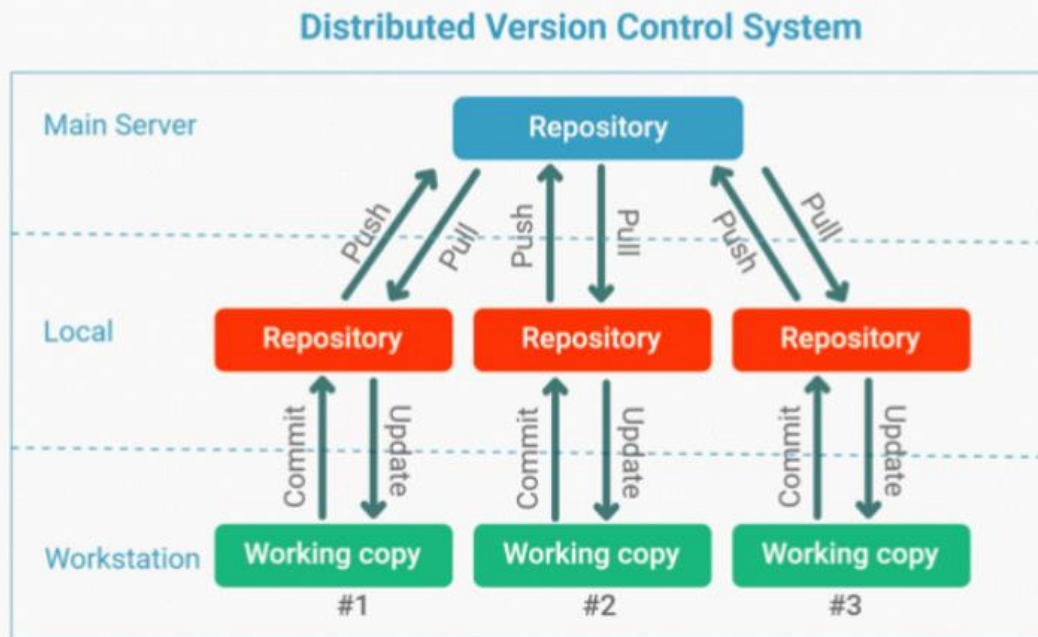
Основні принципи роботи централізованих систем контролю версій (CVCS)


Centralized (CVCS)



Основні принципи роботи розподілених систем контролю версій (DVCS)

Distributed (DVCS)





Конфігурація GIT репозиторію та основні команди

GIT

Git - це розподілена система контролю версій, розроблена Лінусом Торвальдсом у 2005 році. Вона призначена для ефективного керування версіями файлів та спільної роботи розробників над проектами програмного забезпечення.



Області GIT

1.Робоча область (Working Directory): Робоча область - це каталог на вашому комп'ютері, де ви працюєте з файлами проекту. Це місце, де ви створюєте, редагуєте та видаляєте файли та теки. Git слідкує за змінами у вашій робочій області і може відслідковувати, які файли були змінені.

2.Область індекса (Staging Area): Область індекса є проміжною зоною між вашою робочою областю і репозиторієм. Файли, які ви бажаєте включити до наступного коміту (збереження змін), спочатку додаються до області індекса. Це дозволяє вам вибирати, які конкретні зміни ви хочете включити у наступний коміт.

3.Репозиторій (Repository): Репозиторій - це місце, де Git зберігає всю історію змін вашого проекту. Він містить усі коміти, гілки, теги та інші важливі дані проекту. Кожен коміт у репозиторії включає інформацію про зміни, автора, дату та повідомлення про коміт.

Статуси відстеження файлів у GIT

1.Untracked (Невідстежувані): Файли, які Git не відстежує. Це можуть бути нові файли, які ви створили або додали до робочої області, але ще не були додані до області індекса за допомогою команди **git add**.

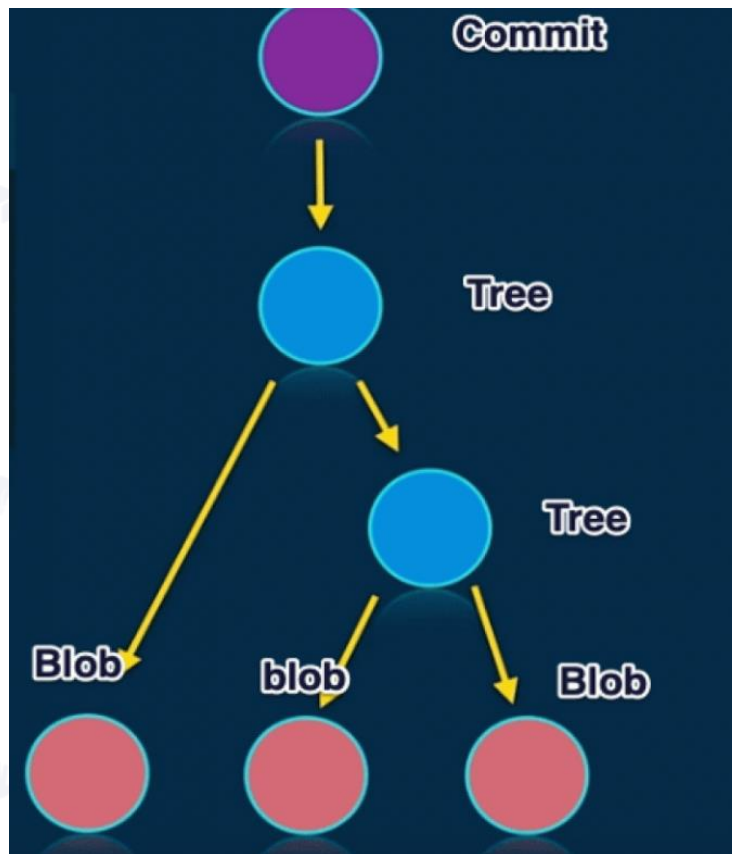
2.Modified (Змінені): Файли, які вже були відстежені Git, але були змінені після останнього коміту. Git розпізнає, що файли мають невідповідності відносно їхньої версії, яка зберігається в останньому коміті.

3.Staged (Готові до коміту): Файли, які вже були додані до області індекса за допомогою команди **git add**. Ці файли готові для включення у наступний коміт.

4.Unmodified (Незмінені): Файли, які відстежуються Git і не були змінені після останнього коміту. Git розпізнає, що файли збігаються з їхніми версіями у репозиторії.

5.Ignored (Ігноровані): Файли, які зазначені у файлі **.gitignore** та вони будуть проігноровані Git. Це дозволяє вам виключити певні файли чи каталоги з відстеження Git (наприклад, файли конфігурації, тимчасові файли тощо).

Основні типи об'єктів Git

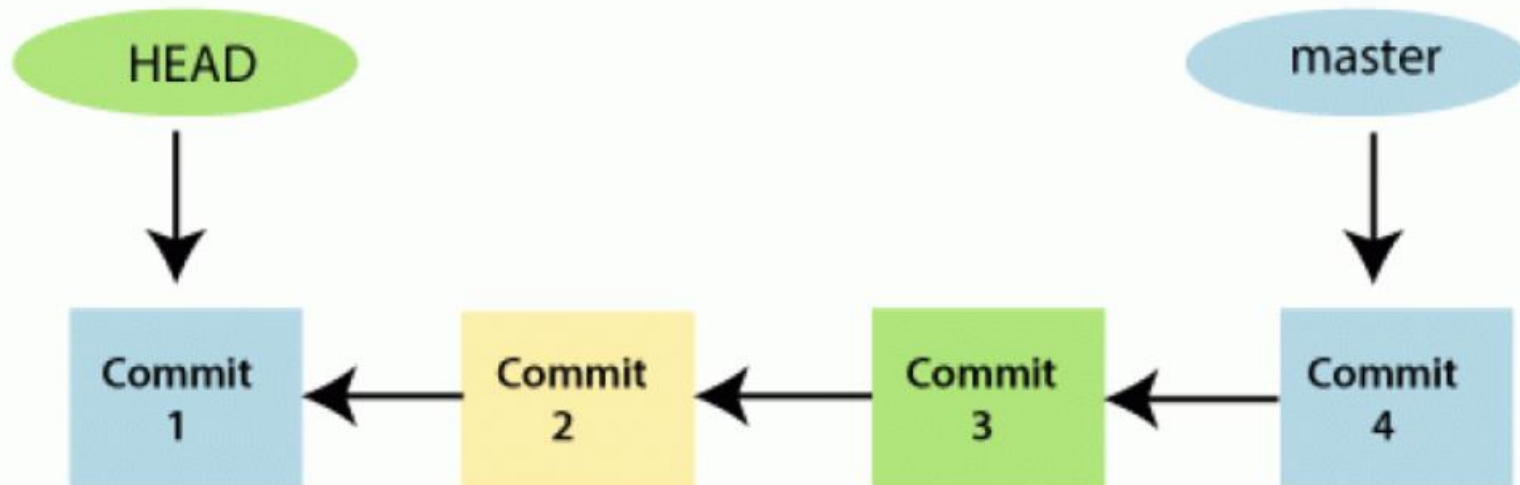


Структура коміту в Git

- 1.Хеш (SHA-1):** Кожен коміт має унікальний хеш-код (SHA-1), який ідентифікує його. Цей хеш використовується для однозначного ідентифікування коміту та забезпечення його унікальності в системі Git.
- 2.Автор та комітер:** Кожен коміт містить інформацію про автора та комітера. Це включає ім'я та електронну адресу людини, яка створила коміт, іноді також може включати додаткову інформацію про автора (наприклад, дату).
- 3.Повідомлення про коміт:** Коміт також має повідомлення, яке пояснює природу змін, що відбулися у цьому коміті. Це повідомлення допомагає іншим розробникам розуміти, що було зроблено у цьому коміті та які були його метою.
- 4.Зміст (Tree):** Кожен коміт вказує на дерево (Tree), яке відображає стан файлів і каталогів у проекті на момент створення коміту. Він містить посилання на Blob-об'єкти (вміст файлів) та інші Tree-об'єкти (підкаталоги), що відображає структуру проекту.
- 5.Батьківські коміти (Parent Commits):** Крім того, кожен коміт може мати посилання на один або кілька попередніх комітів (батьківських комітів), які стали базою для даного коміту.
- 6.Час та дата створення:** Коміт містить інформацію про час та дату його створення.

Head

Checkout<commit-1>



Основні команди Git

Конфігурація та початок роботи

- 1.git init:** Ініціалізує новий Git репозиторій у поточній папці.
- 2.git clone <URL>:** Клонує існуючий віддалений репозиторій на ваш комп'ютер.
- 3.git config:** Команда для налаштування параметрів Git, таких як ім'я користувача, електронна пошта тощо.

Основні команди роботи зі змінами

- 1.git status:** Показує статус файлів у робочій області (змінені, невідстежувані, готові для коміту тощо).
- 2.git add <file>:** Додає конкретний файл до області індекса для підготовки до коміту.
- 3.git reset <file>:** Скасовує зміни в конкретному файлі та видаляє його з області індекса.
- 4.git commit -m "Message":** Зберігає зміни, які були додані до області індекса, у репозиторії, разом із повідомленням про коміт.
- 5.git diff:** використовується для показу різниці між робочою областю та областю індекса. Вона відображає зміни, які ще не були додані до області індекса.
- 6.git clean:** використовується для видалення недороблених або невідстежених файлів у робочій області. Вона видаляє файли, які не знаходяться в області індекса.

Історія та перегляд

- 1.git log:** Виводить історію комітів у вашому репозиторії.
- 2.git show <commit>:** Показує детальну інформацію про конкретний коміт.

Основні команди Git

Робота з гілками

- 1.**git branch**: Відображає список гілок у вашому репозиторії.
- 2.**git branch <branch_name>**: Створює нову гілку.
- 3.**git checkout <branch_name>**: Перемикає вас на іншу гілку або коміт.
- 4.**git merge <branch>**: Зливає зміни з обраної гілки у поточну гілку.
- 5.**git push**: Надсилає локальні зміни у віддалений репозиторій.
- 6.**git pull**: Оновлює ваш локальний репозиторій з віддаленого, виконуючи злиття та отримуючи оновлені дані.

Інші команди

- 1.**git remote -v**: Відображає список віддалених репозиторіїв, пов'язаних з вашим репозиторієм.
- 2.**git diff**: Показує зміни між робочою областю і областю індекса.

Створення та конфігурація віддаленого репозиторію, поєднання з локальним



Віддалений репозиторій у системі контролю версій Git - це версія вашого проекту, яка знаходиться на віддаленому сервері або хостинг-платформі, доступний через Інтернет або мережу. Цей репозиторій може бути використаний для спільної роботи над проектами з декількома розробниками або для зберігання резервних копій вашого коду.



Основні характеристики віддаленого репозиторію в Git

- 1.Спільна робота:** Віддалений репозиторій дозволяє різним розробникам працювати над одним проектом. Кожен може отримувати оновлення та відправляти свої зміни до цього централізованого репозиторію.
- 2.Зберігання даних:** Він зберігає історію комітів, гілки, відстежування змін та інші важливі дані, що відображають стан проекту.
- 3.Резервне копіювання:** Віддалений репозиторій також може слугувати як резервна копія вашого проекту. Це може бути важливим у випадку втрати даних або коли потрібно відновити попередні версії проекту.
- 4.Доступність через мережу:** Віддалені репозиторії зазвичай доступні через HTTP/HTTPS або SSH протоколи. Вони можуть бути розміщені на таких платформах, як GitHub, GitLab, Bitbucket або на власному сервері.

Типи авторизації

1. Авторизація за допомогою облікового запису та пароля: Це тип стандартної авторизації, де користувач вводить свій ім'я користувача та пароль для входу на GitHub.

2. SSH-ключі: Використання SSH-ключів для авторизації дозволяє безпечно з'єднувати ваш локальний комп'ютер з GitHub, не вводячи пароль кожного разу. Це особливо корисно при роботі з віддаленими репозиторіями та виконанні операцій, таких як **git push** та **git pull**.

3. OAuth-токени: OAuth-токени використовуються для отримання доступу до API GitHub або для авторизації в інших сервісах, які використовують GitHub API для інтеграції з GitHub. Ці токени можна створити в налаштуваннях облікового запису на GitHub та використовувати для забезпечення доступу до різних функцій.

4. Авторизація через соціальні мережі: GitHub також підтримує авторизацію через облікові записи в інших соціальних мережах, таких як Google або Facebook. Вона дозволяє вам увійти на GitHub за допомогою облікового запису соціальної мережі без створення окремого пароля.

5. GitHub App Tokens: Цей тип авторизації використовується для забезпечення доступу до GitHub API через GitHub Apps, які можуть бути встановлені на облікових записах користувачів або організаціях.

Розгалуження та пов'язані з ним команди та процеси



Розгалуження (Branching) в Git

Розгалуження (**Branching**) в Git - це механізм, що дозволяє створювати альтернативні лінії розвитку для вашого коду. Кожна гілка (branch) представляє собою окрему версію вашого проекту, де ви можете робити зміни, не впливаючи на основну лінію розвитку.

Основні аспекти розгалуження в Git:

1. Створення нової гілки: `git branch <ім'я_гілки>`

2. Перехід між гілками: `git checkout <ім'я_гілки>`

3. Створення та перехід на нову гілку одразу: `git checkout -b <ім'я_гілки>`

4. Подивитися список гілок: `git branch`

5. Злиття гілок (Merge):

`git checkout master`

`git merge <ім'я_гілки>`

6. Видалення гілки:

`git branch -d <ім'я_гілки>`



Feature branching

Feature branching - це практика розробки програмного забезпечення, яка передбачає створення окремої гілки (feature branch) для кожної нової функції чи фічі, яку розробник хоче додати до проекту. Цей підхід дозволяє розробникам працювати над конкретною функціональністю у відокремленому середовищі, не впливаючи на основний код проекту.

Основні аспекти Feature branching

1.Відокремлені гілки для функцій: Кожна нова функція чи фіча розробляється у своїй власній гілці (feature branch), що дозволяє ізолювати роботу над цією функцією від основного коду проекту.

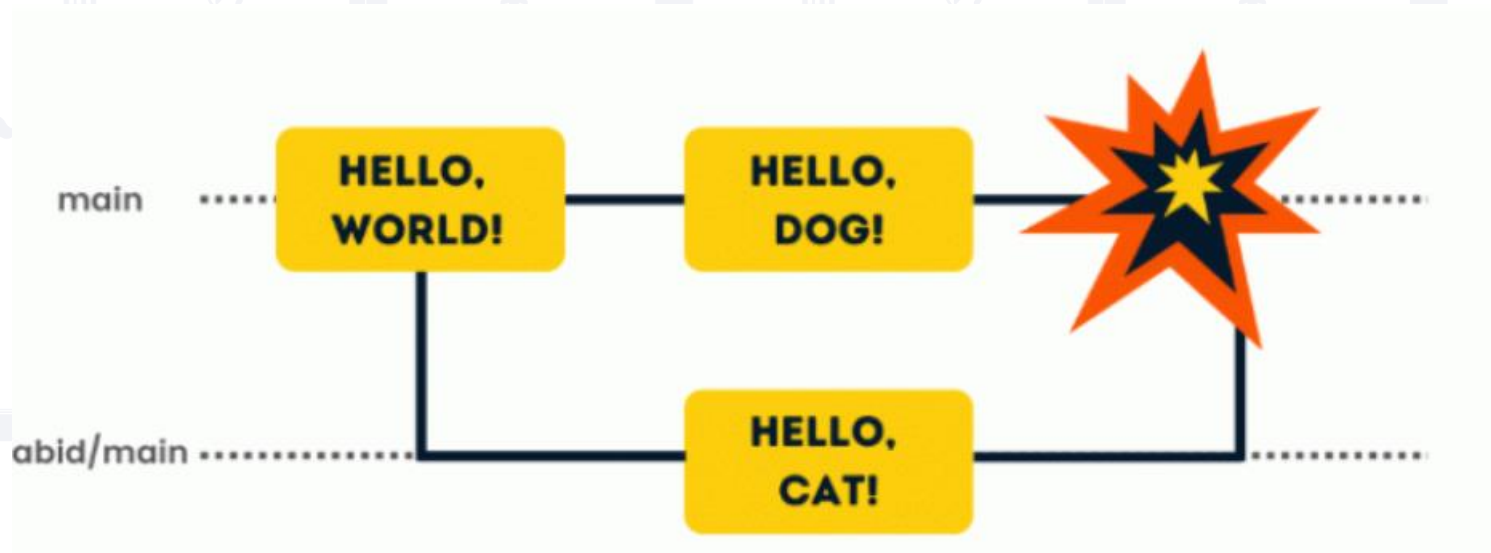
2.Незалежність роботи розробників: Кожен розробник може мати власну гілку для роботи над певною функцією, що уникає конфліктів між змінами, які вносять різні розробники.

3.Локалізовані зміни: Зміни, що вносяться в feature branch, відбуваються локально, доки вони не будуть готові для об'єднання з основною гілкою.

4.Тестування та рецензування: Функціональність в кожній feature branch може бути тестована та рецензована окремо, щоб переконатися, що нова функціональність працює коректно та відповідає вимогам.

5.Об'єднання з основною гілкою: Після завершення розробки функції відбувається об'єднання (merge) гілки з основною гілкою розвитку (наприклад, **master** чи **main**), і ця нова функціональність стає частиною основного коду.

Конфлікти в Git



Git pull

Команда `git pull` забирає дані з віддаленого репозиторію та об'єднує їх з поточною локальною гілкою. Вона використовується для отримання оновлень з віддаленого репозиторію та автоматичного об'єднання цих змін з вашою локальною гілкою.

Синтаксис:

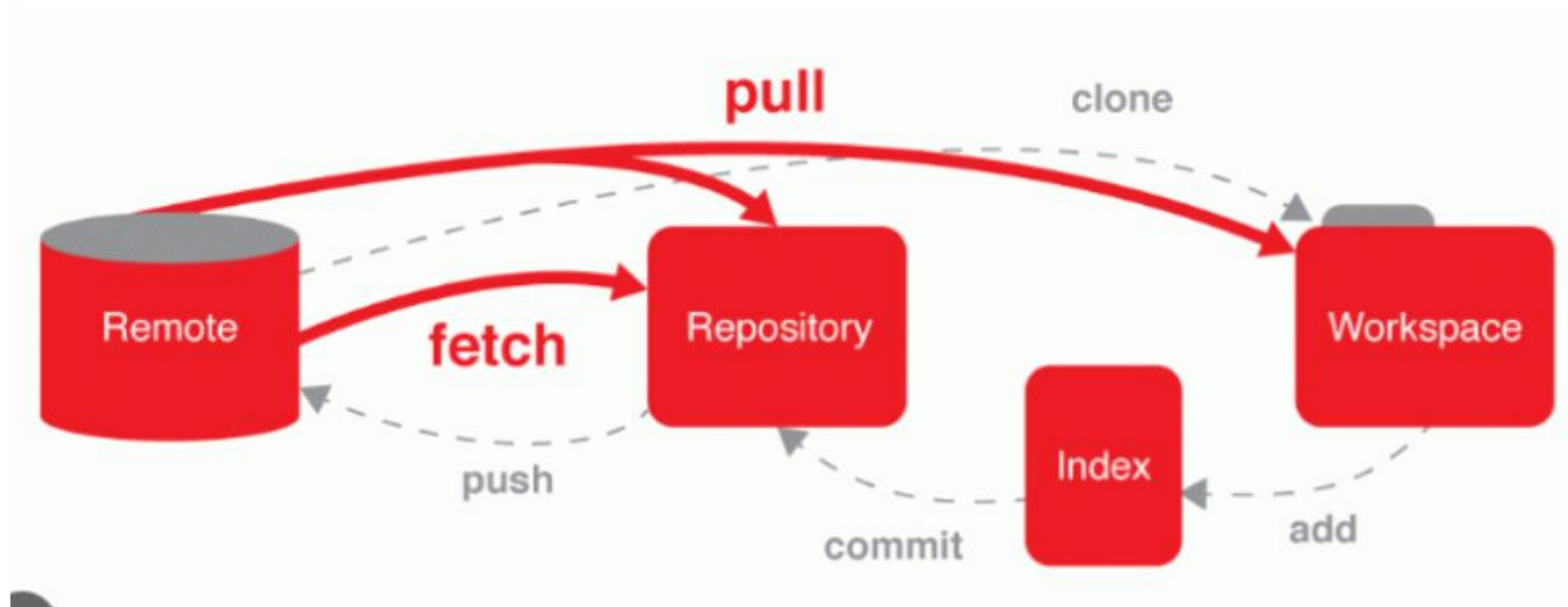
```
git pull <remote_name> <branch_name>
```

Git fetch

Команда **git fetch** забирає дані з віддаленого репозиторію, але не об'єднує їх автоматично з локальними гілками. Вона оновлює інформацію про віддалений репозиторій та гілки у вашому локальному репозиторії, але не змінює ваш робочий стан.

Синтаксис:

```
git fetch <remote_name>
```



Git merge

git merge об'єднує зміни з іншої гілки в поточну гілку. Коли ви виконуєте **git merge**, Git створює новий коміт, який об'єднує зміни з об'єднуваної гілки в поточну гілку. Це зазвичай приводить до створення додаткового коміту з повідомленням про злиття.

Синтаксис:

```
git merge <branch_name>
```

Git rebase

git rebase переносить зміни з однієї гілки на іншу, використовуючи історію комітів. Він "переміщує" ваші коміти з однієї гілки на іншу, видаляючи відмінності між гілками. Замість створення нового коміту з злиттям, **git rebase** змінює історію комітів, застосовуючи зміни на кінець гілки, на яку ви перебазуєтеся.

Синтаксис:

```
git rebase <branch_name>
```

Pull request (PR)

Pull request (PR) - це механізм, що використовується у системах керування версіями, наприклад, у Git та платформах для спільної розробки, наприклад, GitHub, GitLab, Bitbucket тощо. Це запит на обговорення та об'єднання змін, які внесені в одну гілку репозиторію (зазвичай у відокремленій гілці), з іншою гілкою чи головною гілкою (наприклад, **master** або **main**), з метою їх об'єднання.



ANY QUESTIONS ?

