

Робота з Git та GitHub

адаженда

01 / Що таке системи контролю версій?

02 / Що таке git?

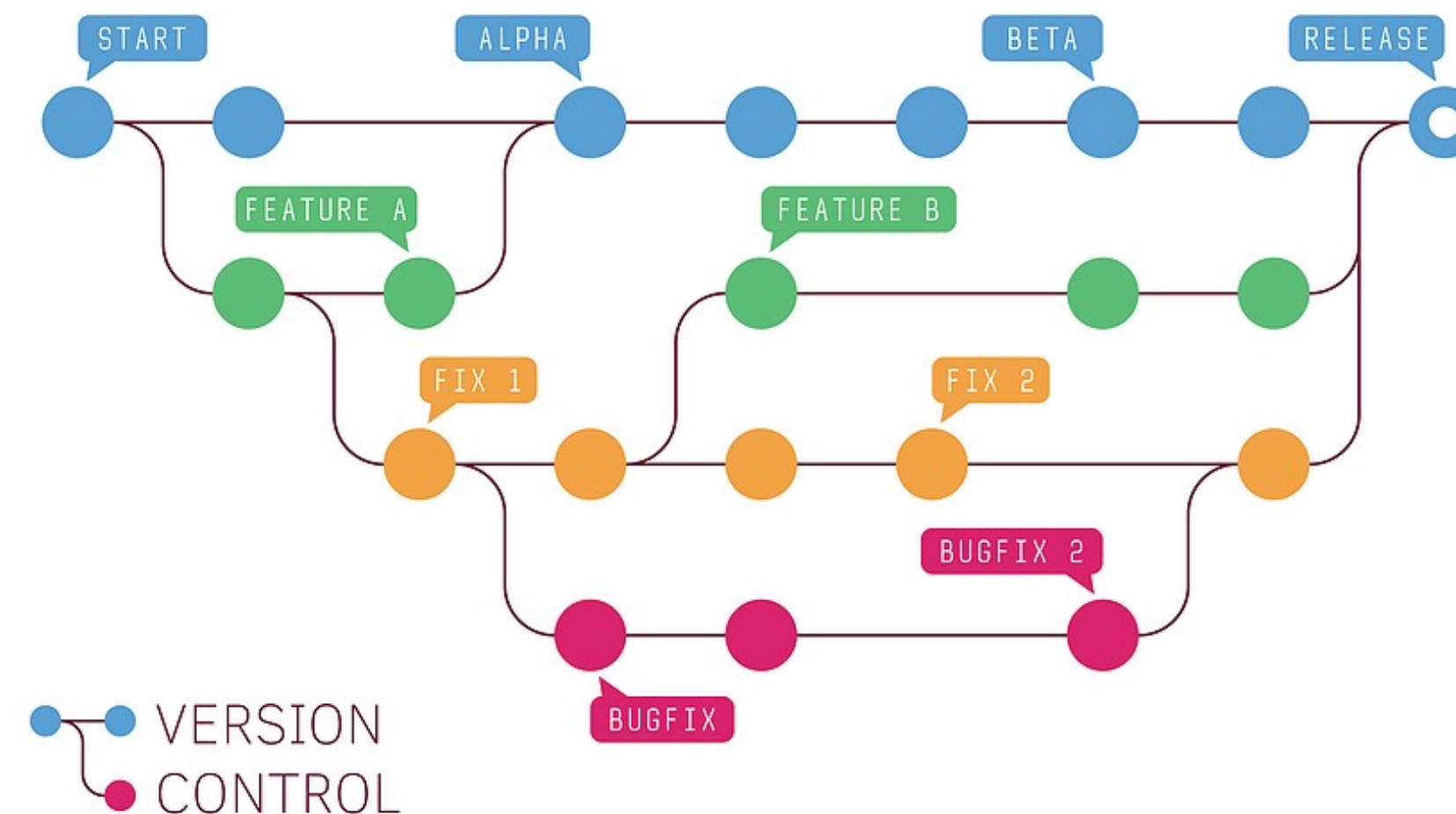
03 / Що таке GitHub?

04 / Робота з гілками

Що таке системи контролю версій?

Системи контролю версій

Системи контролю версій — це інструменти, які дозволяють кільком людям працювати з одним і тим же файлом або набором файлів, при цьому вони відстежують всі зміни, які робляться у цих файлах.



Чому системи контролю версій є важливими?

- Відновлення попередніх версій.
- Співпраця. Кілька людей може одночасно працювати над одним проектом, і системи контролю версій допомагають уникнути конфліктів у файлі.
- Відстеження історії. Ви завжди можете подивитися, хто, коли і які зміни зробив, що є корисним при пошуку помилок або при визначенні причини певних рішень.
- Гілки (Branching) та злиття (Merging). Ці функції дозволяють розробникам працювати над новими функціями або виправленнями, не турбуючись про основну версію програми. Коли робота завершена, зміни можна легко інтегрувати назад в основний код.

Системи контролю версій

Найпопулярнішою системою контролю версій на сьогоднішній день є [Git](#), але існують і інші, такі як [Mercurial](#), [Subversion](#) та [CVS](#).

[Git](#) часто використовується разом із онлайновими платформами, такими як [GitHub](#), [GitLab](#) чи [Bitbucket](#), які надають додаткові можливості для співпраці, відстеження проблем та інтеграції з іншими інструментами.

Q&A

Що take git?

Git

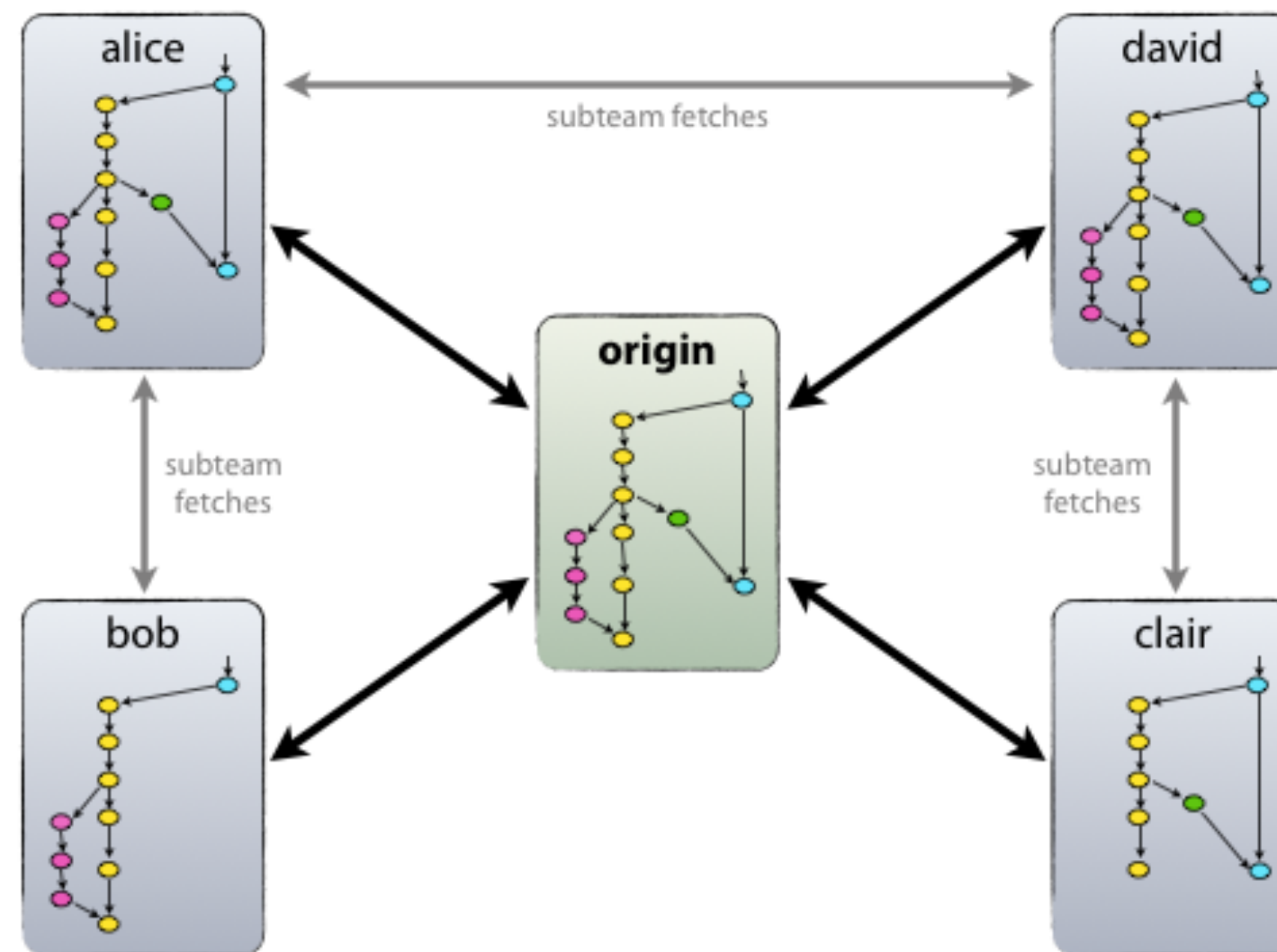
Git — це розподілена система контролю версій, яка допомагає командам та окремим розробникам відстежувати зміни в коді в процесі часу.

Він був розроблений Лінусом Торвальдсом в 2005 році для розробки ядра Linux, але згодом став популярним у розробницькій спільноті завдяки своєму ефективному підходу до контролю версій.



Переваги системи git

Розподілена природа: Відмінність **Git** від багатьох інших систем контролю версій полягає в тому, що він є розподіленим. Це означає, що кожен користувач має локальну копію репозиторія, яка містить всю історію змін. Це дозволяє працювати офлайн і забезпечує додатковий рівень безпеки.



Переваги системи git

Простота: Незважаючи на свою потужність та гнучкість, основні команди **Git** вивчаються досить легко, особливо для початківців.

Завдяки популярності **Git** існує велика кількість ресурсів, керівництв та документації для користувачів різного рівня досвіду, що спрощує вивчення та використання системи.

Переваги системи git

Розповсюдженість: Git став де-факто стандартом для контролю версій у багатьох організаціях та спільнотах. Це означає, що більшість розробників знайомі з Git, і вам не потрібно буде вчити нову систему контролю версій при переході між проектами або роботодавцями.

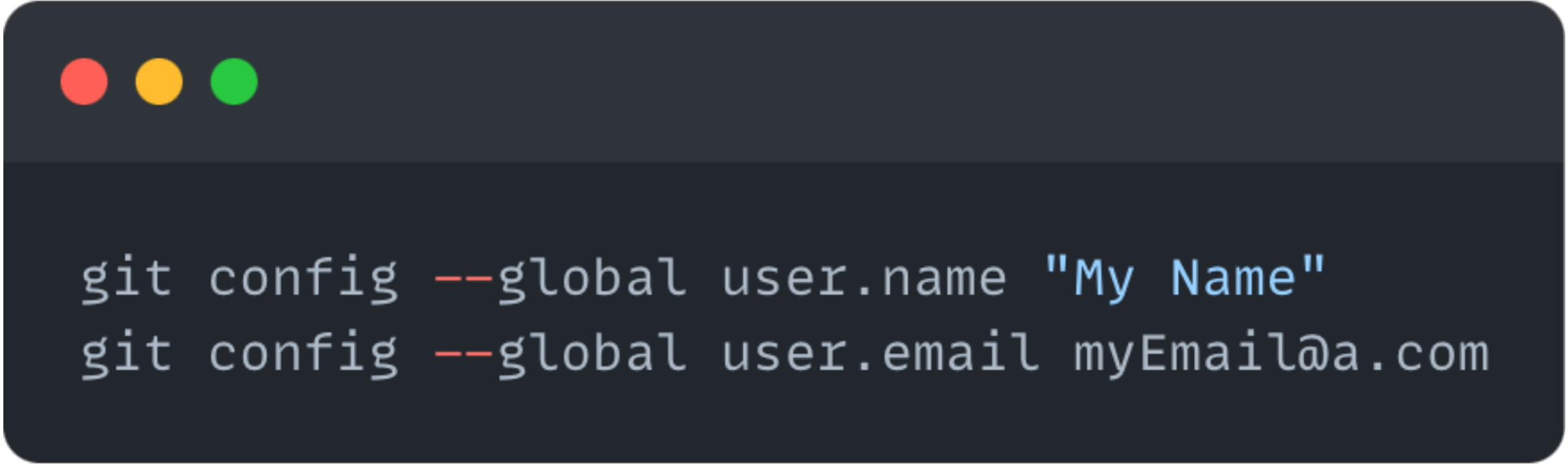
Інсталювання Git

Для початку **Git** треба інсталювати на вашу систему.

- Офіційний сайт Git: <https://git-scm.com/>
- Windows: <https://gitforwindows.org/>

Інсталювання Git

Перед роботою треба дещо налаштувати, а саме ім'я користувача та адресу електронної пошти:



```
git config --global user.name "My Name"  
git config --global user.email myEmail@a.com
```

Створення репозиторія

Репозиторій — це місце зберігання, яке містить всі файли проекту, його історію змін, а також додаткову інформацію, необхідну для роботи системи контролю версій.



Статус репозиторія

Команда `git status` використовується в `Git` для відображення стану файлів у вашому репозиторії. Вона показує, які файли були змінені в робочій директорії, які зміни були додані, але ще не були зафіксовані (committed), і які файли ще не відстежуються Git'ом.



Проміжна область

Проміжна область (часто називається "областю додавання" або "staging area" англійською) у [Git](#) – це проміжний етап між робочою директорією та вашим репозиторієм.

У вашій робочій директорії ви вносите зміни до файлів. Коли ви готові зберегти ці зміни, ви "додаєте" змінені файли до проміжної області використовуючи команду [git add](#).

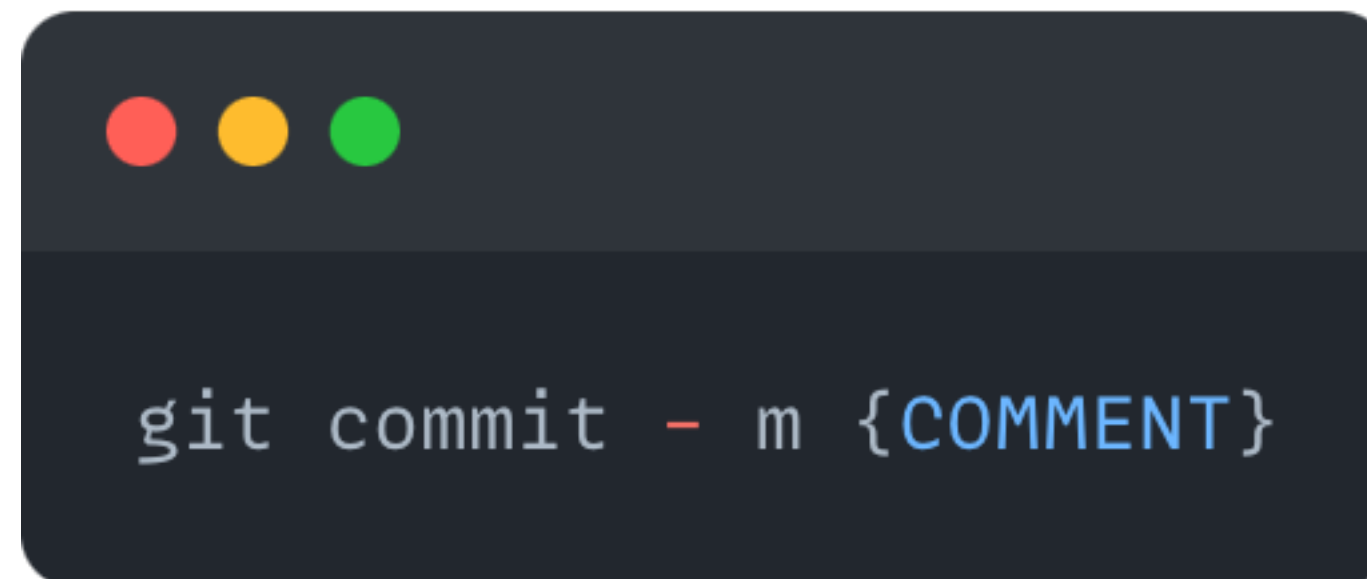
A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. It contains two lines of text: 'git add {FILENAME}' and 'git add .' on separate lines.

```
git add {FILENAME}
git add .
```

Коміт

Коміт — це "запис" або "фіксація" змін, внесених у файл(и) в репозиторії.

Кожен коміт включає в себе вказівник на певний набір змін, метаінформацію (таку як автор, дата та коментар до коміту), а також посилання на попередній коміт(и), що дозволяє відслідковувати історію змін.

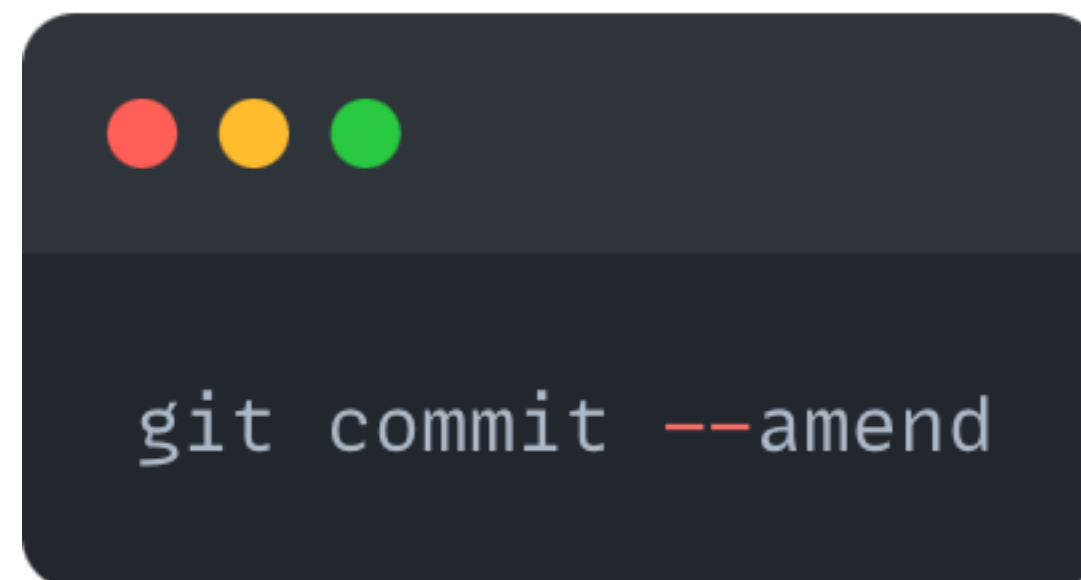
A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. The terminal displays the command `git commit -m {COMMENT}` in a light blue monospace font.

```
git commit -m {COMMENT}
```

Редагування коміта

Якщо ви просто зробили коміт і зрозуміли, що забули додати якийсь файл або хочете змінити повідомлення коміту, ви можете використовувати наступну команду.

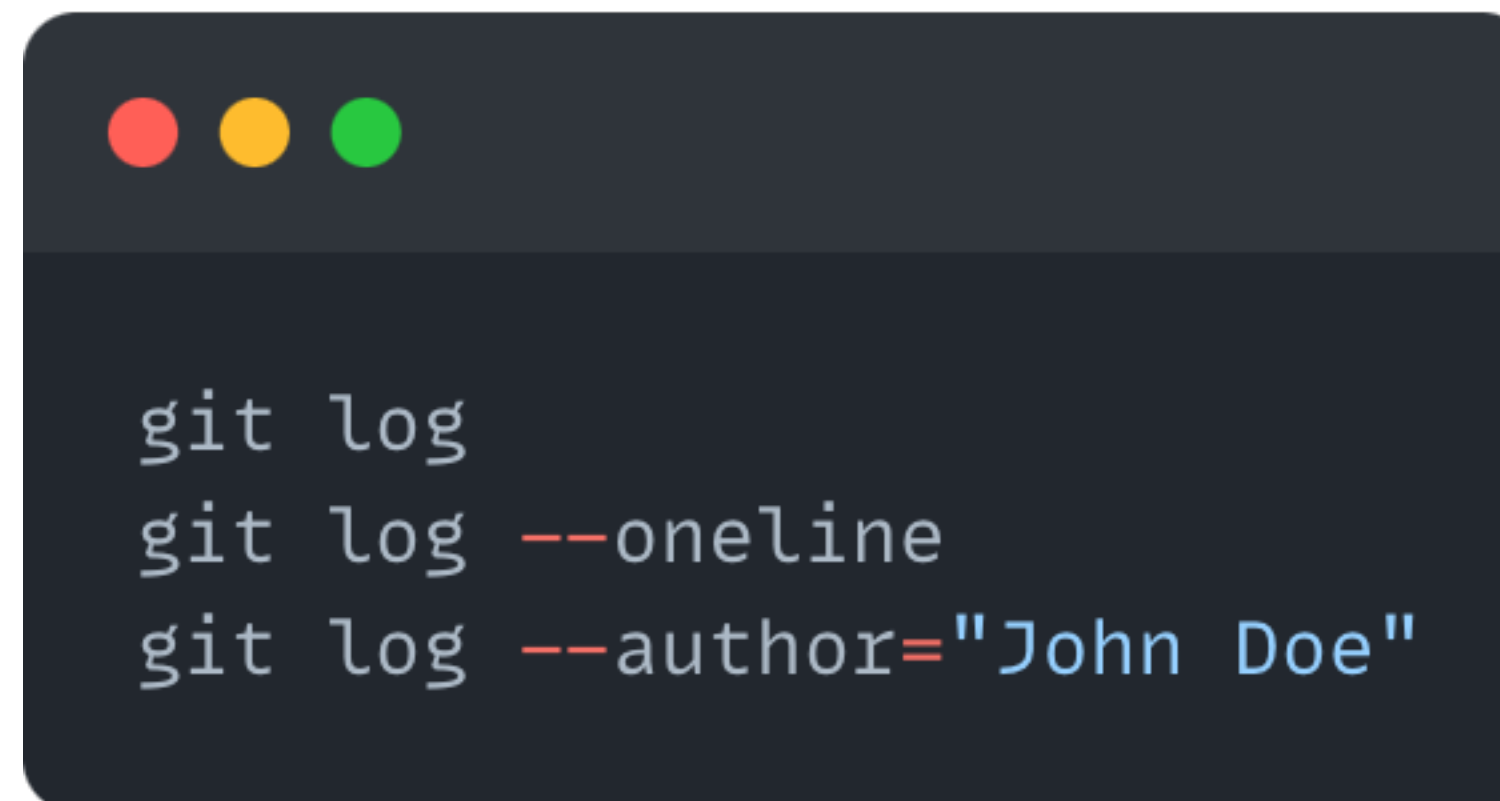
Ця команда дозволить вам внести зміни в **останній** коміт. Якщо ви хочете додати новий файл, спочатку виконайте **git add** для цього файла, а потім цю команду.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. The text 'git commit --amend' is displayed in a light gray monospace font.

```
git commit --amend
```

Історія комітів

Щоб отримати історію комітів в [Git](#), ви можете використовувати команду [git log](#). Ця команда показує список комітів для поточної гілки в хронологічному порядку (найновіший коміт вверху).

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. It contains three lines of text representing Git commands.

```
git log
git log --oneline
git log --author="John Doe"
```

.gitignore

`gitignore` — це спеціальний файл в `Git`, який допомагає визначити, які файли чи директорії слід ігнорувати при фіксаціях (комітах) в репозиторії. Інакше кажучи, файл `.gitignore` дозволяє вказати, які файли не мають бути відслідковані `Git`.

Q&A

Що таке GitHub?

GitHub

GitHub — це веб-сервіс, що базується на системі контролю версій Git, і дозволяє розробникам співпрацювати над проектами. **GitHub** став найбільш популярним хмарним сервісом для зберігання коду, співпраці над проектами та іншими завданнями.



Віддалений репозиторій

Віддалений репозиторій дозволяє кільком користувачам співпрацювати над одним проектом, а також служить резервною копією вашого коду.

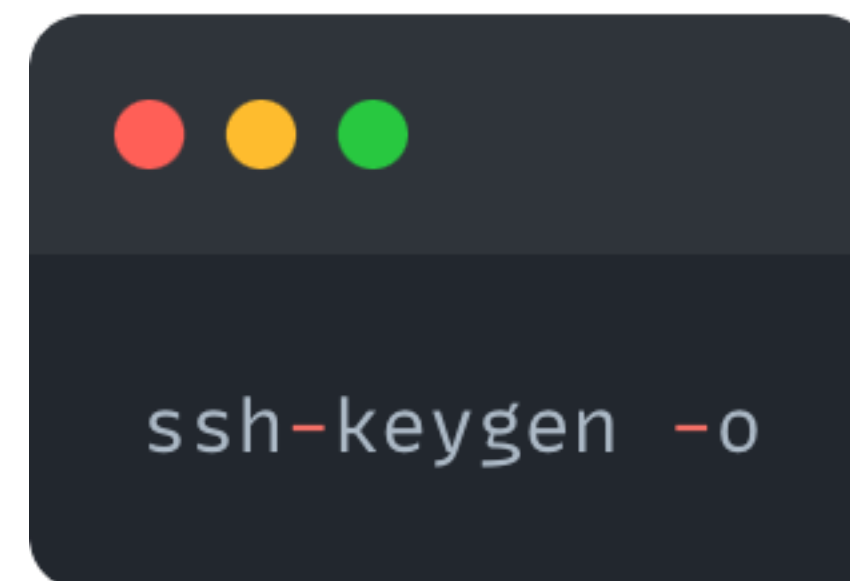
Віддалені репозиторії роблять можливим спільне використання коду та співпрацю між розробниками. Розробники можуть "витягувати" (pull) оновлення з віддаленого репозиторію та "завантажувати" (push) свої зміни назад у нього.



Налаштування SSH ключа

Для того щоб надати відкритий ключ, кожен користувач у системі повинен його згенерувати, якщо цього вже не було зроблено раніше. Цей процес аналогічний у всіх операційних системах.

Для того щоб згенерувати ключ, потрібно виконати наступну команду:

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. The text 'ssh-keygen -o' is displayed in a light gray monospace font.

```
ssh-keygen -o
```

Налаштування SSH ключа

Для того щоб побачити згенерований ключ потрібно виконати наступну команду.



```
cat ~/.ssh/id_rsa.pub
```

Якщо у вас операційна система Windows, то файл з ключем можна знайти таким чином:
C:\Users\UserName\.ssh\id_rsa.pub (можна відкрити за допомогою VS Code)

"Клонування" репозиторію

Щоб створити локальний репозиторій (копію віддаленого) необхідно виконати наступну команду:

```
git clone {REPOSITORYLINK}
```

Відправлення у віддалений репозиторій

Щоб відправити дані у віддалений репозиторій, потрібно виконати наступну команду:



Зміни з віддаленого репозиторію

Для того, щоб стягнути актуальні зміни з віддаленого репозиторію, необхідно виконати команду:

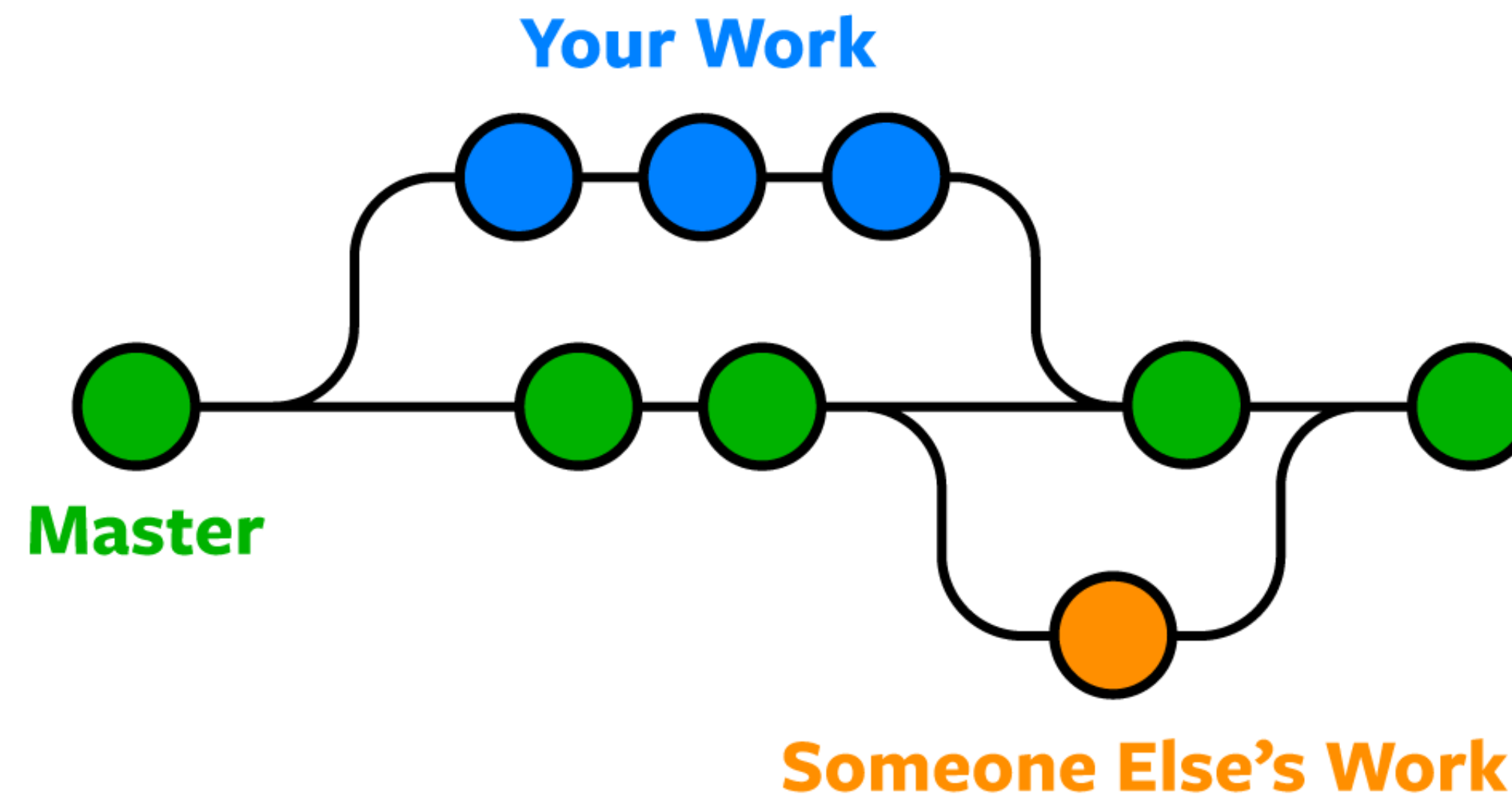


Q&A

Робота з гілками

Гілка (branch)

Гілки допомагають ізольовано та одночасно працювати над різними функціями чи корекціями без впливу на основний код.

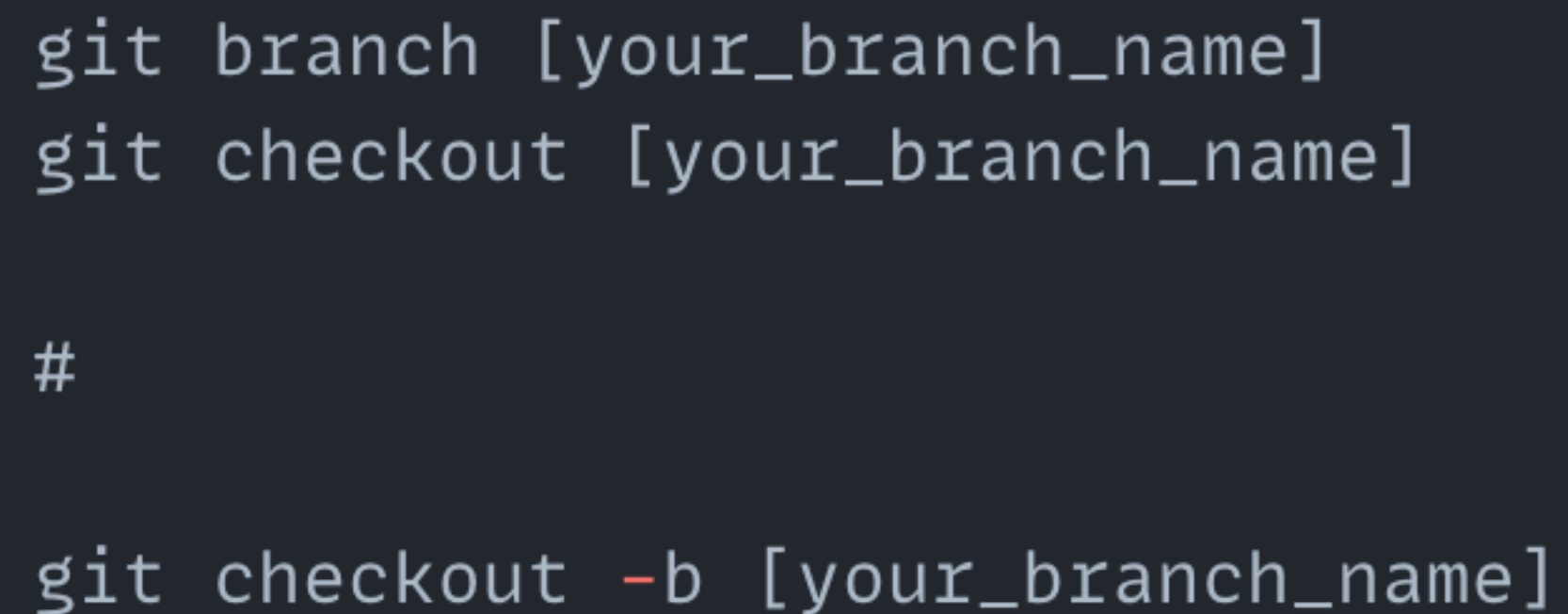


Master

Головна гілка (часто називається "**master**" або "**main**") залишається стабільною, так як вся нова робота виконується в окремих гілках. Це забезпечує, що неперевірені зміни не потраплять в основний код до того моменту, як вони будуть тщательно перевірені та визнані готовими.

Створення гілки

Щоб створити нову гілку, ви можете використовувати наступні команди:



```
git branch [your_branch_name]
git checkout [your_branch_name]

#

git checkout -b [your_branch_name]
```

Злиття гілок

Злиття гілок (merging) — це процес об'єднання змін з однієї гілки в іншу. Найчастіше злиття використовується для об'єднання змін з робочої (або функціональної) гілки назад до основної гілки.

```
# Перемкніться на гілку, в яку ви хочете злити зміни
git checkout main

# Виконайте команду злиття
git merge [your_branch_name]
```

Конфлікти

Іноді, коли ви намагаєтеся злити гілки, можуть виникнути конфлікти.

Це трапляється, коли в обох гілках були зроблені зміни в одних і тих самих рядках файлу. Git не зможе автоматично вирішити, яку версію залишити, тому вам потрібно буде вручну виправити конфліктні рядки, а потім додати виправлений файл та завершити процес злиття.

Перехід на конкретний коміт в історії

Ви можете зробити це за допомогою команди `git checkout`. Вам потрібно мати на увазі хеш (або частину хеша) коміту, до якого ви хочете перейти.

При виконанні цієї команди ви перейдете у так званий "detached HEAD" стан. Це означає, що ви перебуваєте не на кінці гілки, а на конкретному коміті.

У цьому стані ви можете переглядати код, запускати його тощо, але якщо ви зробите які-небудь зміни та спробуєте зберегти їх новим комітом, Git порекомендує вам створити нову гілку, щоб не втратити зміни.

Відміна змін

Є два основні шляхи відміни змін в Git - перший це використувувати `git reset` й інший це `git revert`

`git reset` відмінює зміни переміщуючи вказівник гілки назад в історії на старіший коміт. В принципі, можна вважати що це певне "переписування історії;" Коли `git reset` перемістить гілку назад буде здаватися, що відмінений коміт взагалі ніколи не створювався.

Команда `git revert` створює новий коміт, який ефективно "відмінює" зміни попереднього коміту. Це безпечний спосіб відміни змін, оскільки він не перезаписує історію.

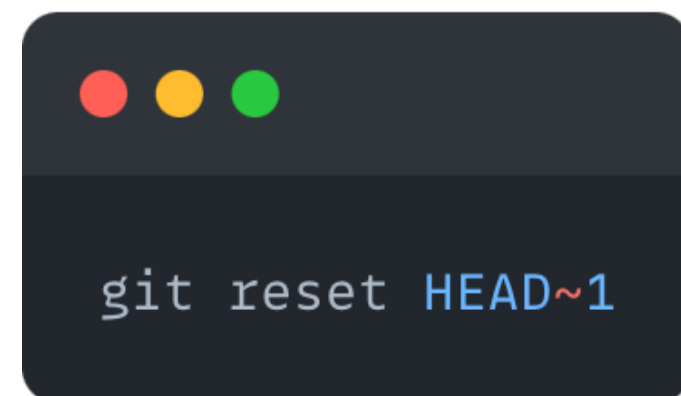
Редагування коміта

Для цього вам потрібно скористатися інтерактивним режимом git rebase:

```
git rebase -i HEAD~[number_of_commits]
```


Видалення коміта

Якщо ви потребуєте видалити останній коміт з вашої локальної гілки. Ця команда зсуне "голову" вашої гілки на один коміт назад, тобто ефективно "видаливши" останній коміт. Проте зміни з коміта залишатимуться в вашій робочій директорії.



Щоб видалити конкретний коміт із середини історії, ви можете скористатися інтерактивним режимом [git rebase](#). У редакторі, який відкриється, просто видаліть рядок, що відповідає коміту, який ви хочете видалити, а потім збережіть і вийдіть.

cherry-pick

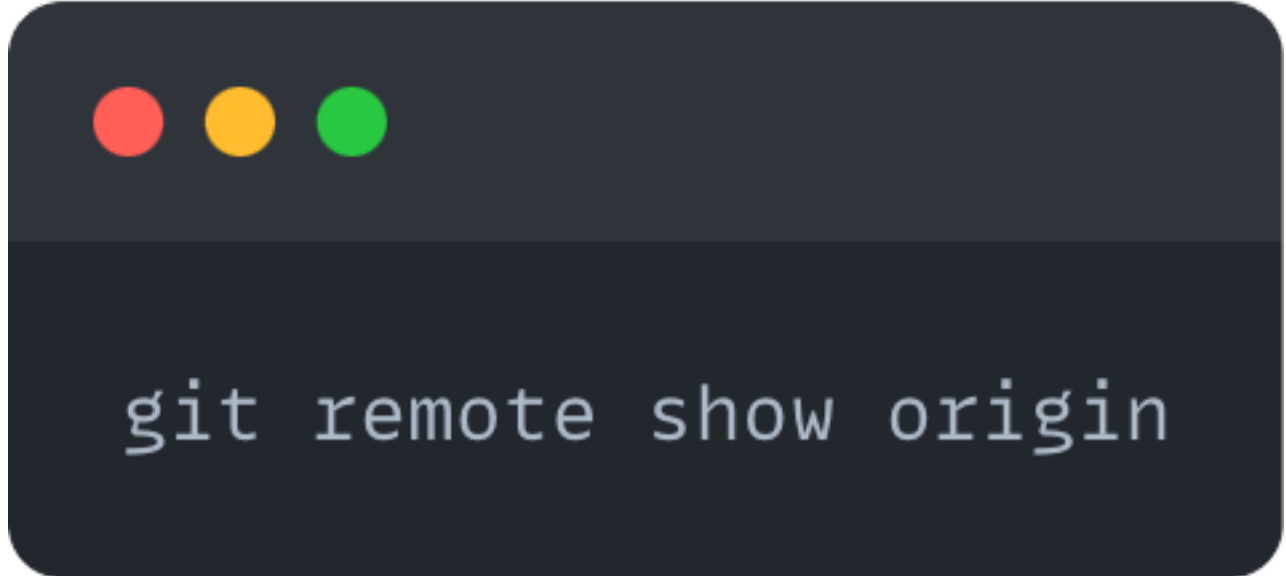
cherry-pick — це команда в Git, яка дозволяє вам вибрати конкретний коміт з однієї гілки та застосувати його до іншої. Це корисно, наприклад, коли вам потрібно застосувати певний багфікс або зміну з однієї гілки до іншої, не зливаючи при цьому всі зміни з оригінальної гілки.



```
git cherry-pick [commit_hash]
```

Віддалений репозиторій

Дізнатися імена та гілки віддаленого репозиторія
можна за допомогою команди:



```
git remote show origin
```

Pull request

"Pull Request" (або PR) у GitHub — це механізм для внесення змін у репозиторій, який забезпечує можливість обговорення та перевірки цих змін до їх злиття.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: my-patch-1 ✓ Able to merge. These branches can be automatically merged.

Choose a head ref

my

my-patch-1

myarb-patch-1

my

GitHub Desktop

GitHub Desktop — це графічний інтерфейс (GUI) для Git, розроблений і супроводжуваний компанією GitHub. Він надає візуальний інтерфейс для основних операцій Git, таких як клонування репозиторіїв, внесення змін, створення комітів, переключення між гілками та інше.

<https://desktop.github.com/>

GitHub Pages

GitHub Pages — це сервіс від GitHub, який дозволяє користувачам публікувати статичні веб-сайти безпосередньо з репозиторіїв на GitHub без необхідності у використанні додаткового хостингу.

<https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site>

Як команда розробників зазвичай працює з Git?

Ідеально, кожен коміт повинен представляти завершену частину роботи, наприклад, виправлення однієї помилки або додавання однієї функції.

Code Review: Перед злиттям гілки до основної гілки, зміни зазвичай подаються на перевірку. Інші члени команди перевіряють код, вказуючи на можливі проблеми та пропонуючи покращення.

Q&A

Summary

CREATIVE
& TECH
PRJCTR
ONLINE
INSTITUTE

01 / Дізнались що таке Git та GitHub



prjctr.com

What's next?

CREATIVE
& TECH
PRJCTR
ONLINE
INSTITUTE

01 / Продвинута робота з функціями

02 / Деструктуризація

prjctr.com

Додаткові матеріали

01 / [Коротка документація стосовно Git та GitHub](#)

02 / [Тренажер для роботи з гілками](#)

03 / [FAQ та гайди по вирішенню проблем](#)

04 / [Швидкі підказки](#)

Q&A