

Основи Git

№ уроку: 1 Основи Git

Засоби навчання: Комп'ютер

Огляд, мета та призначення уроку

Мета уроку - познайомитись з однією з найпопулярніших систем контролю версій – Git.

Вивчивши матеріал даного заняття, учень зможе:

- Створювати репозиторії та працювати з віддаленими репозиторіями.
- Створювати коміти,

Зміст уроку

1. Системи контролю версій
2. Локальна робота з репозиторієм
3. Базові команди

Резюме

У процесі створення коду будь-якого проекту одночасно працює кілька розробників. Виникає проблема із взаємообміном коду - хто і коли вніс якісь зміни, чи можна їх скасувати, а чи можна зберігати кілька версій одного і того ж файлу? Всі ці проблеми дають змогу вирішувати системи контролю версій (VCS = Version Control System).

Система управління версіями (або "система контролю версій", від англ. Version Control System, VCS або Revision Control System) - програмне забезпечення для спрощення роботи з інформацією, яка може змінитися. Система керування версіями дозволяє зберігати кілька версій одного і того ж документа, при необхідності повертатися до більш ранніх версій, визначати, хто і коли зробив ту чи іншу зміну та багато іншого.

Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів програми, що розробляється. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, які безперервно змінюються. Зокрема, системи керування версіями застосовуються у САПР (**С**истема **а**втоматизованого **п**роєктування), зазвичай у складі систем керування даними про виріб (PDM). Керування версіями використовується в інструментах конфігураційного управління (Software Configuration Management Tools).

Локальні системи контролю версій

Багато людей як метод контролю версій застосовують копіювання файлів в окремий каталог (можливо навіть каталог з позначкою за часом, якщо вони досить кмітливі). Цей підхід дуже поширений через його простоту, проте він неймовірно сильно схильний до появи помилок. Можна легко забути, в якому каталозі ви знаходитесь і випадково змінити той файл або скопіювати не ті файли, які ви хотіли.

Для того, щоб вирішити цю проблему, програмісти давним-давно розробили локальні СКВ із простою базою даних, яка зберігає записи про всі зміни у файлах, здійснюючи тим самим контроль ревізій.

Системи контролю версій бувають **централізовані (CVCS)** та **розподілені (DVCS)**.

CVCS – це старіший вид контролю версій. Вони використовувалися ще у сімдесяті роки. Єдине сховище версій – центральний репозиторій. Розробник працює з локальною копією та надсилає зміни до центрального репозиторію. Репозиторій видно всім (у кого є доступ) і обмін кодом – тільки через нього.

Приклади: SVN, Perforce, MS TFS, ClearCase.

DVCS – це «нова течія», перші системи з'явилися в дев'яності, початок 2000-х, але масового поширення набули з 2005 року. Кожен розробник володіє копією репозиторію, фактично, своїм локальним «сервером» контролю версій. Копії легко створювати: простіше експериментувати із кодом. Передавати зміни можна між будь-якою парою репозиторіїв. У розподілених Version Control System немає «головного» репозиторію.

Приклади: git, Mercurial, Bazaar.

Порівняння систем контролю версій

- На DVCS можна все те ж саме, що й на CVCS.
- На DVCS легше виконувати злиття гілок.
- На DVCS вся історія зберігається локально. Можна працювати офлайн і робота загалом швидша.
- Гнучкіша модель обміну змінами.
- Розробники звикли до CVCS, потрібно перебудовуватись.
- У CVCS нижче «порог входження» – для роботи з DVCS треба краще розуміти концепцію контролю версій.
- Найчастіше у світі використовується представник CVCS – SVN, а DVCS – git.

Базовий сценарій роботи із системами контролю версій

- Отримати локальну «робочу копію» коду з репозиторію.
- Внести зміни.
- У разі потреби: виконати злиття (merge) змін з новими правками в репозиторії.
- Зафіксувати зміни в репозиторії.
- **Git** – це безкоштовна та відкрита система контролю версій (SCM). Вона створена для відстежування змін коду у ваших проектах. Це надає вам контроль на кожному етапі розвитку вашого додатку.
- **Git** створено так, щоб над проектом могла працювати команда програмістів. Це полегшує координацію завдань між співробітниками та допомагає відстежувати зміни коду.
- **Git** – це просто інструмент. Він не прив'язаний до конкретної мови програмування. З його допомогою можна систематизувати код Python, C#, JavaScript, Java, Ruby та інших мов програмування.
- Тільки мови програмування? Ні, Git здатний працювати з будь-яким текстом.
- Щоб не прив'язувати наш курс до мови програмування, ми будемо використовувати псевдокод.

Встановлення

Щоб розпочати роботу з Git, його потрібно встановити. Завантажте встановлювач з офіційного сайту (<https://git-scm.com/>), запустіть його та дотримуйтесь його вказівок. Після цього в меню Пуск з'являться дві нові програми: Git Bash та Git GUI.

Як відкрити консоль?

- Варіант 1. Напишіть у пошуковому рядку меню Пуск cmd
- Варіант 2. Натисніть поєднання клавіш Win + R, у вікні, котре з'явилося, введіть cmd і натисніть Enter.

Початок роботи

- Відкрийте консоль Git Bash через меню Пуск. Щоб переконатися, що Git встановлено коректно, виконайте команду: `git --version`. Якщо у відповідь ви отримаєте версію встановленої програми, Git встановлений успішно.
- Оскільки Git допускає роботу в команді, кожному учаснику проекту потрібно "представитися" перед Git'ом, щоб він міг розрізняти зміни від різних розробників.
- Виконайте наступні команди, щоб git дізнався ваше ім'я та електронну пошту (змінить на свої дані):

```
git config --global user.name "Leonid Podriz"
git config --global user.email "leonidpodriz@gmail.com"
```

Створення порожнього проекту

Підготовка робочої директорії

Перехід до директорії: `cd d:`
 Шлях до поточної директорії: `cd .`
 Шлях до батьківської директорії: `cd ..`
 Створення нової директорії: `mkdir my_first_project`
 Створення файлу з текстом: `echo "my text" > test.txt`
 Перелік файлів у директорії: `ls`
 Читання файлу: `cat test.txt`
 Видалення файлу: `rm test.txt`

Почніть роботу зі створення порожнього каталогу з іменем *code*, потім увійдіть в нього і створіть там файл з ім'ям *main.code* з таким псевдокодом:

```
виведи("Привіт світ")
```

Створення порожнього репозиторію

Тепер у вас є каталог із одним файлом. Щоб створити git-репозиторій із цього каталогу, виконайте команду: `git init`

В результаті створюється прихована папка `.git`
 Для того, щоб побачити приховані папки та файли, скористайтесь командою: `ls -la`
 Для PowerShell: `ls - Force`
 Шлях до поточної директорії: `pwd`

Створення порожнього файлу

- Щоб створити файл, виконайте команду `touch`. Результат:

```
touch ім'я_файлу
```

Зміна файлу

- Щоб змінити файл, виконайте команду `nano`. Результат:

```
nano ім'я_файлу
```

- Щоб вийти зі стану зміни файлу, виконайте комбінацію клавіш `Ctrl+X`.

Git розглядає кожен файл у вашій робочій копії як файл одного з трьох вказаних нижче типів:

- файл, що відстежується – файл, який був попередньо проіндексований або зафіксований у коміті.
- файл, що не відстежується – файл, який не був проіндексований або зафіксований у коміті.
- ігнорований файл – файл, явно позначений для Git як файл, який необхідно ігнорувати. Це, як правило, артефакти складання та файли, що генеруються машиною з вихідних файлів у вашому репозиторії, або файли, які з будь-якої іншої причини не повинні потрапляти до комітів.

Поширені приклади таких файлів:

- кеші залежностей, наприклад, вміст `/node_modules` або `/packages`;
- скомпільований код, наприклад файли `.o`, `.pyc` і `.class`;
- каталоги для вихідних даних збирання, наприклад `/bin`, `/out` або `/target`;
- файли, згенеровані під час виконання, наприклад, `.log`, `.lock` або `.tmp`;
- приховані системні файли, наприклад, `.DS_Store` або `Thumbs.db`;
- особисті конфігураційні файли IDE, наприклад `.idea/workspace.xml`.

Ігноровані файли відстежуються у спеціальному файлі `.gitignore`, який реєструється в кореневому каталозі репозиторію. У Git немає спеціальної команди для вказівки файлів, що ігноруються: замість цього необхідно вручну відредагувати файл `.gitignore`, щоб вказати в ньому нові файли, які повинні бути проігноровані.

Файли `.gitignore` містять шаблони, які зіставляються з іменами файлів у репозиторії визначення необхідності ігнорувати ці файли.

Додавання файлу до репозиторію

- Створення репозиторію не означає, що Git відразу ж почав стежити за всіма файлами в папці. Файли потрібно додати вручну. Це дає контроль над файловою структурою Git репозиторію та гарантує, що жодні конфіденційні дані не потраплять у відкритий доступ без вашої згоди.
- Тепер давайте додамо до репозиторію файл з кодом.

```
git add main.code
git commit -m "Add my first code file"
```

- Команда `git commit` – служить для фіксації змін коду. Ми можемо додати багато файлів за допомогою команди `git add` і тільки один раз зафіксувати зміни.

Перевірка стану репозиторію

Використовуйте команду `git status`, щоб перевірити поточний стан репозиторію.

```
git status
```

Ця команда повертає інформацію про репозиторію:

1. Скільки файлів змінено з останнього коміту
2. Скільки файлів було додано до файлу, але не додано до репозиторію
3. Які файли було видалено і так далі.

По факту, `git status` дивиться на останній коміт і на поточний стан файлів у робочій папці. Якщо є відмінності – `git` повідомляє.

Не соромтеся частіше використовувати команду `git status` – це може вберегти вас від ненавмисних помилок. У нашому випадку ви побачите:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Команда перевірки стану повідомить, що комітити нічого. Це означає, що в репозиторії зберігається поточний стан робочого каталогу і немає жодних змін, які чекають на запис.

Ми будемо використовувати команду `git status`, щоб продовжувати відстежувати стан репозиторію та робочого каталогу.

Зміни

Тепер наша мета - навчитися відстежувати стан робочого каталогу. Змінимо код у `main.code`:

```
виведи("Привіт, Світ")
```

У код додана кома і "Світ" написано з великої літери. Як на це відреагує Git?

```
C:\Users\38099\Desktop\git_lesson> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   main.code

no changes added to commit (use "git add" and/or "git commit -a")
```

- Git знає, що файл `hello.html` був змінений, але ці зміни ще не зафіксовані в репозиторії.
- Зверніть увагу, що повідомлення про стан дає вам підказку про те, що потрібно робити далі. Якщо ви хочете додати ці зміни до репозиторію, використовуйте команду `git add`. В іншому випадку використовуйте команду `git checkout` для скасування змін.

Обробка змін

Додайте зміни

Після змінення файлів, потрібно проіндексувати зміни, виконавши команду `git`. Перевірте стан:

```
C:\Users\38099\Desktop\git_lesson>git add main.code

C:\Users\38099\Desktop\git_lesson>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   main.code
```

- Зміни файлу `main.code` проіндексовані. Git тепер знає про зміну, але зміна поки що не перманентно (читай: не назавжди) записана до репозиторію. Наступний коміт включатиме проіндексовані зміни.
- Якщо ви вирішили, що не хочете комітити зміни, команда стану нагадає вам про те, що за допомогою команди `git restore` можна зняти індексацію цих змін.

Коміт

- Окремий крок індексації у git дозволяє вам продовжувати вносити зміни до робочого каталогу, а потім, у момент, коли ви захочете взаємодіяти з версійним контролем, git дозволить записати зміни у малих коммітах, які фіксують те, що ви зробили.
- Припустимо, що ви відредагували три файли (main.code, module.code та math.code). Тепер ви хочете закомітити всі зміни, при цьому, щоб зміни в main.code і module.code були одним коммітом, тоді як зміни в math.code логічно не пов'язані з першими двома файлами і повинні йти окремим коммітом.

Теоретично, ви можете зробити наступне:

```
git add main.code
git add module.code
git commit -m "Changes for main and module"
git add math.code
git commit -m "Unrelated change to math"
```

Розділяючи індексацію та коміт, ви маєте можливість з легкістю налаштувати, що йде в який коміт.

Комітимо зміни

- Повернемося до нашого прикладу. Ми змінили та проіндексували файл main.code.
- Раніше ми використовували git commit для коміту початкової версії файлу main.code у репозиторій. Ми включили мітку -m, яка робить коментар у командному рядку.
- Команда commit дозволить вам інтерактивно редагувати коментарі для коміту. Якщо ви опустите мітку -m з командного рядка, git перенесе вас до редактора на ваш вибір.

Виконайте:

```
git commit
```

Ви побачите у вашому редакторі:

```
|
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git restore --staged <file>..." to unstage)
#
#   modified:   main.code
#
```

У першому рядку введіть коментар: Fix typo mistakes. Збережіть файл і вийдіть з редактора (для цього в редакторі за замовчуванням (Vim) потрібно натиснути клавішу ESC, ввести ":wq" і натиснути Enter). Ви побачите:

```
C:\Users\38099\Desktop\git_lesson>git commit
[master 52f5a44] Fix typo mistakes
1 file changed, 1 insertion (+), 1 deletion (-)
```

Перевірте стан

Насамкінець давайте ще раз перевіримо стан.

```
git status
```

Ви побачите:

```
C:\Users\38099\Desktop\git_lesson>git status
On branch master
nothing to commit, working tree clean
```

Робочий каталог є чистим, можете продовжити роботу.

Історія

Щоб отримати історію комітів у репозиторії, використовуйте команду: `git log`.

Можливий результат:

```
commit 52f5a44d66e6a6040885b6ee73929d9d08e93e65 (HEAD -> master)
Author: Leonid Podriz <leonidpodriz@gmail.com>
Date: Tue Aug 18 11:11:41 2020 +0300
```

Fix typo mistakes

```
commit e41ae66ba3b20c6f7f4287305a306513c24788d1
Author: Leonid Podriz <leonidpodriz@gmail.com>
Date: Tue Aug 18 10:33:23 2020 +0300
```

Add fisrt code file

Версії

- Git - система контролю версіями. Настав час дізнатися, як перемикатися між нашими версіями (комітами).
- У попередньому розділі ви дізналися, як можна переглянути історію комітів. Припустимо, що мені потрібно перейти на найперший коміт і продовжити роботу з нього. Для цього мені потрібно дізнатися хеш цього коміту.
- Щоб дізнатися хеш, можна знову скористатися командою `git log` і подивитися на потрібний коміт. У моєму випадку це наступний коміт:

```
commit e41ae66ba3b20c6f7f4287305a306513c24788d1
Author: Leonid Podriz <leonidpodriz@gmail.com>
Date: Tue Sep 29 10:33:23 2020 +0300
```

Add fisrt code file

e41ae66ba3b20c6f7f4287305a306513c24788d1 – це хеш. Він генерується залежно від змін файлу автоматично.

Щоб повернути свій проект до стану цього коміту, виконайте команду `git checkout <hash>`:

```
C:\Users\38099\Desktop\git_lesson>git checkout
e41ae66ba3b20c6f7f4287305a306513c24788d1
Note: switching to 'e41ae66ba3b20c6f7f4287305a306513c24788d1'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-c` with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to `false`

HEAD is now at e41ae66 Add fisrt code file

- Не соромтеся читати результати команд. Вони ємно пояснюють усі тонкощі роботи з Git.
- Зауважте, що вміст файлу `main.code` повернувся до стану коміту.
- Для повернення до останніх змін використовуйте команду: `git checkout master`:

```
C:\Users\38099\Desktop\git_lesson>git checkout master
Previous HEAD position was e41ae66 Add fisrt code file
Switched to branch 'master'
```

`master` – ім'я гілки за замовчуванням. Перемикаючи імена гілок, ви потрапляєте на останню версію обраної гілки.

Що далі?

- Git має багато функцій. Щоб вивчити їх усі, знадобиться дуже багато часу. Ми знайомили вас із БАЗОВИМ функціоналом Git.
- Деяку частину функціоналу вам, можливо, ніколи не доведеться використовувати. Щось – навпаки, ви використовуватимете щодня.
- Якщо вам сподобалося працювати з Git і бажаєте опанувати його краще, рекомендую вивчити роботу з гілками.

Нами були розглянуті способи організації систем контролю версій, способи вирішення поставлених завдань перед цими системами, переваги та недоліки кожної з них, а також ознайомилися з історією системи контролю версій Git.

Закріплення матеріалу

- Що таке система контролю версій?
- Що таке Git?
- Навіщо потрібен коміт?
- Яка команда використовується, щоб створити git репозиторій?

Додаткове завдання

Завдання 1

Додайте кілька комітів після того, як виконаєте Завдання 1 з розділу «Самостійна діяльність учня».

Завдання 2

Вивчивши додаткові матеріали створити(згенерувати) файл `.gitignore`, який ігноруватиме вашу ОС та IDE.

Самостійна діяльність учня

Завдання 1

Створіть порожній проект і додайте файли до репозиторію

Завдання 2

Створіть 3 файли та внесіть зміни (в консолі):

- 1й: прізвище, ім'я;
- 2й: email;
- 3й: назва курсу.

Додайте файли до репозиторію.

Завдання 3

Створити структуру файлів:

geometry_lib

```
|—docs
|   |—instruction.txt
|
|—my_pyfiles
|   |—volumes.py
|   |—areas.py
|   |—perimeters.py
```

Завдання 4

Створити структуру файлів:

my_site

```
|—img
|   |—dog.jpg
|   |—cat.jpg
|   |—turtle.jpg
|—css
|   |—my_style.css
|—index.html
```

Рекомендовані ресурси

- [Робота с ветками в Git](https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D0%B2%D0%B5%D1%82%D0%BA%D0%B0%D0%BC%D0%B8)
<https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D0%B2%D0%B5%D1%82%D0%BA%D0%B0%D0%BC%D0%B8>
- Книга «Git для професійного програміста» (С. Чакон, Б. Штрауб)
- <https://www.toptal.com/developers/gitignore>

Документація:

- <https://git-scm.com/doc>
- <https://git-scm.com/docs/gitignore>

