



# Python for Data Science

Лекція 8. Git - робота з мережевими сховищами і гілками

# Зміст уроку

- Робота з мережевими сховищами
- GitHub – веб-сервіс для хостингу ІТ-проектів і колективної розробки
- Розгалуження і злиття в Git
- Вирішення конфліктів
- Засоби роботи з VCS в PyCharm

# Мережеві репозиторії Git

- Мережеві репозиторії створюються для забезпечення колективної роботи над ресурсами проекту.
- Мережевий репозиторій може бути створений на корпоративних серверах розробника або з використанням веб-сервісів для хостингу ІТ-проектів.
- Git дозволяє використовувати різноманітні протоколи для мережевого доступу: SMB, FTP, HTTP/HTTPS, SSH і ряд інших. Моніторинг і управління доступом до ресурсів забезпечуються засобами сервісу.

# Найпопулярніші веб сервіси для хостингу ІТ-проектів



[GitHub](https://github.com)



[GitLab](https://gitlab.com)



[Bitbucket](https://bitbucket.org)

# Команди Git для роботи з мережевими репозиторіями

- `git remote add <name> <url>` — підключення до віддаленого репозиторію з назвою `<name>` за посиланням `<url>`.
- `git push <remote> <local_branch>` — відправка всіх змін в гілку `<local_branch>` за посиланням `<url>`.
- `git fetch <remote>` — завантаження об'єктів і посилань з віддаленого репозиторію.
- `git pull <remote>` — включає правки з віддаленого репозиторію в поточну гілку.
- `git clone <url>` — клонування віддаленого репозиторію.

# GitHub

**GitHub** (<https://github.com/>) – найбільший і найпопулярніший веб-сервіс для хостингу ІТ-проектів і колективної розробки. Веб-сервіс працює на основі системи контролю версій Git. Пропонує як безкоштовні, так і комерційні пакети послуг.

- Має зручний веб-інтерфейс.
- Вимагає попередньої реєстрації для роботи.
- Підтримується всіма найпопулярнішими IDE.

# Доступ до GitHub-репозиторію через SSH (1)

- На своєму комп'ютері, в консолі, генеруємо RSA-ключ командою:

```
ssh-keygen.exe -o -t rsa -C your_email@mail.com
```

В режимі діалогу вказуємо пароль для ключа. По замовчуванню, ключ зберігається в каталозі `C:\Users\<username>\.ssh` в ОС Windows, або в каталозі `/home/<username>/` в ОС Linux.

# Доступ до GitHub-репозиторію через SSH (2)

- Переходимо в папку з ключами і відкриваємо у текстовому редакторі файл ключа з розширенням `file_rsa.pub`.

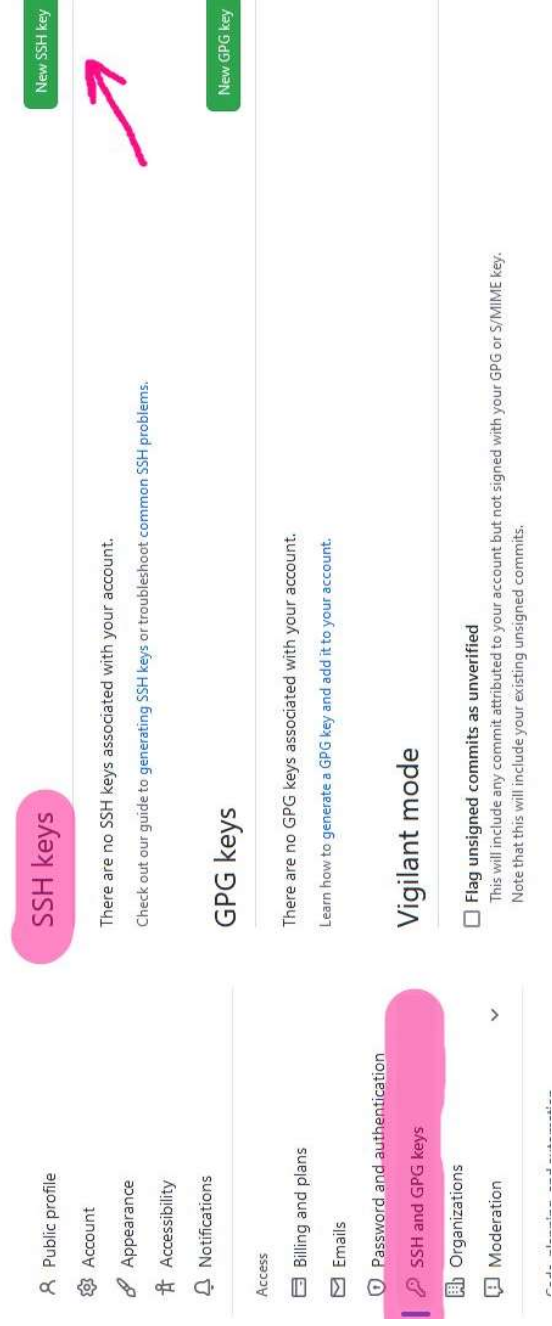
```
ssh-rsa
BBBAb3NzaC1yc2EAAAADAQABBBgQDDR07GYOhtCcMJRU+8KBq7
MghNNyExumdZRM5lmduwytugcj+OkIH/pWtJS/B7f1kOGfxUJSYt
3BoJmkR/dyffXFNI8GSb72y6L/qIZHgiHl9YmQMmrnkYd8UX7sPoB
hpA/eF7kyI8OEHwMy1UUAa+ePDUaojPFInLgqiCXf8l4ip4ZAQfn
zXuA9DUmsZX+Hta/TOTULxDwvJMcV3pdkHsGtAvmThsiwnkGXzzD
JFzCKAoopn9bH8SzGvikjtDU78obtaUlJlBeg8eCLYoJpQJFCfj4
A+Xj5McY2YwDSRmBtskrG60WjqI3Wcl9vd9OFYpIymuwsG5hV4ZO
zPpLclIdSxDp5oirjsWOYJdU31khUoQquyooY+ccl8CvVXECOjcn
+yyFwXzhUgWayLnFKZ2TsjK3PoLa/ruv4w0k33Hjdrk7dGVM7fA1
efwVxpLCCSUV1BCTJKyZvpidf8iujTyH+vE58aJKN4wjE2eXgM5m
eS+r7Picy0IcAV9bDcYuDlq3k= your_email@mail.com
```

Виділяємо і копіюємо зміст даного файлу.



# Доступ до GitHub-репозиторію через SSH (3)

- В браузері авторизуємось на GitHub. В налаштуваннях профіля переходимо в розділ `Settings > SSH keys` і натискаємо на кнопку `New SSH key`.



## Доступ до GitHub-репозиторію через SSH (4)

- Заповнюємо форму. Вказуємо ім'я ключа, а в поле Key вставляємо з буферу обміну зміст файлу ключа. Натискаємо 'Add SHN key'.

SSH keys / Add new		
Title	Key type	Key
	Authentication Key	<pre>ssh-rsa BBBBAB3nzaC1yc2EAAAAADAQABABBgQDDOR07GYOHCmJRU+8K8q7MghNNyEumdzBm5imduwytug+OkiH/pWt JS/B7kKOGSjUJSTy3BoImkR/dy-fXFNI8GSb72y6L/qIZHGhI9ymQMmkYd8UXT7sPoBhpA /e7fkyI8OEHWmXyIUJaa++ePDUa-ojPFIInLqoiCX8f84ip4ZAQfnzXua9DUmsZX+Hta /TOTUJkDwJMcV3pdKhsGtAvmtThsiwnkGxzdJfzCKAooon9bH8szGvktDUJ78obtaUJ11Beg8eCLYoJpQJFCt4A+Xj5Mc Y2YwD5SRmBtskrg60Wjqi3Wd9vd9OFpIymuwsG5hV4ZOzPpLc1IdSxDpSOirjsWOYJdu31khuoQayyooY+cd8CvVXEC Ojcn+yYfwXznUgWYayLqfKZ21tsjK3PoLa /rux4w0k3GHjdk7dGVMW7fA1erwXpLCCSUUV1BCTJkyZvpidr8IuJTyH+veF58aJKN4wJEzeXgM5meS+r7P1cy0IcAV9bDcYU D1g3k= your_email@mail.com</pre>

# Доступ до GitHub-репозиторію через SSH (5)


- В результаті, ми додали до нашого облікового запису буде доданий новий SSH key.

## SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

### Authentication Keys

	<div>rsa.key</div> <div>SHA256: eVFCxXqT [redacted] jUr-Nk</div> <div>Added on 15 Sep 2022</div> <div>Never used — Read/write</div>	<div>SSH</div> <div>Delete</div>
---	---	----------------------------------

Check out our [guide to generating SSH keys](#) or [troubleshoot common SSH problems](#).

# Доступ до GitHub-репозиторію через SSH (6)

Переходимо до нашого репозиторію. Натискаємо на кнопку Code. Обираємо тип з'єднання SSH. Копіюємо в буфер обміну рядок для під'єднання у форматі: `git@github.com:<username>/<rep_name>.git`



## Доступ до GitHub-репозиторію через SSH (7)

- Далі, ми отримаємо можливість клонувати репозиторій з GitHub на свій комп'ютер командою:

```
git clone git@github.com:<username>/<rep_name>.git
```

або підключитись до віддаленого репозиторія:

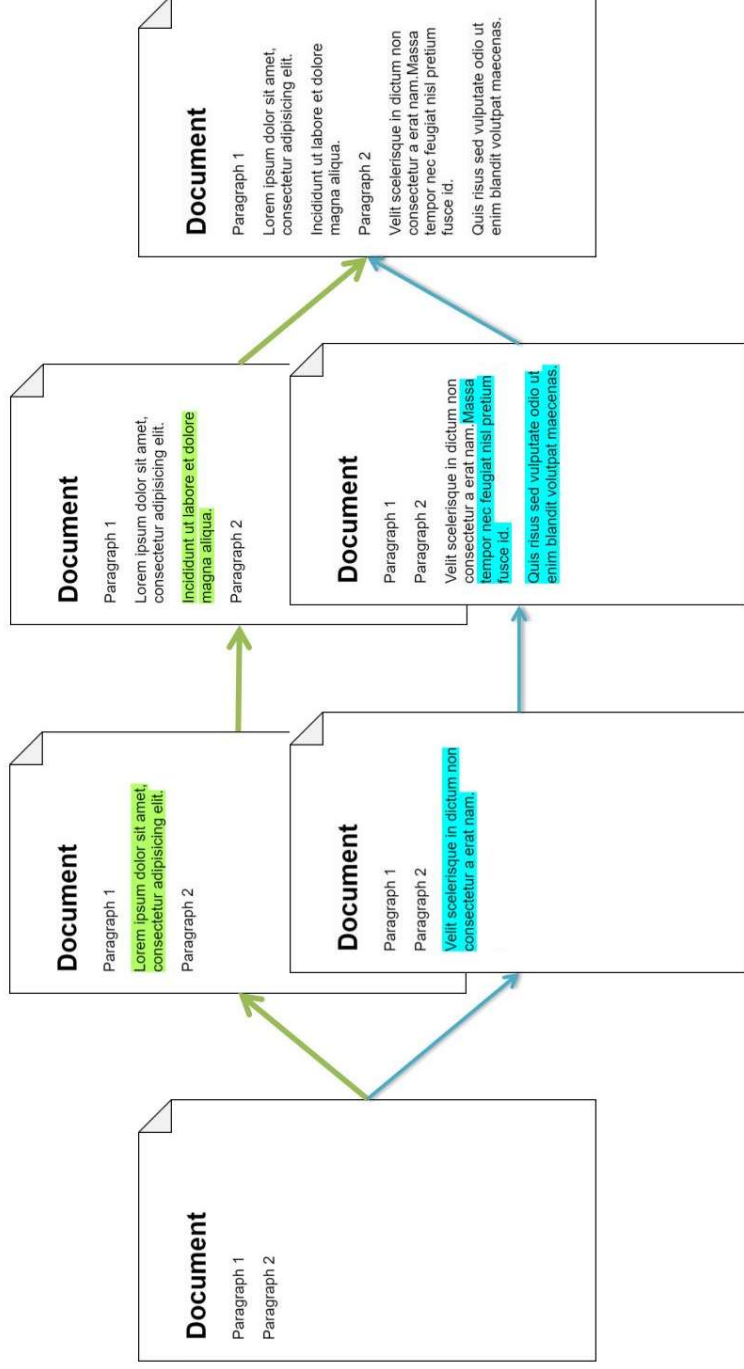
```
git remote add <rep_name>_  
git@github.com:<username>/<rep_name>.git
```

# Гілка

- **Гілка** — це указчик на коміт. За замовчуванням, основною гілкою Git є master. Після створення кожного нового коміта гілка завжди вказує на останній коміт.

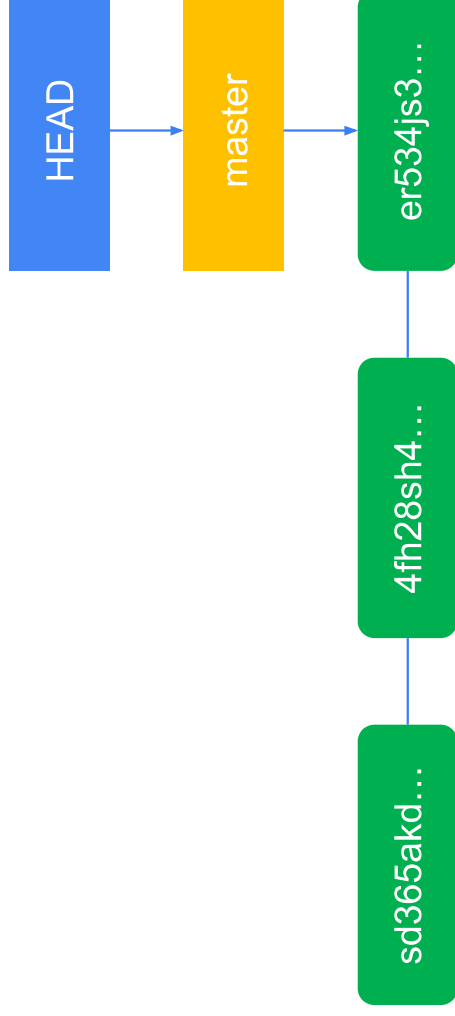
```
Vlad@NAVI MINGW64 /d/---222--- (master)
$ git branch
* master
  second
```

# Злиття різних гілок документу



# Розгалуження і злиття (1)

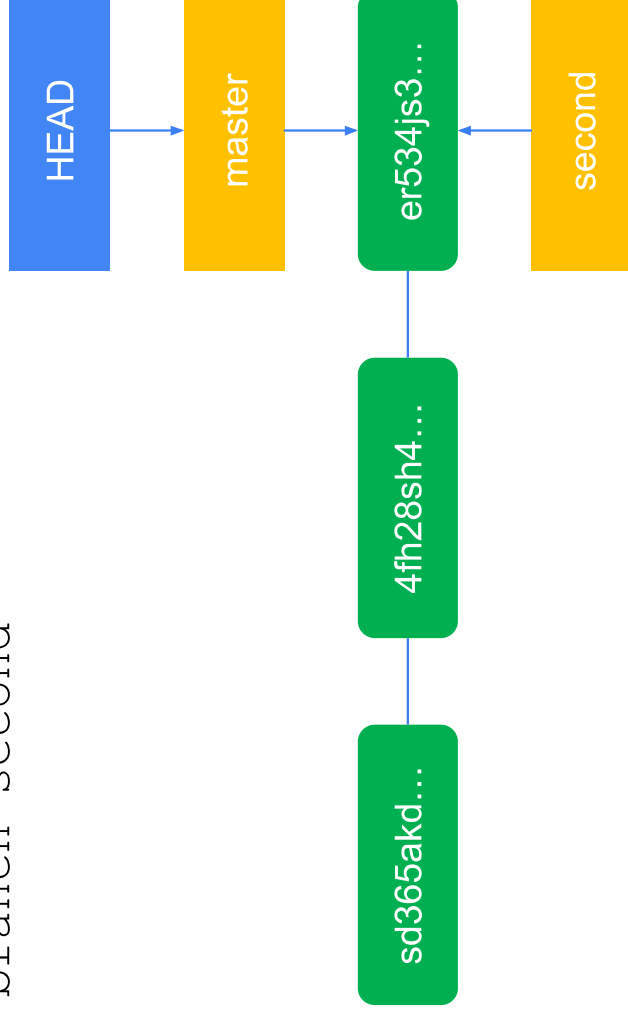
`git commit`





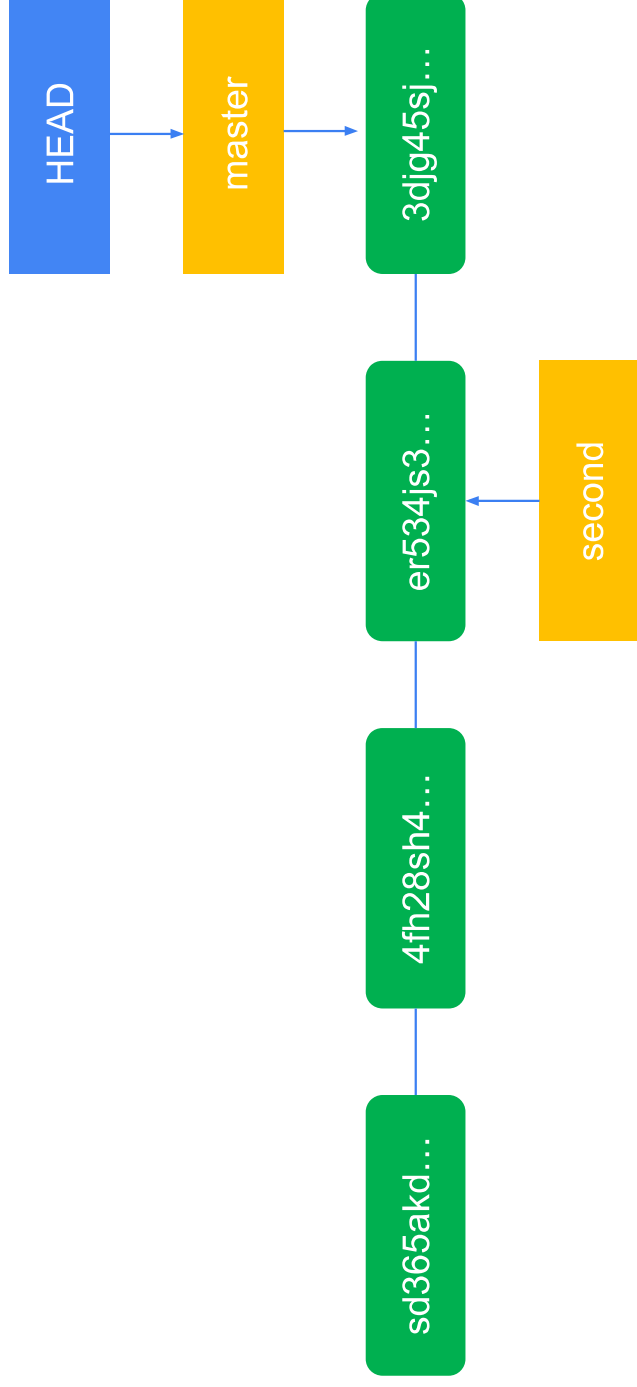
## Розгалуження і злиття (2). Створення нової гілки

git branch second



# Розгалуження і злиття (3)

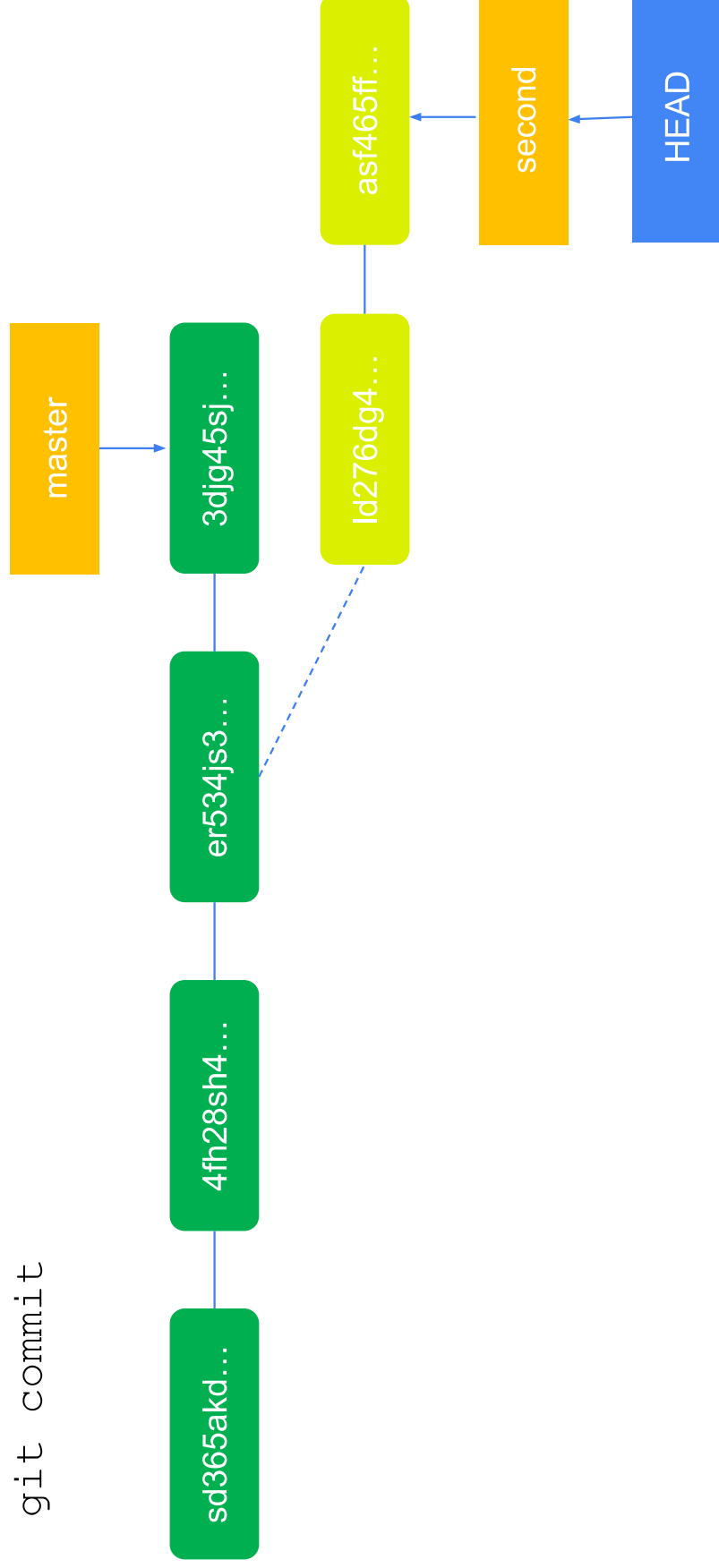
`git commit`



# Розгалуження і злиття (4)

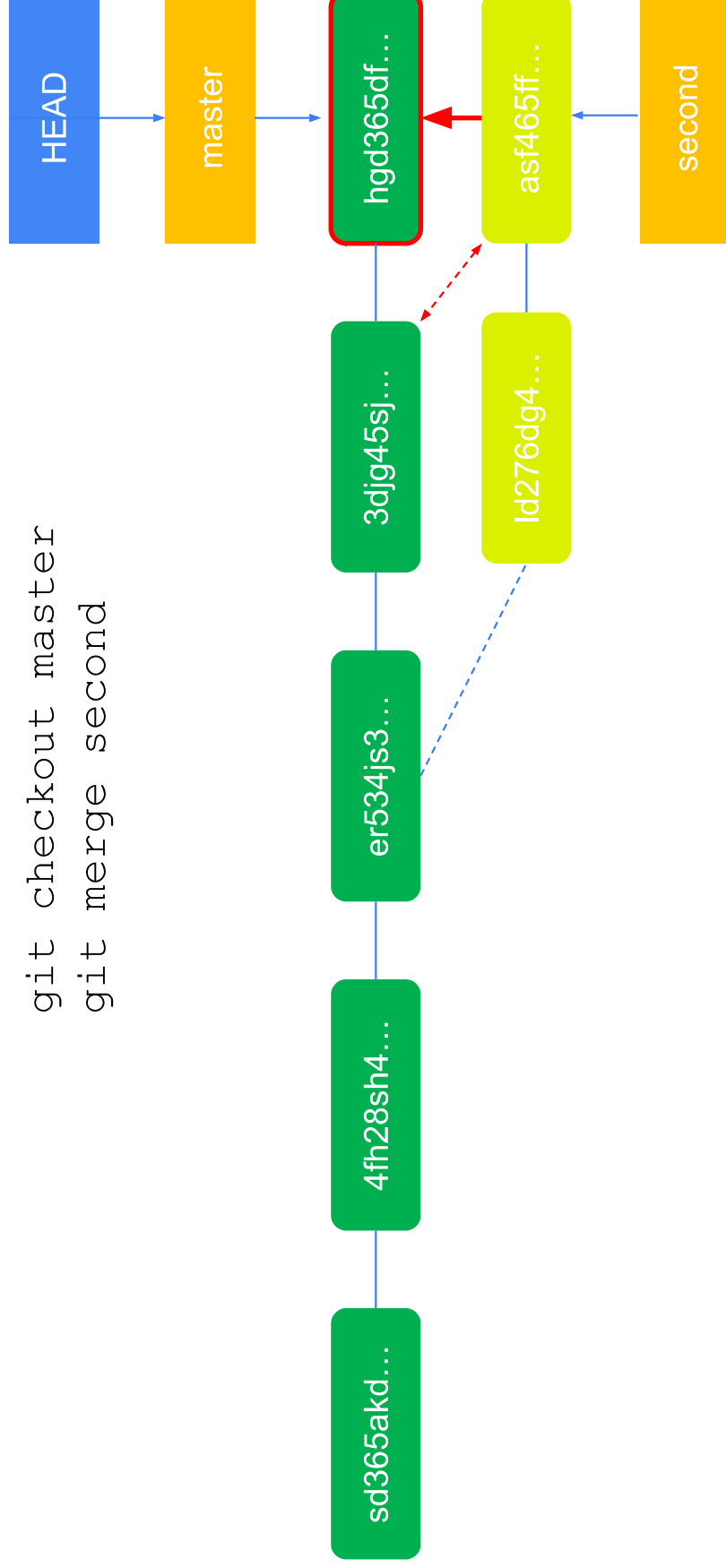
git checkout second

git commit



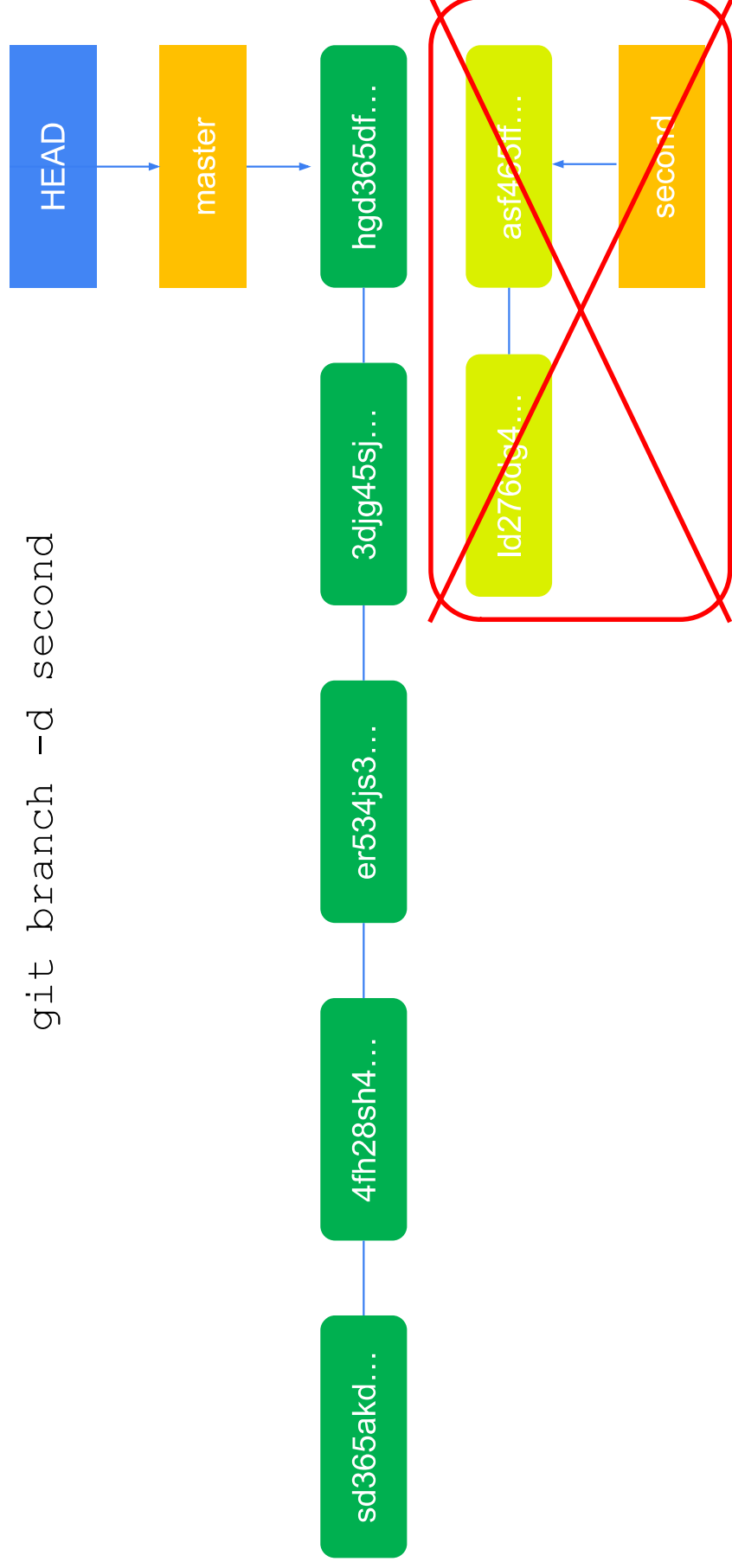
# Розгалуження і злиття (5). Злиття об'єднанням

git checkout master  
git merge second



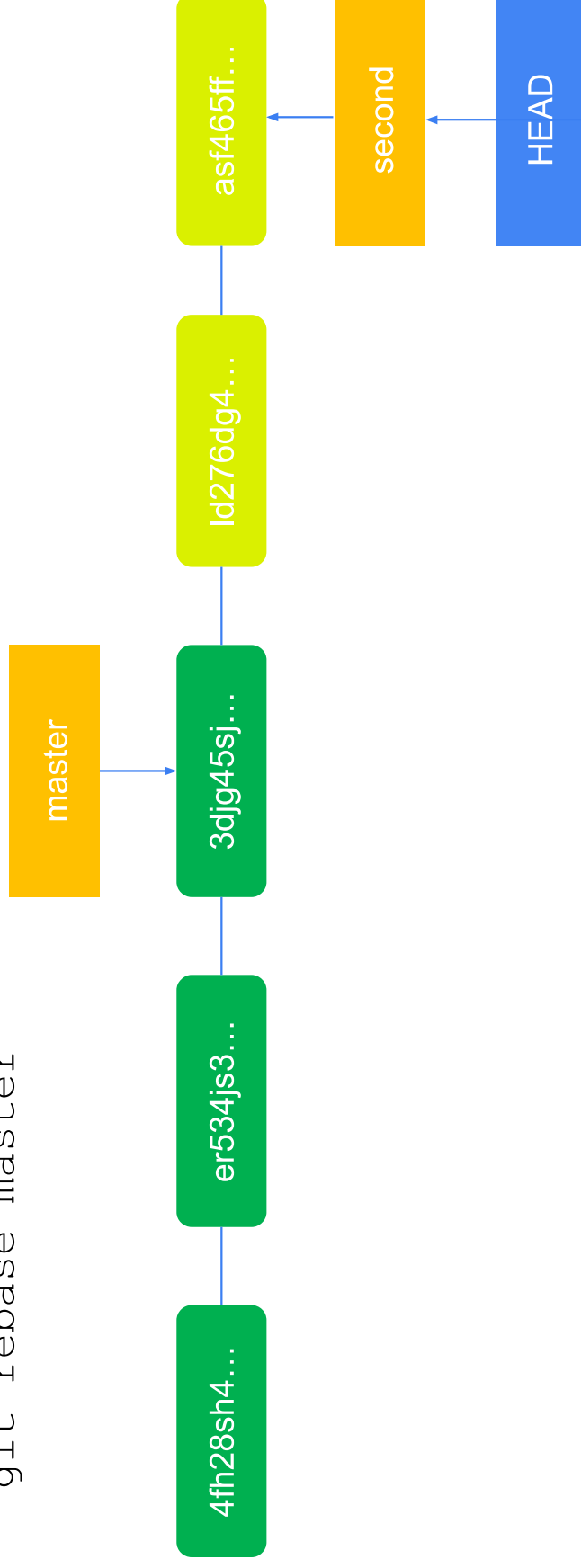
# Розгалуження і злиття (6). Видалення гілки

`git branch -d second`



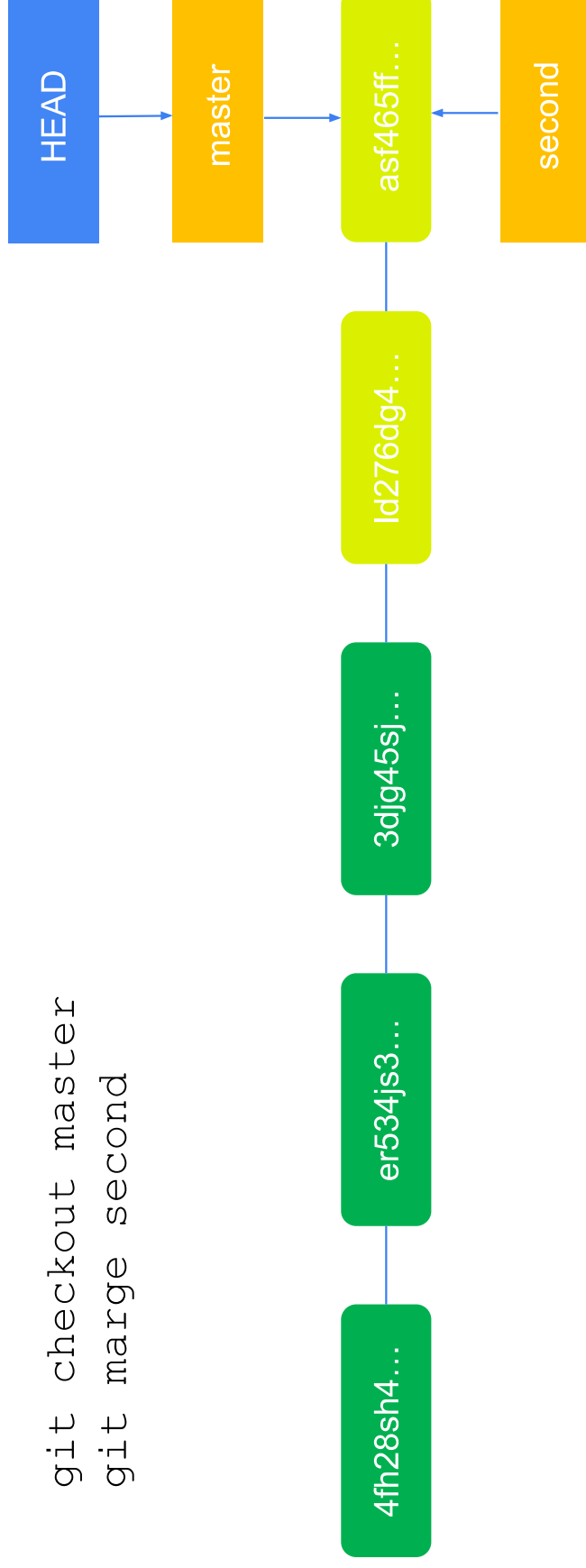
# Злиття заміщенням (1)

git commit  
git checkout second  
git rebase master



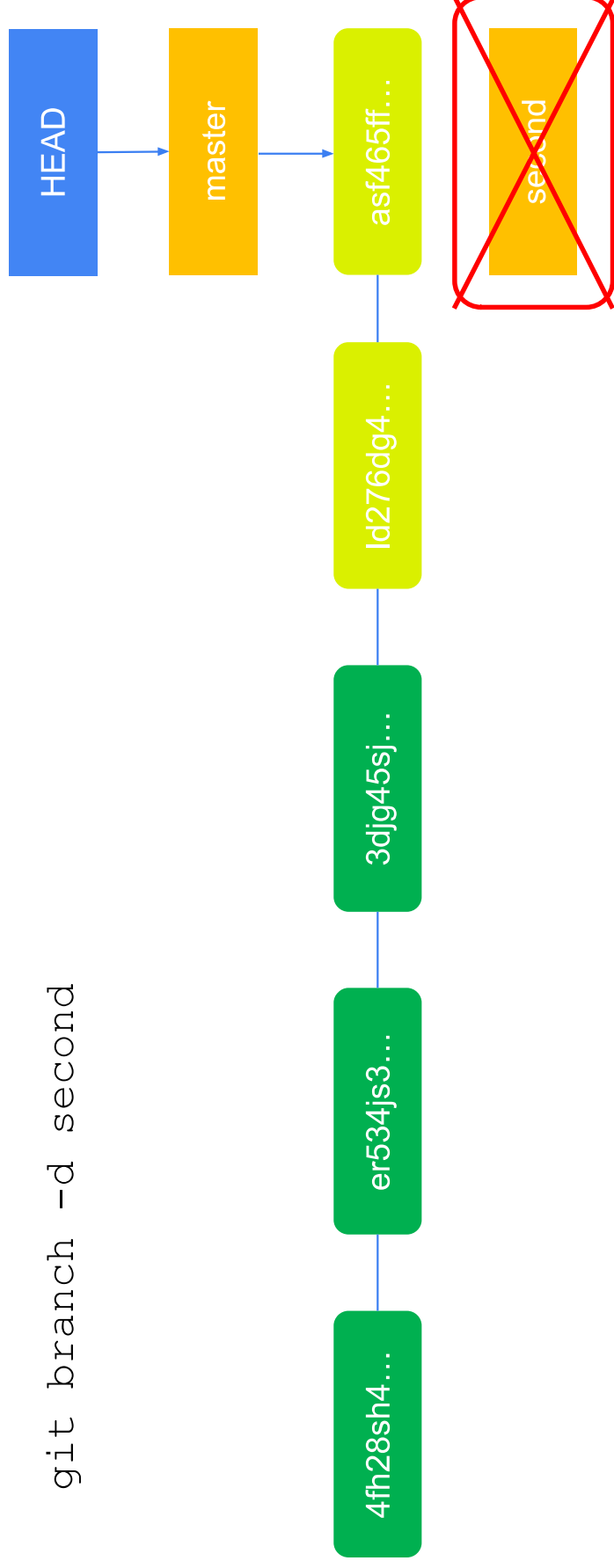
## Злиття заміщенням (2)

```
git checkout master  
git merge second
```



## Злиття заміщенням (3)

`git branch -d second`





# Конфлікти злиття

- Конфлікт злиття відбувається, коли одну і туж частину коду змінювали в обох гілках, які зливаються в єдину. Якщо Git не може автоматично вирішити, які зміни слід залишити в результаті злиття, це необхідно зробити вручну.

# Основні команди Git для роботи з гілками та злиття

- `git branch` — перегляд всіх існуючих гілок.
- `git branch <branch_name>` — створення нової гілки репозиторію.
- `git branch -d <branch_name>` — видалення вказаної гілки.
- `git checkout <branch_name>` — переключення на вказану гілку.
- `git checkout -b <branch_name>` — створення нової гілки і переключення на неї.
- `git merge <branch_name>` — злиття гілок шляхом об'єднання. Вказана гілка «зливається» з гілкою, на яку вказує HEAD.
- `git rebase <branch_name>` — злиття шляхом заміщення (перебазування). Коміти з гілки, на яку вказує HEAD, переміщуються вперед комітів, вказаних у параметрі `<branch_name>`.

# Загальні інструменти для виправлення помилок

- `git status` — для виявлення місця конфлікту, та інформації про помилку
- `git log --merge` — створення журналу зі списком конфліктів коммітів для яких виконується злиття
- `git diff` — пошук відмінностей

# Конфлікти на початку злиття

error: Entry '<fileName>' not uptodate. Cannot merge. (Changes in working directory)

## Інструменти для виправлення:

- **git checkout** — є можливість скасовувати зміни в файлах або гілках
- **git reset** — скасування змін в проіндексованих файлах

# Конфлікти під час процесу злиття

error: Entry '<fileName>' would be overwritten by merge. Cannot merge. Changes in staging area

## Інструменти для виправлення:

- `git merge --abort` — процес злиття зупинений і гілка буде перейти в стадію до початку злиття
- `git reset`

# Робота з VCS в PyCharm

- IDE PyCharm підтримує всі популярні системи VCS та веб-сервіси ІТ-проектів.

No VCS enabled

VCS Operations	
1	Create Mercurial Repository
2	Import into Subversion...
3	Create Git Repository...
4	Enable Version Control Integration...
Local History	
5	Show History
6	Put Label...

VCS enabled

VCS Operations	
Git	
✓ 1	Commit... ⌘K
2	Commit File
↶ 3	Rollback... ⌘Z
🕒 4	Show History
5	Annotate
↕ 6	Show Diff
🔗 7	Branches...
➦ 8	Push... ⌘K
9	Stash Changes...
0	Unstash Changes...
Local History	
	Show History
	Put Label...

# Основні комбінації клавіш для роботи з VCS в PyCharm

VCS Operations Popup...	Alt+`
Commit...	Ctrl+K
Update Project	Ctrl+T
Rollback	Ctrl+Alt+Z
Push...	Ctrl+Shift+K
Next Change	Ctrl+Alt+Shift+Down
Previous Change	Ctrl+Alt+Shift+Up
Show Version Control window	Alt+9
Show Commit window	Alt+0

# Стратегії розгалужень

## Популярні види:

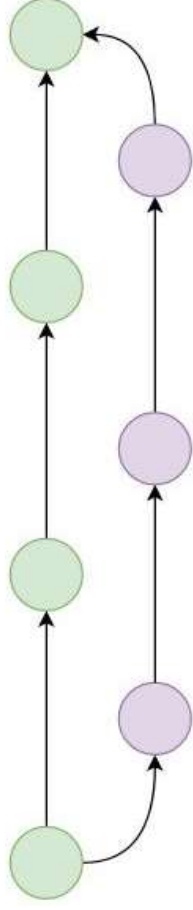
1. Функціональні гілки
2. Git-flow
3. Forking Workflow
4. Магістральна розробка



# Функціональні гілки

## Структура:

- Існує головна гілка **main (master)**.
- Для розробки нового функціоналу створюється незалежна гілка, яку теж додають в віддалений репозиторій.
- Для підтвердження злиття в **main (master)** або для обговорення використовують **pull request**.



```
$ git checkout master  
$ git pull origin master  
$ git tag v1.2.3  
$ git push origin master --tags
```

# Git-flow

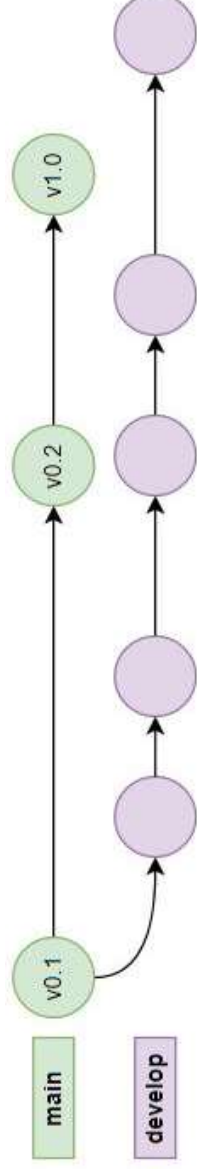
`git flow init` — ініціалізація команд `git-flow` для проекту.

Структура:

- **main (master)** гілка
- **develop** гілка
- **feature** гілка (для кожної нової функції створюється нова)
- **release** гілка
- **hotfix** гілка

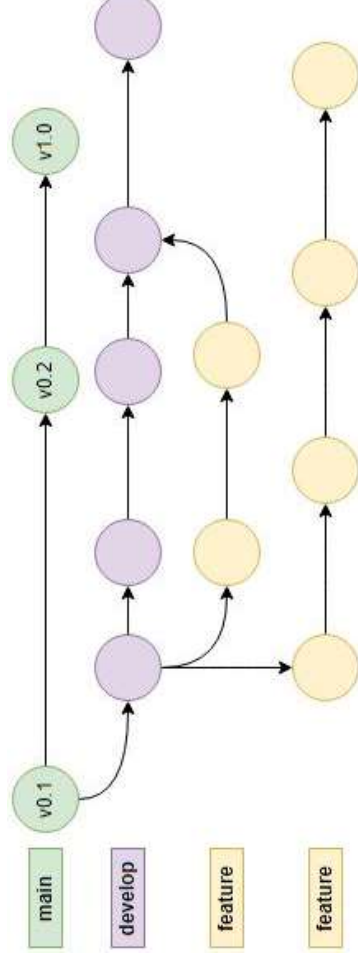
# Головна гілка та гілка `develop` (розробки)

- Головна гілка **main** містить історію релізів (скорочену історію проекту). Всі коміти починаються з номера версії.
- Гілка **develop** об'єднує всі функції. Зберігається повна історія проекту.



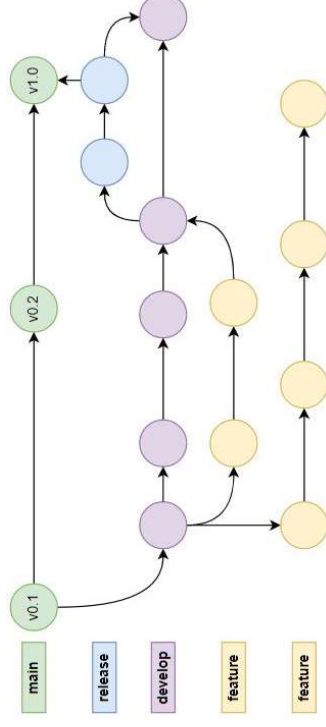
# Гілка feature

- Основу становить гілка **develop**.
- В кінці роботи зливається гілкою **develop**.



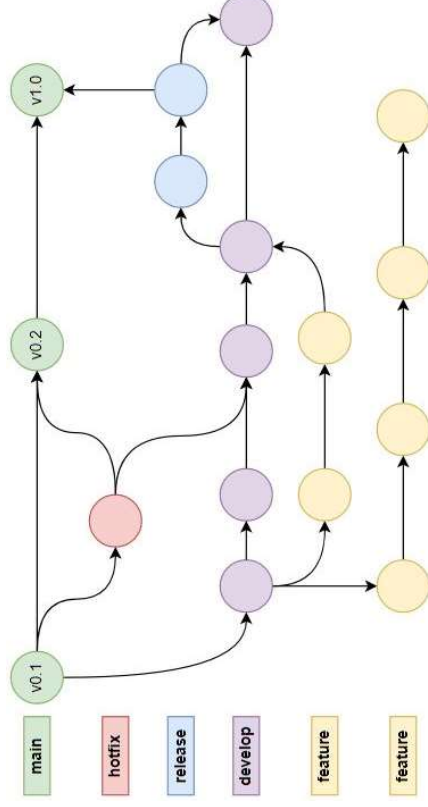
# Гілка випуску (release)

- Основу становить гілка **develop**.
- Створюється, коли наближається дата релізу або достатньо функцій для випуску.
- Створення гілки **release** запускає наступний цикл релізу.
- Нові функції не додаються.
  - можливе виправлення “багів”.
  - створення документації тощо.
- Об'єднується з **main (master)** та з гілкою **develop**.
- Після об'єднання видаляється.



# Гілка виправлень (hotfix)

- Основа **main (master)**.
- Для внесення виправлень в робочий реліз.
- В кінці роботи зливається з гілкою **main (master)** і **develop** та присвоюється оновлений номер версії релізу.

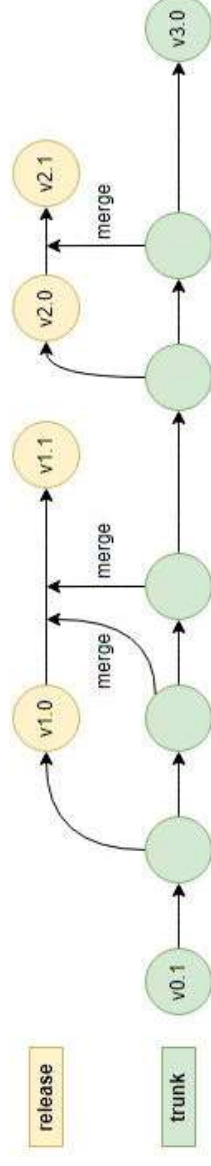


# Forking Workflow

- Розробник створює локальну копію **main (master)** репозиторію через **fork**, далі **git clone**, разом з під'єднанням проекту до **main (master) url**.
- Додає функціонал створюючи нову **feature** гілку, яка теж додається у віддалений публічний репозиторій та створює **pull request** для її інтеграції в **main (master)**.
- Для перевірки, відповідальна особа додає **feature** учасника, спочатку у свій локальний **main (master)** та вже після в **центральному**.

# Trunk based development

- Є main гілка **trunk** (стовбур), в яку додаються всі коміти.
- **Релізи** випускаються через **trunk** або гілку релізу.
- Для перевірки коду користуються **CI/CD pipeline** (налаштована автоматична програмна перевірка етапів релізу).
- Постійне відстеження якості роботи програми.





# Задачі

- Зареєструйтесь на GitHub. Створити мережевий репозиторій для проекту на Python.
- Згенеруйте RSA-ключ та екпортуйте його на у свій профіль на GitHub. Отримайте URL для підключення до вашого репозиторію через протокол SSH.
- Налаштувати підключення до вашого репозиторію через інтерфейс PyCharm і виконайте його клонування на свій комп'ютер.
- В середовищі PyCharm, додайте декілька комітів і відправте їх на GitHub. Перевірте, що вся історія завантажена.

## Задачі

- В гілці 'master' і виконайте декілька комітів. Створіть нову гілку 'second' і зробіть в ній декілька комітів. Виконайте злиття гілки 'second' у гілку 'master'. Видаліть гілку 'second'.
- Повторіть попередні кроки, тільки на цей раз виконайте об'єднання гілок 'master' і 'second' шляхом заміщення.