



Microsoft Partner
Silver Learning

Git

Основи роботи з Git



ITVDSN
IT VIDEO DEVELOPERS NETWORK

Git

Після уроку обов'язково



Повторіть цей урок у відео форматі на [ITVDN.com](https://itvdn.com)

Доступ можна отримати через керівництво вашого навчального центру



Перевірте, як Ви засвоїли цей матеріал на [TestProvider.com](https://testprovider.com)

Git

Основи роботи з Git

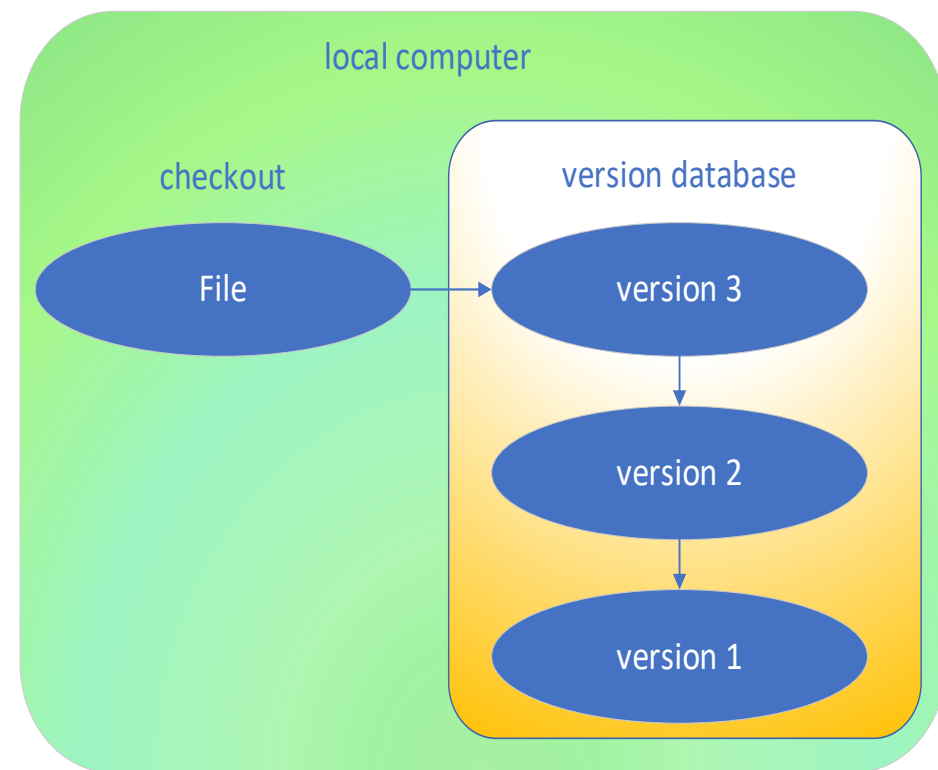
Що таке VCS?

Git

Що таке VCS?

Система контролю версій (**VCS (Version Control System)**) – це система, що широко використовується в процесі розробки ПЗ розробниками, яка записує зміни до файлу або перелік файлів у процесі розробки, котра дозволяє повернутися в необхідний момент до певної версії.

Насправді можна використовувати контроль версій практично для файлів будь-якого типу.

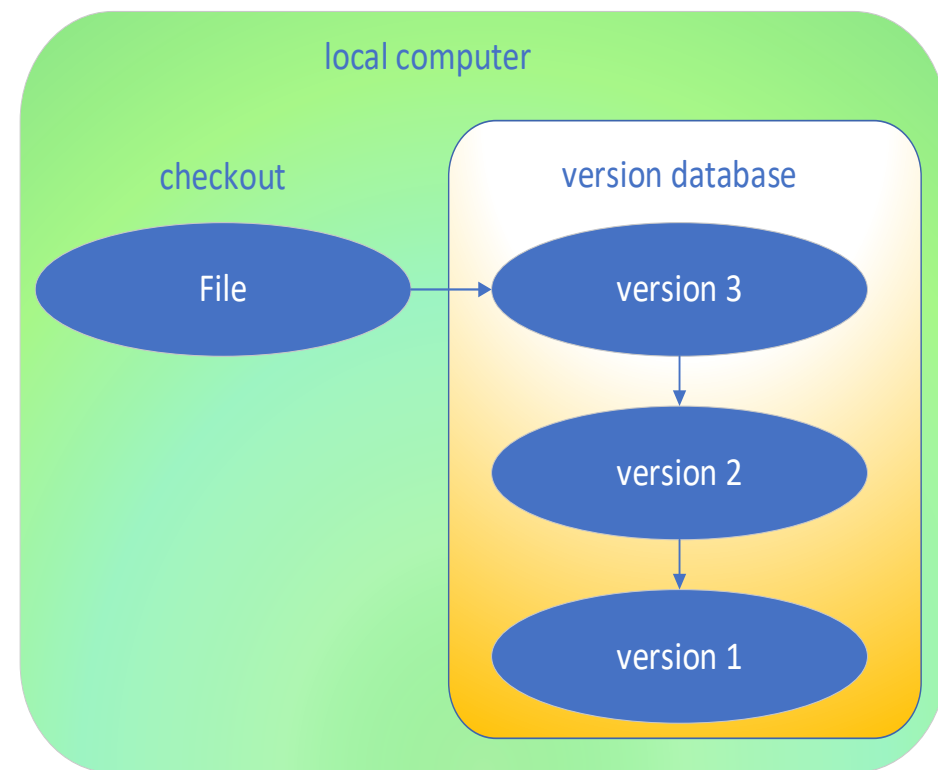


Git

Що таке VCS?

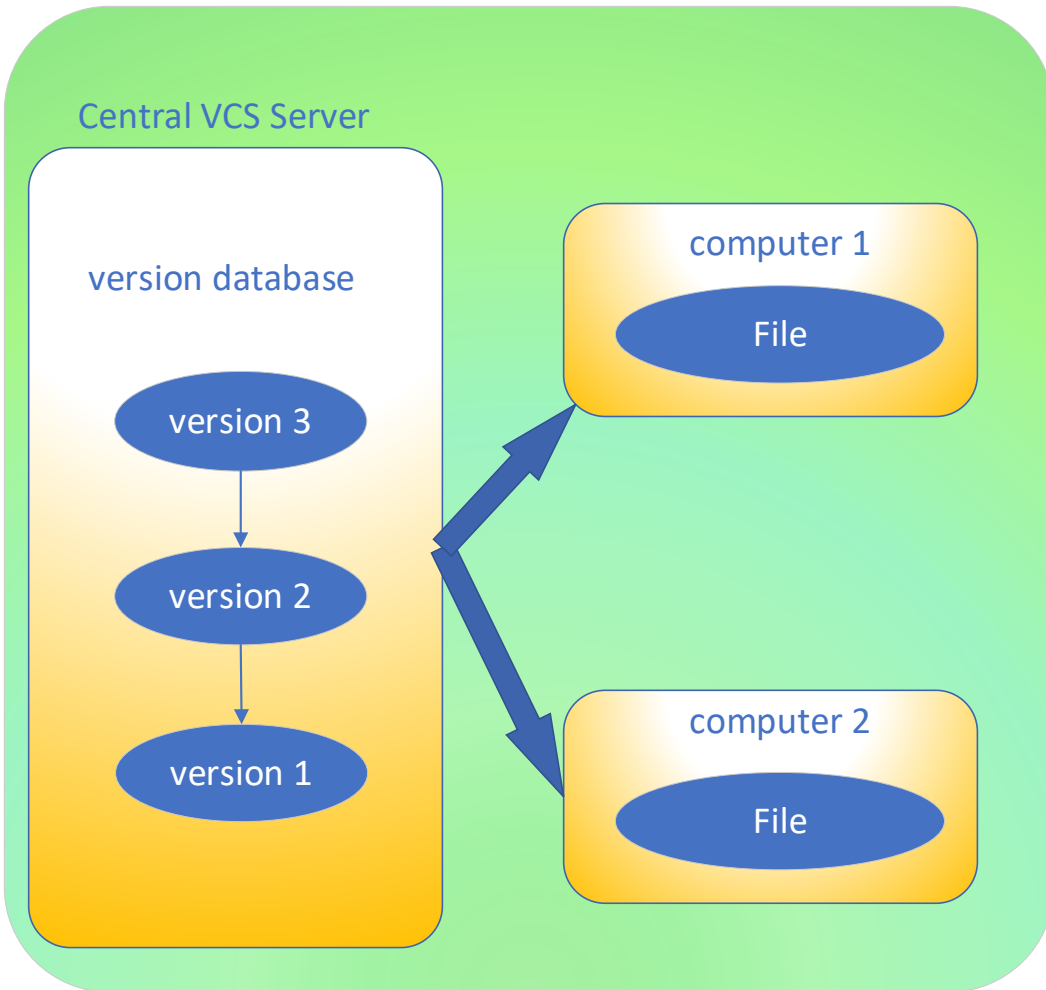
Дані системи дозволяють:

- зберігати різні версії одного й того самого файлу;
- у разі потреби відкотитися до більш ранніх версій;
- визначати, хто і коли здійснив яку-небудь зміну;
- здійснювати процес розробки кількох гілок одночасно.



Git

Види VCS



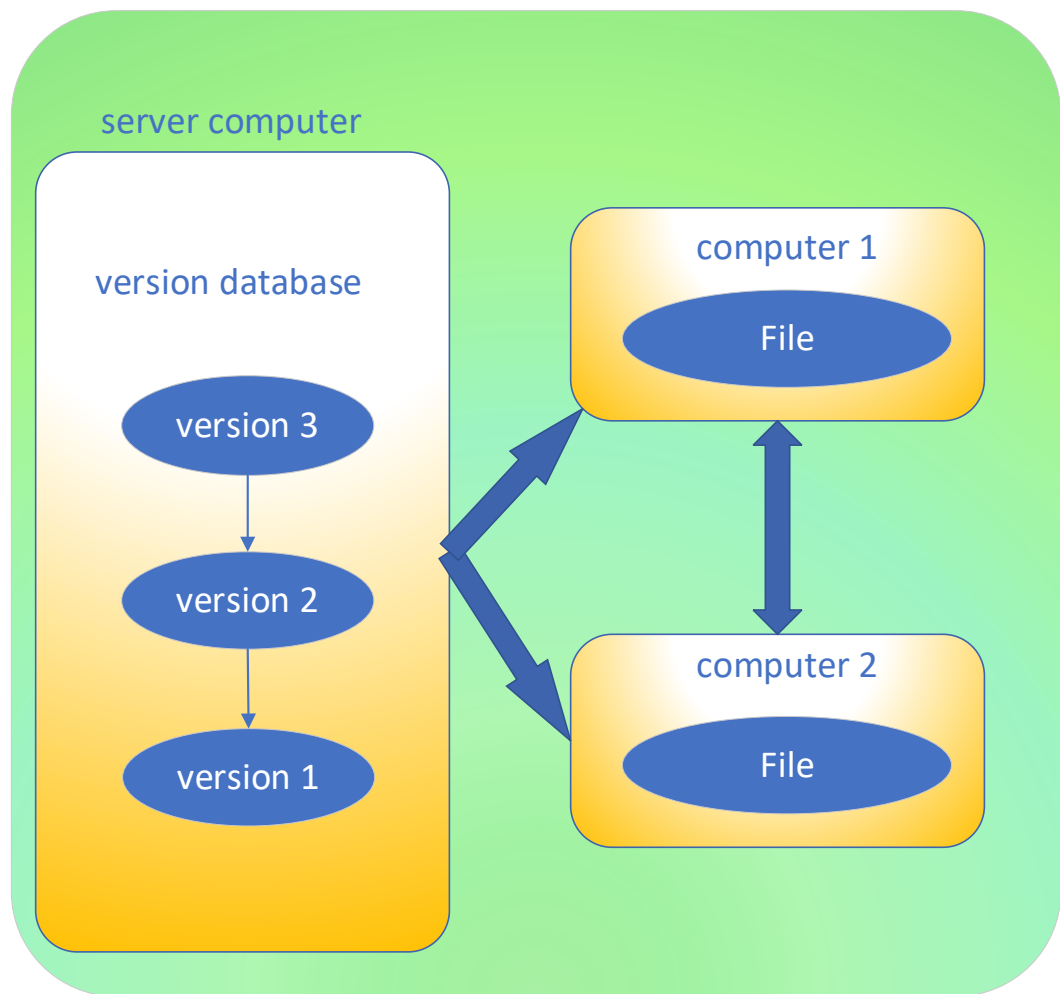
Централізовані CVCS (Central Version Control System) – призначені на вирішення основної проблеми локальної системи контролю версій.

Для організації такої системи контролю версій використовується єдиний сервер, який містить усі версії файлів. Клієнти, звертаючись до цього сервера, одержують файли з цього централізованого сховища.



Git

Види VCS



Розподілені DVCS (Distributed Version Control System) – мають на увазі, що клієнт викачає собі весь репозиторій повністю замість викачування конкретних файлів, які цікавлять клієнта. Якщо помре будь-яка копія репозиторію, це не призведе до втрати кодової бази, оскільки вона може бути відновлена з комп'ютера будь-якого розробника. Кожна копія є повним бекапом даних.

Усі копії є рівноправними і можуть синхронізуватися між собою.

Подібний підхід є реплікацією виду master-master.



git



mercurial



Bazaar



darcs

Порівняння систем контролю версій

- на DVCS можна все те саме, що і на CVCS;
- на DVCS простіше виконувати злиття гілок;
- на DVCS вся історія зберігається локально, можна працювати офлайн і робота загалом швидше;
- більш гнучка модель обміну змінами;
- розробники звикли до CVCS, потрібно переналаштовуватися;
- у CVCS нижче "порог входження" - для роботи з DVCS треба краще розуміти концепції контролю версій;
- найчастіше у світі використовується представник CVCS – SVN, а DVCS – Git.



Git

Основи роботи з Git

Що таке Git?

Git

Визначення

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

Встановлення

Щоб розпочати роботу з Git, його потрібно встановити. Завантажте встановлювач із [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.



Git

Початок роботи

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"
```

Git

Основи роботи з Git

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

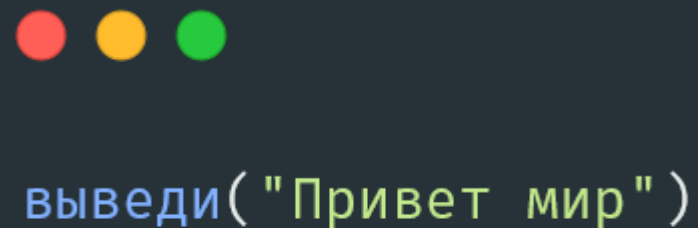
Перелік файлів у директорії: **ls**

Читання файлу: **cat** test.txt

Видалення файлу: **rm** test.txt

Підготовка робочої директорії

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



```
выведи("Привет мир")
```

Створення порожнього репозиторію

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

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`



Git

Створення файлу для додавання його до репозиторію

Створення README файлу та .gitignore:

```
touch README.md
```

```
touch .gitignore
```

Git

Зміна файлу для додавання його до репозиторію

Зміна README файлу та .gitignore:

```
nano README.md
```

```
nano .gitignore
```

Якщо файл не було створено, і необхідно зберегти його зміни - при виході (Ctrl + X) потрібно вказати його ім'я.

Git

Файл .gitignore

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.

Git

Файл .gitignore

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`.

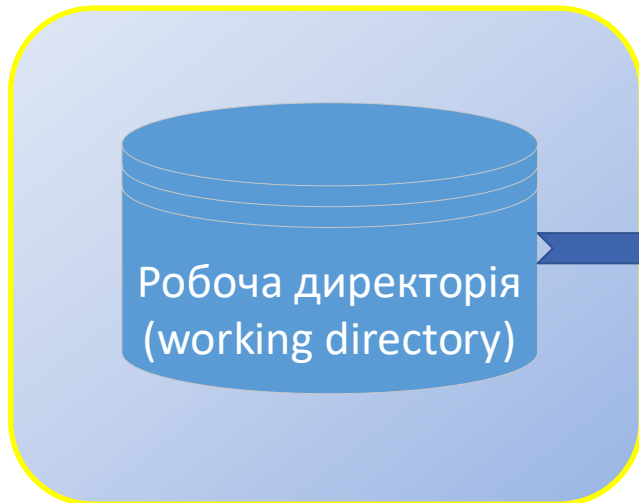
Ігноровані файли відстежуються у спеціальному файлі `.gitignore`, який реєструється в кореневому каталозі репозиторію. У Git немає спеціальної команди для вказівки файлів, що ігноруються: замість цього необхідно вручну відредагувати файл `.gitignore`, щоб вказати в ньому нові файли, які повинні бути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

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

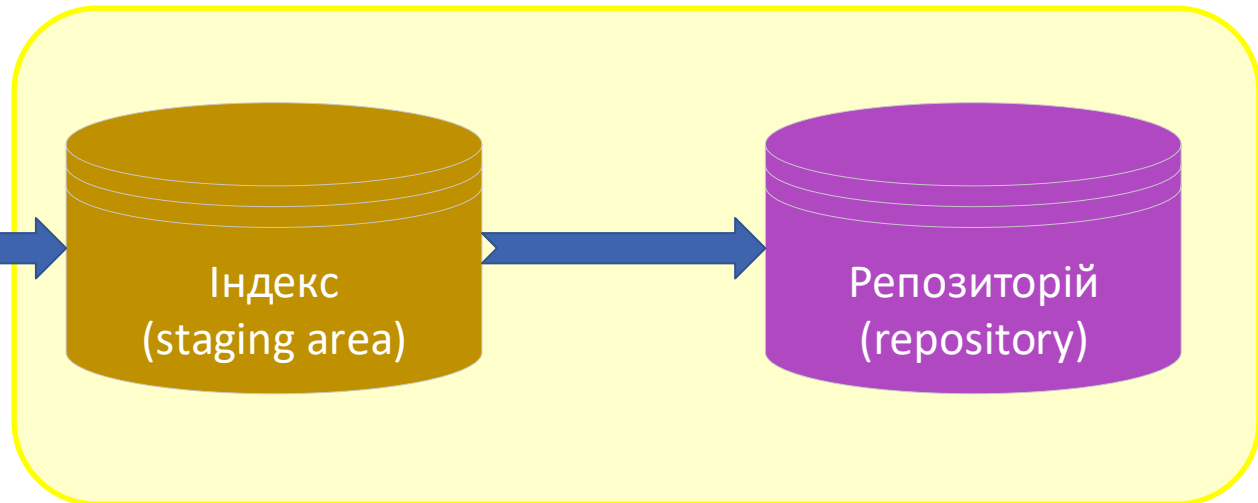
Основи роботи з Git

Області GIT

Інформація, яку
бачить користувач

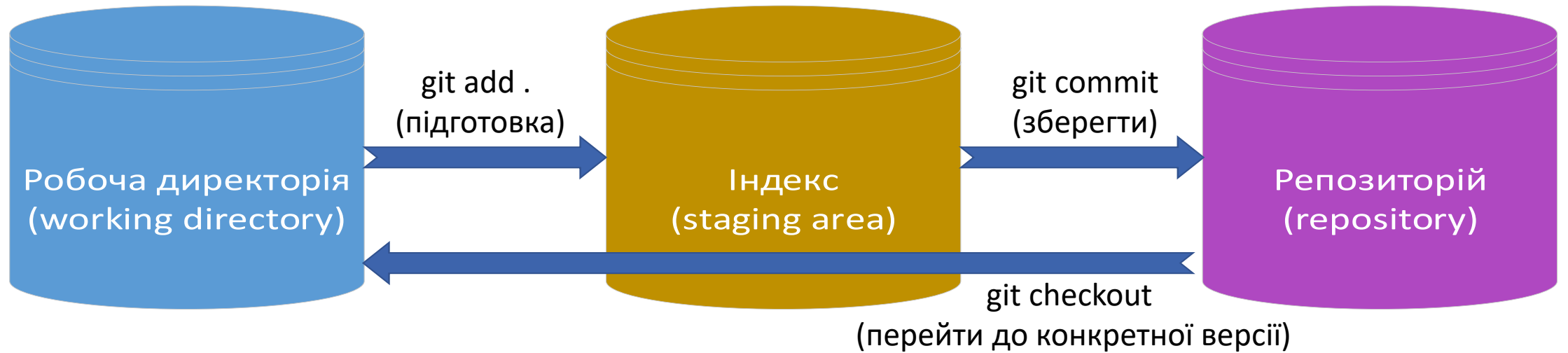


Інформація, яка прихована від користувача



Основи роботи з Git

Області GIT, в яких можуть зберігатися файли



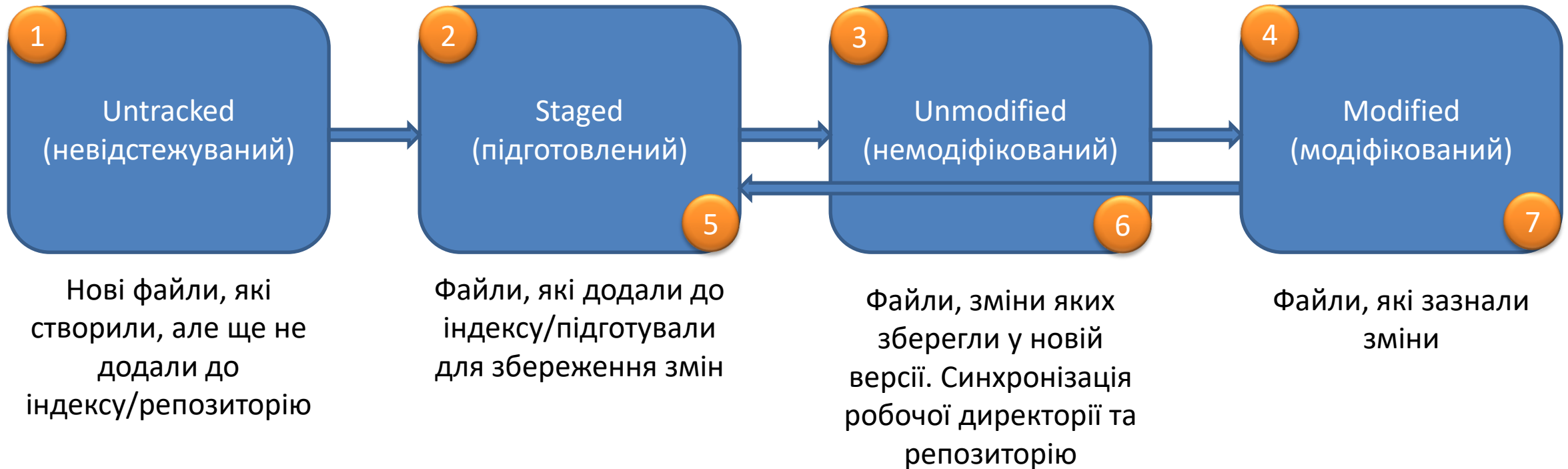
Містить видимі файли та папки(можна побачити за допомогою команди `ls`)

Додаються файли перед тим, як ми хочемо їх зберегти. Можна змінювати файли, які вже знаходяться у репозиторії.

Усі зміни записуються у репозиторій.
Розміщується в папці `.git`.
У папці `.git` є папка `objects`, де зберігаються усі версії файлів вашого проекту. Тут зберігаються коміти.

Основи роботи з Git

Статуси відстежуваних файлів у GIT



Основи роботи з Git

Типи об'єктів у GIT

Blob
(файл)

Об'єкти створюються для файлів. Різні версії файлів зберігаються у директорії objects.

Tree
(директорія)

Об'єкти являють собою директорії. Зберігає посилання на файли, які зберігаються у директорії, зберігаються у директорії objects.

Commit
(коміт)

Об'єкти створюються коли зберігається поточна версія проекту. Коміт зберігає метадані (інформація про автора та email).

Annotated Tag
(анотований тег)

Об'єкти створюються на будь-якому етапі проекту. Можна переміщуватися між різними версіями проектів за назвою тегів.

Кожний об'єкт у Git має унікальний ID (SHA1 хеш), який має фіксовану довжину – 160 біт (40 шістнадцяткових символів).

Додавання файлу до репозиторію

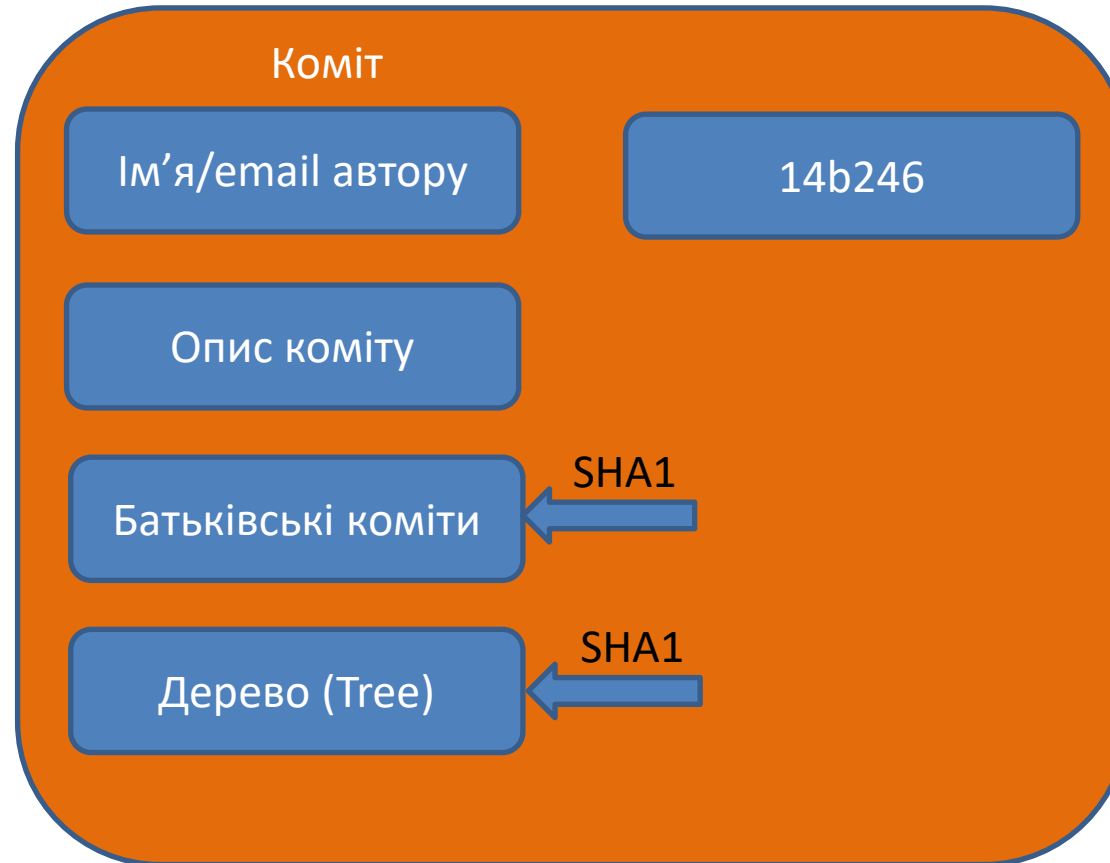
Створення репозиторію не означає, що Git відразу ж почав стежити за всіма файлами в папці. Файли потрібно додати вручну. Це дає контроль над файловою структурою Git репозиторію і гарантує, що жодні конфіденційні дані не потраплять у відкритий доступ без вашої згоди.



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

Основи роботи з Git

Структура коміту GIT



Перевірка стану репозиторію

Команда **git status** повертає інформацію про репозиторій:

- скільки файлів змінено з останнього коміту
- скільки файлів було додано до файлу, але не додано до репозиторію
- які файли були видалені
- і так далі.



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

Git

Зміни

Змінімо код у main.code:

```
выведи("Привет, Мир")
```

До коду додана кома і "Мир" написано з великої літери. Як на це відреагує Git?

```
> 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


Основи роботи з Git

Обробка змін

Git

Додати зміни

Виконайте команду git, аби проіндексувати зміни. Перевірте стан:



```
> git add main.code
> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file> ..." to unstage)
       modified:   main.code
```

Git

Коміт

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 комітом, тоді як зміни в **math.code** логічно не пов'язані з першими двома файлами і повинні йти окремим комітом.

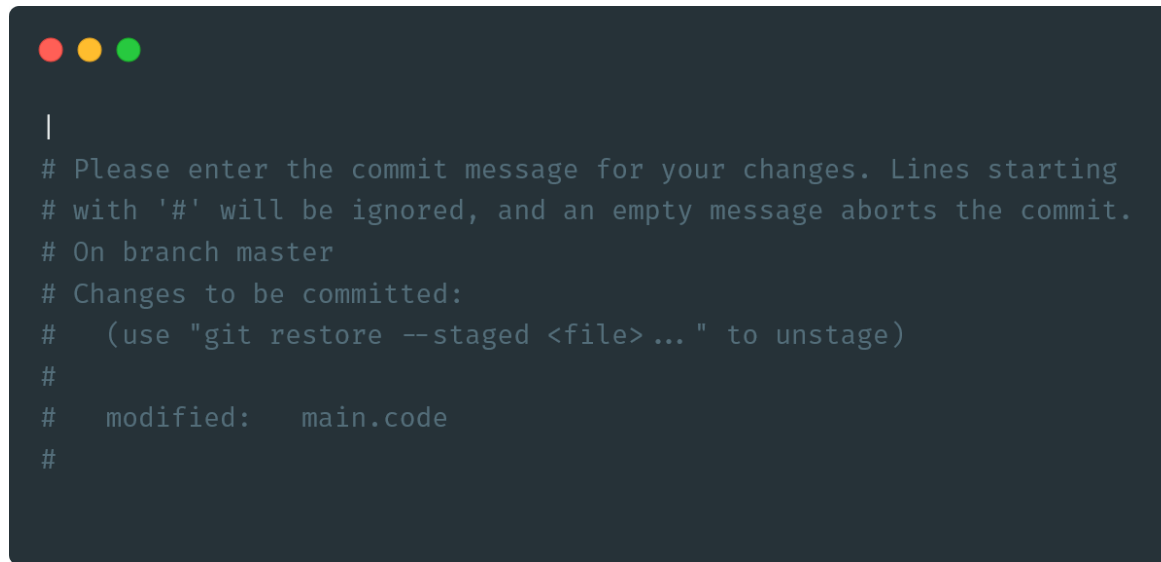
```
git add main.code
git add module.code
git commit -m "Changes for main and module"
```

```
git add math.code
git commit -m "Unrelated change to math"
```

Git

КОМІТИМО ЗМІНИ

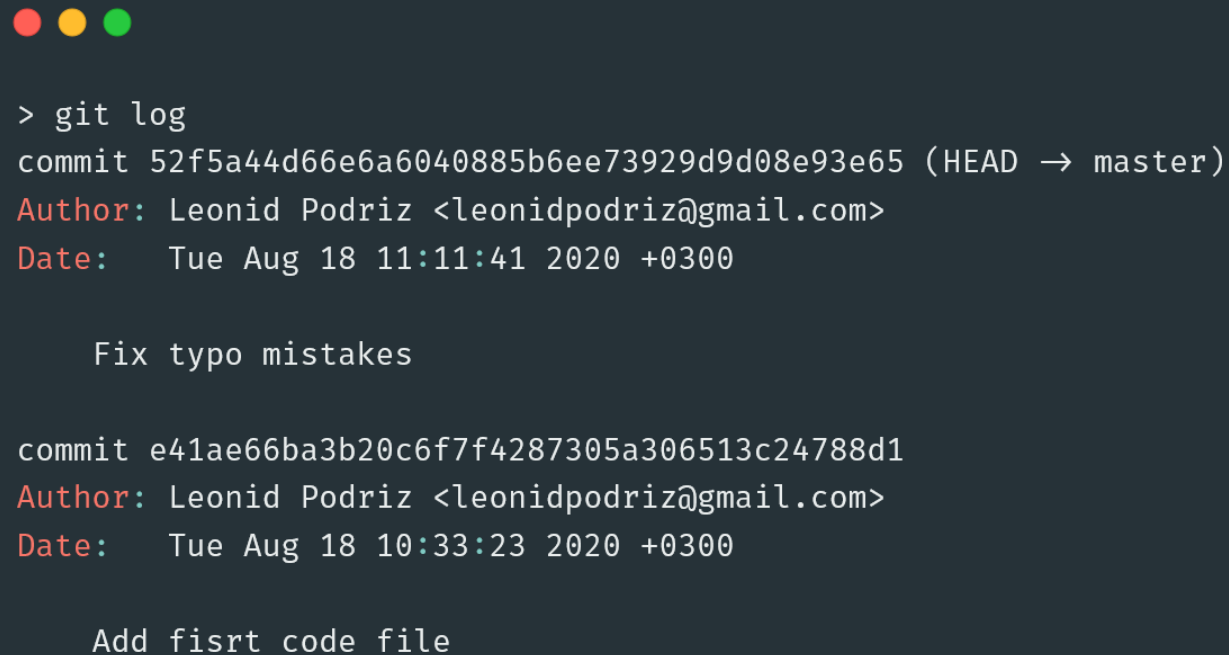
Команда **git commit** дозволить вам інтерактивно редагувати коментарі для коміту. Якщо ви опустите мітку *-m* з командного рядка, git перенесе вас до редактора на ваш вибір.



Git

Історія

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



```
> git log
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
```

Git

Що далі?

Git має багато функцій. Щоб вивчити все, знадобиться багато часу. Ми ознайомилися з **БАЗОВИМ** функціоналом Git.

Деяку частину функціоналу вам, можливо, ніколи не доведеться використовувати. Щось – навпаки, ви використовуватимете щодня.

Якщо вам сподобалося працювати з Git і ви бажаєте опанувати його краще, раджу вивчити роботу з гілками. Також зверніть увагу на рекомендації, де ви знайдете матеріал для закріплення і нові теми, які можуть стати вам у нагоді та полегшити роботу.

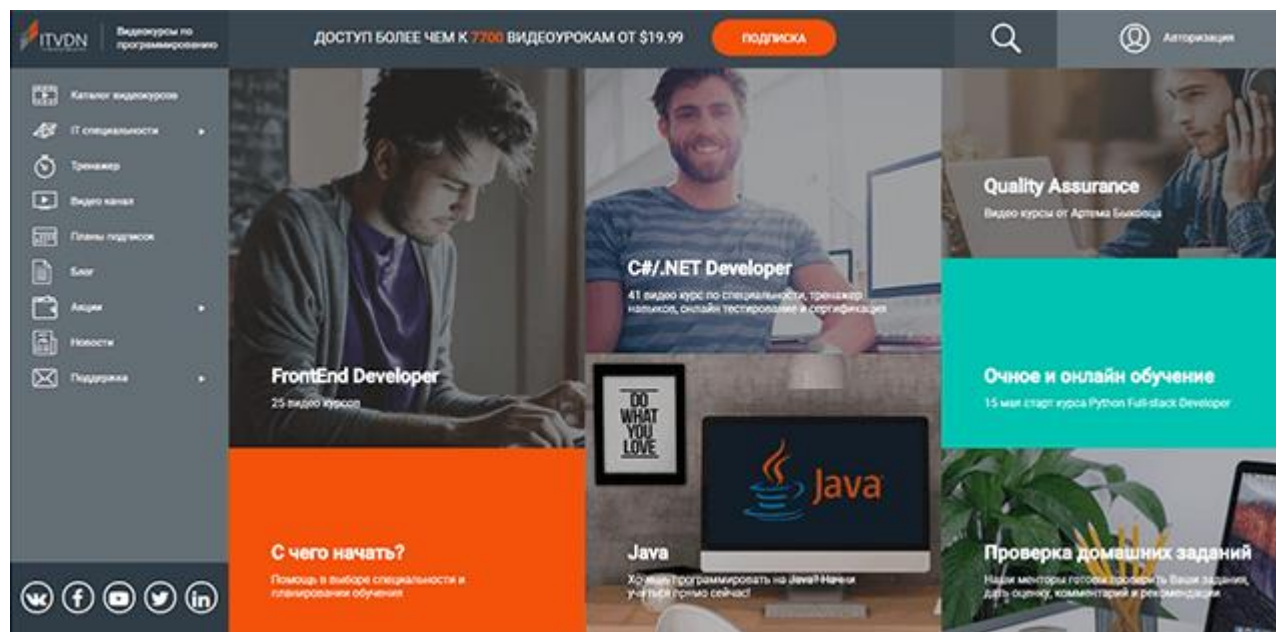
Git

Основи роботи з Git

Практична частина

Дивіться наші уроки у відео форматі

ITVDN.com



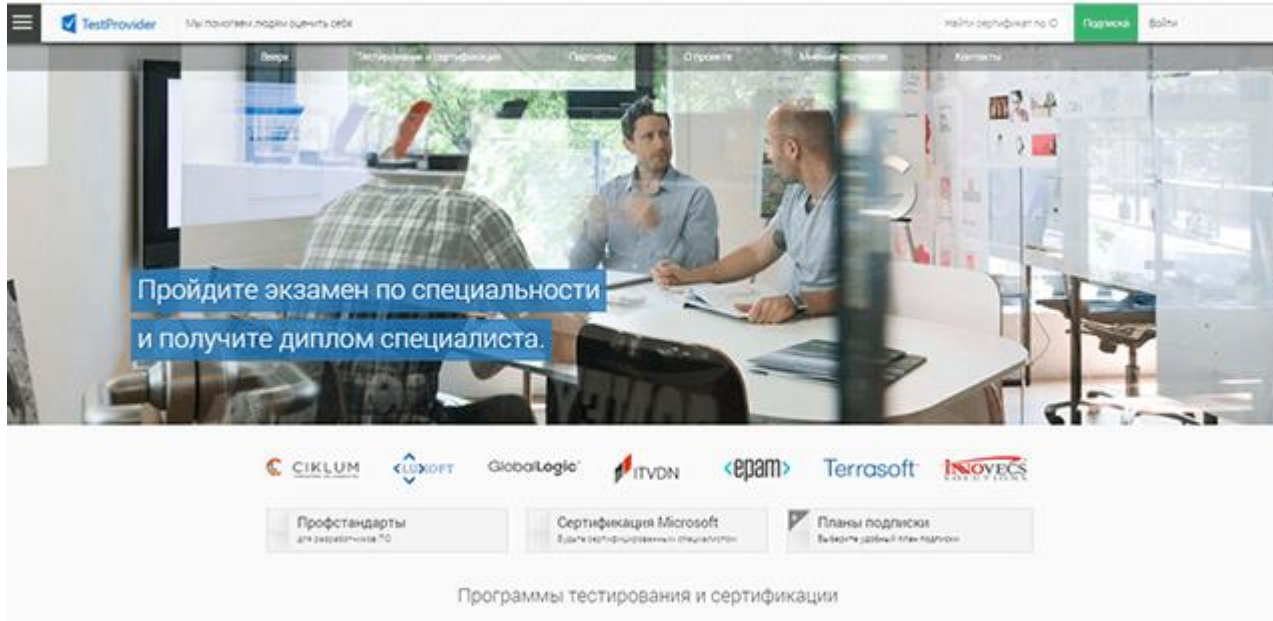
Перегляньте цей урок у відео форматі на освітньому порталі [ITVDN.com](https://itvdn.com) для закріплення пройденого матеріалу.

Курси записані сертифікованими тренерами, які працюють у навчальному центрі CyberBionic Systematics, та іншими висококваліфікованими розробниками.



Перевірка знань

TestProvider.com



TestProvider – це online сервіс перевірки знань з інформаційних технологій. За його допомогою Ви можете оцінити Ваш рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і для загальної оцінки знань IT-спеціаліста.

Після кожного уроку проходите тестування для перевірки знань на [TestProvider.com](https://testprovider.com)

Успішне проходження фінального тестування дозволить Вам отримати відповідний Сертифікат.



Q&A

Інформаційний відеосервіс для розробників програмного забезпечення

