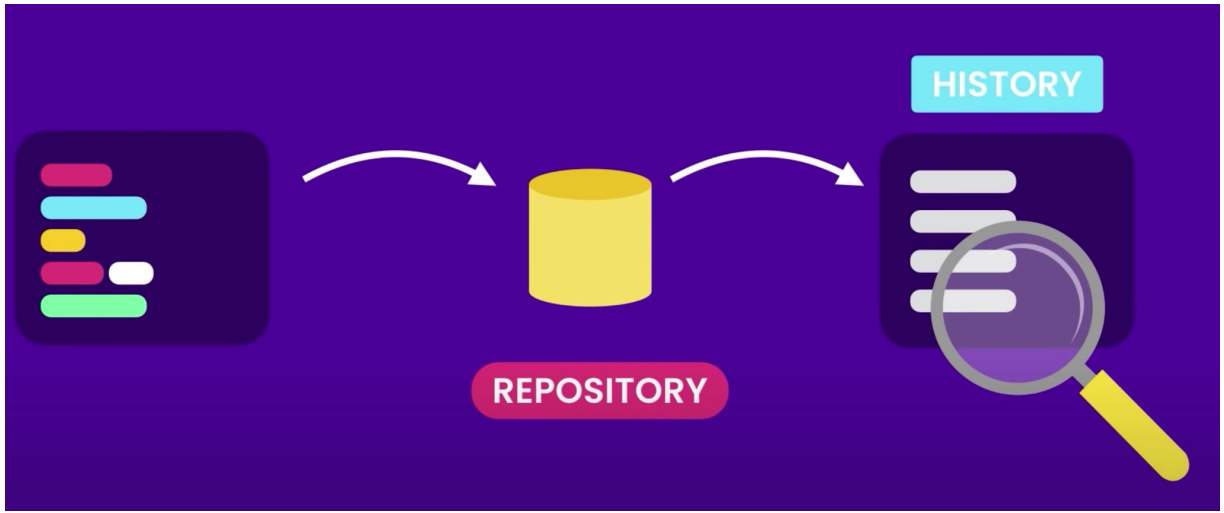
Git для початківців

Що таке Git і чому він такий популярний?

Git є найпопулярнішою системою контролю версій у світі.
Система контролю версій записує зміни, внесені до нашого коду з часом. У **спеціальній базі даних** під назвою **Repository** ми можемо переглянути історію нашого проекту та побачити, хто, коли і чому вніс зміни. І якщо якісь зміни щось зіпсують, ми можемо легко повернути наш проект до попереднього стану.



Чи можна без системи контролю версій?

Без системи контролю версій нам доведеться постійно зберігати **копії всього проекту в різних папках**. Це дуже повільно і взагалі не масштабується, особливо якщо над одним проектом працюють кілька людей. Нам доведеться постійно перекидати останній код електронною поштою чи іншими механізмами, а потім вручну об’єднувати зміни.

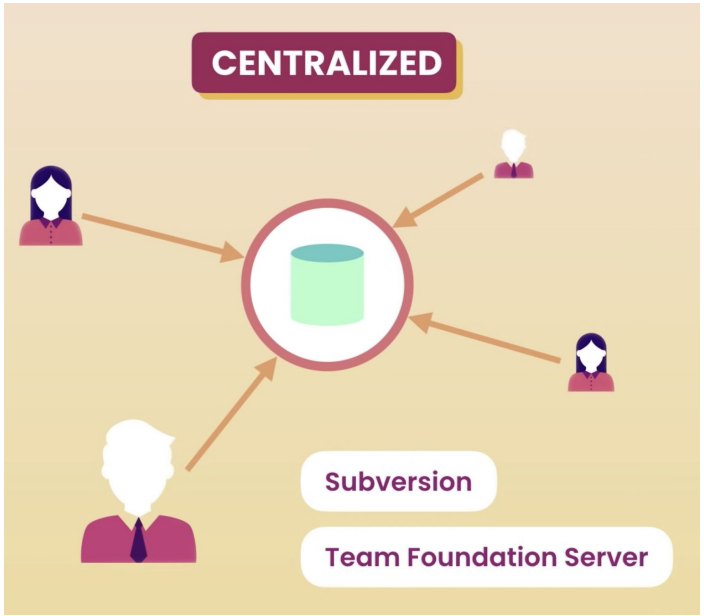
Система контролю версій дозволяє відстежувати історію нашого проекту та працювати разом.

Системи контролю версій бувають

- Централізовані
- Децентралізовані (розподілені)

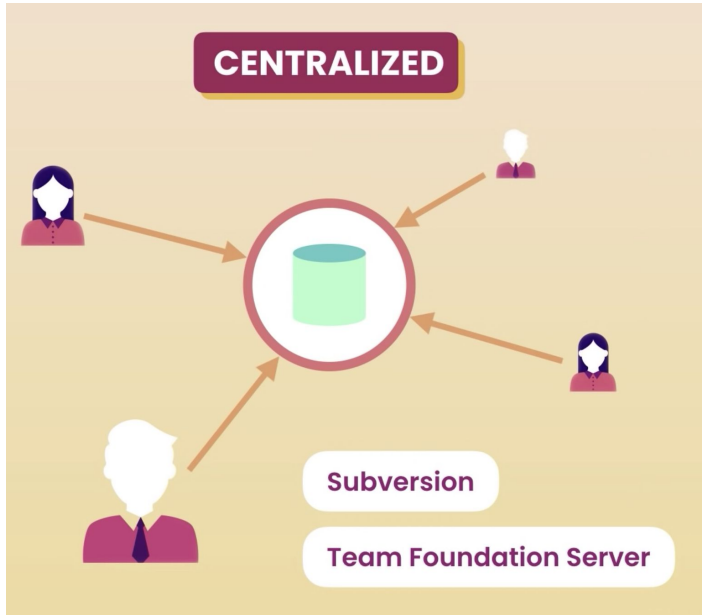
Централізовані системи контролю версій

У централізованій системі всі члени команди підключаються до центрального сервера, щоб отримати останню копію коду та поділитися своїми змінами з іншими. Subversion і Microsoft Team Foundation Server є прикладами централізованих систем контролю версій.



Централізовані системи контролю версій

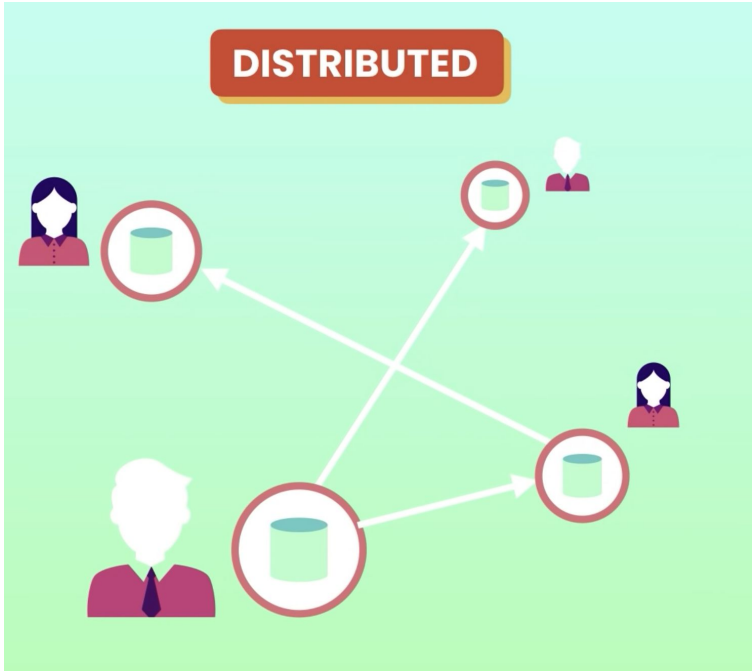
Проблема централізованої архітектури полягає в **єдиній точці відмови**. Якщо сервер відключається, ми не можемо зберегти знімки (снапшоти) нашого проекту, тому нам доведеться чекати, доки сервер відновиться.



Розподілені системи контролю версій

У розподілених системах у нас немає цих проблем. Кожен член команди має копію проекту з **усією історією проекту на своїй машині**. Тож, ми можемо зберігати знімки нашого проекту локально на нашій машині. Якщо центральний сервер офлайн, ми можемо синхронізувати нашу роботу безпосередньо з іншими.

Git і Mercurial є прикладами розподілених систем контролю версій.



Чому git ?

Бо ця система контролю версій

- безкоштовна
- з відкритим кодом
- надшвидка
- легко маштабується

Git використовується у 90% проектів в світі!

Де можемо використовувати git

- в командному рядку
- більшість сучасних редакторів коду та IDE* мають вбудовану підтримку основних функцій git
- GUI** для git - знайти їх найпростіше тут

Я покажу як використовувати командний рядок і GUI

* IDE - interactive development environment, наприклад, PyCharm або VSCode

** GUI - graphical user interface

Переваги командного рядку

- GUI містять найбільш популярні команди git, але не всі. При роботі з GUI все одно доведеться час від часу робити частину операцій з командного рядка
- Командний рядок завжди доступний. На віддаленому сервері у вас може не бути можливості встановити GUI
- Але іноді значно зручніше зробити щось через GUI замість командного рядка.

Я рекомендую суміщати роботу в GUI і в командному рядку.

Встановлення git

Перевірка встановленої версії git

```
git --version
```

Якщо версія застаріла - варто оновити на новішу.

Аби встановити git на вашу робочу машину - виконайте інструкції з офіційного сайта. Це нескладна процедура, у вас не має виникнути жодних проблем.

Для користування git на Windows я рекомендую користуватись git BASH терміналом (що встановиться після виконання інструкцій), замість стандартного командного рядка.

Конфігурація git

Ми можемо зафіксувати наступні конфігураційні параметри git:

- Ім’я
- Емейл
- Редактор за замовчанням
- Закінчення рядків

Конфігурація git

Конфігурування можливе на наступних рівнях

- Системний - для всіх користувачів
- Глобальний - для всіх репозиторіїв даного користувача
- Локальний - для поточного репозиторія

Конфігурація git

Переглянути поточні конфігураційні параметри

```
git config --list
```

Встановити ім’я, емейл

```
git config --global user.name “Hanna Pylieva”

git config --global user.email pylyeva@gmail.com
```

Встановлення редактора за замовчанням

Якщо цього не зробити, за замовчанням буде використовуватись vim - редактор, який багатьох лякає і є не дуже інтуїтивно зрозумілим.

Моя рекомендація для будь-якої ОС - Sublime Text.

Після встановлення редактора - необхідно додати його в PATH.

Змінити редактор в git

```
git config --global core.editor “subl --wait”
git config --global core.editor "'c:/program files/sublime text 3/sublime_text.exe' -w"
```

Всі налаштування в файлі

Усі ці налаштування зберігаються в текстовому файлі і ми можемо його відредагувати вручну. Переглянути файл:

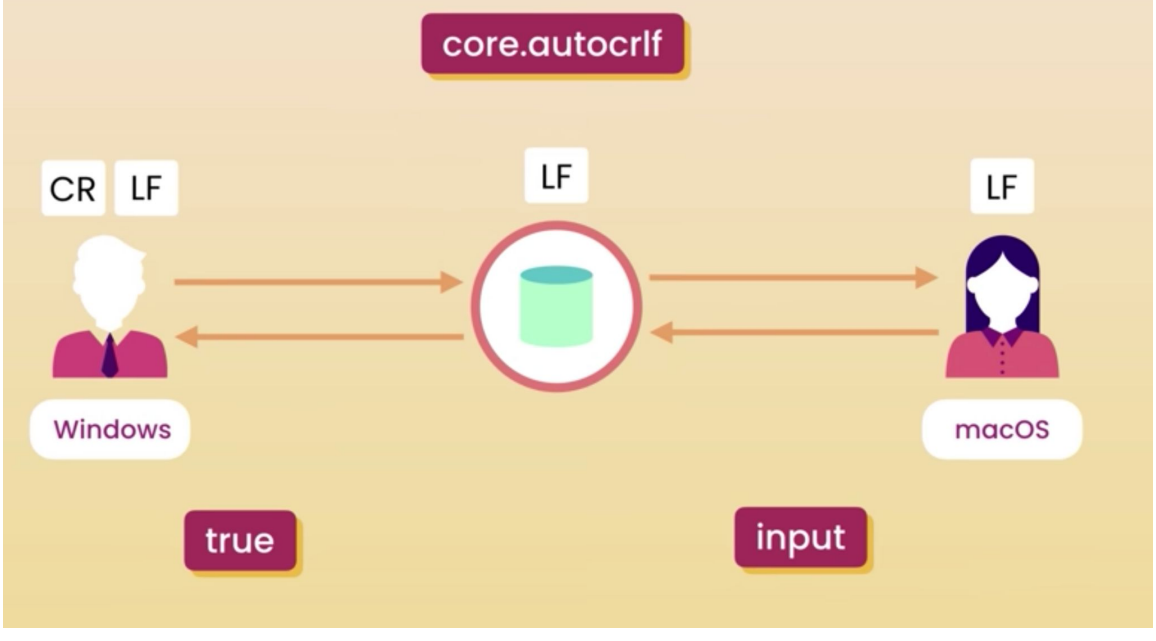
```
git config --global -e
```

Налаштування закінчення рядків

У Windows кінці ліній позначаються двома спеціальними символами, символами Return і Line Feed. У системах Mac і Linux кінці ліній позначаються переводом рядка. Відповідно за колаборативної роботи над проектом і неправильними налаштуваннями, можуть відбутись набажані речі.

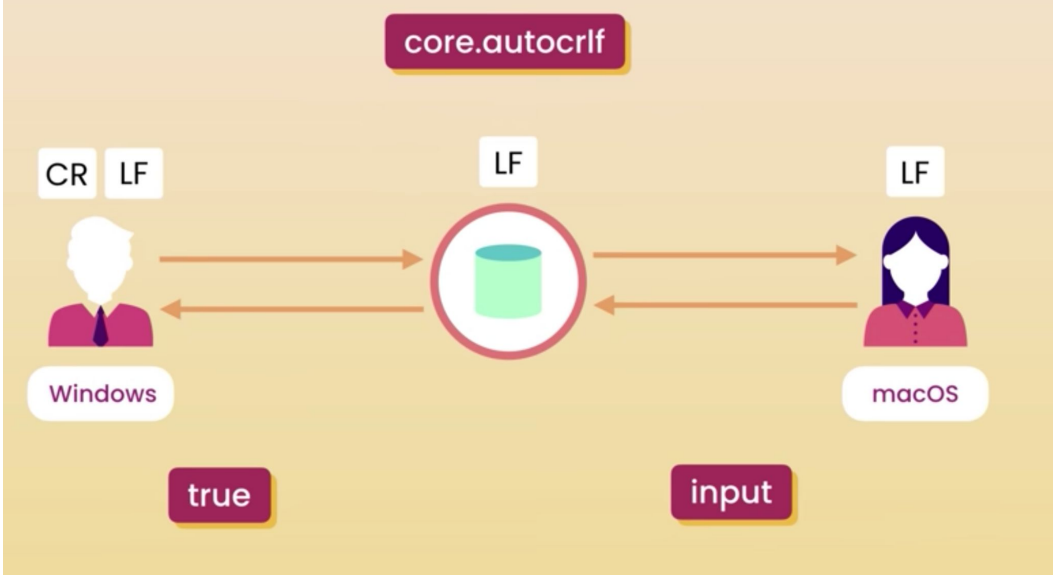
Налаштування закінчення рядків

Джон і Джулі працюють з одним репозиторієм. У Джона Windows, у Джулі Mac. Коли Джон захоче відправити свій код до сховища, гіт забаре CR. Так само, коли Джон забиратиме код із сховища, git має оновити кінець рядків і додати символ повернення каретки. Щоб досягти такої поведінки, ми повинні встановити для цієї властивості значення True в конфігі.



Налаштування закінчення рядків

З іншого боку, коли Джулі забирає код з репозиторія, або надсилає його туди - їй не потрібен символ повернення каретки, тому git не слід вносити змін у закінчення рядків. Однак, якщо повернення оператора випадково додається до кінця рядка через редактор, який використовує Джулі, git має видалити його під час зберігання коду в репозиторії. Щоб досягти такої поведінки, ми повинні встановити для цієї властивості значення Input, що означає, що git має змінювати рінці рядків лише під час зберігання коду в репозиторії.



Налаштування закінчення рядків

Mac/Linux

```
git config --global core.autocrlf input
```

Windows

```
git config --global core.autocrlf true
```

Де шукати підказки по командам git

Google:

наприклад “git config”

В командному рядку (видасть ту саму інформацію, що на офіційному сайті git в гуглі)

```
git config -help
```

В командному рядку натискаємо Enter аби прокрутити далі, q - аби вийти

Команди

Клонування існуючого репозиторія

```
git clone <https://name-of-the-repository-link>
```

Ініціювати гіт репозиторій локально

```
git init
```

Створити комміт - запис у історію гіта

```
git commit -m ‘Commit message’
```

Команди: робота з гілками

Створити гілку

```
git branch <branch-name>
```

Перейти на обрану гілку

```
git checkout <branch-name>
```

Створити гілку і одразу на неї перейти

```
git checkout -b <branch-name>
```

Внесення змін у репозиторій

Статус робочої директорії

```
git status
```

Створити комміт - запис у історію гіта

```
git add file1.txt file2.txt
git add *.txt
git add .
```

Створити комміт - запис у історію гіта

```
git commit -m ‘Commit message’
```

Кращі практики коміту коду

- Коміт має бути не надто малим і не надто великим.
- Не варто робити коміт щоразу, коли ми оновлюємо файл з повідомленням “Update file”. Коміт має відображати певну логічну зміну. Наприклад, “Change source database of raw data”.
- В роботі ми робимо зазвичай мінімум 1 коміт на день. Бо ми фіксуємо чекпоінти, аби могти відкатити зміни, якщо щось не запрацює.
- Щоразу, коли ви досягли нового статусу в задачі - робіть коміт.
- Повідомлення в коміті в імперативній формі: “Fix the bug”, а не “Fixing bug”, “Fixed the bug”

Перегляд історії

Історія комітів

```
git log
```

Історія всіх дій в гіті

```
git reflog
```

Зміни в історії

Відмінити останній комміт

```
git reset HEAD~1
```

Змінити head на якийсь конкретний коміт з reflog

```
git reset --hard <commit_hash>
```

Змінити head на якийсь конкретний коміт з reflog

```
git diff 1c603a5 01dfc2d > master.patch
git apply master.patch
```

Більше - у статті.

Happy commiting!