Git Practice

Обов'язкові налаштування

- Необхідно налаштувати інформацію про автора. Інакше, без цих даних Git не дозволить нам робити комміти. Використовуйте своє ім'я та адресу ел. пошти.

- $ git config --global user.name “Baal Child"
- $ git config --global user.email sarevok@gmail.com

- Важливо! Вказані дані повинні збігатися з ім'ям та електронною поштою облікового запису на GitHub

Починаємо відслідковувати історію

- Створіть порожню директорію practice-git і перейдіть до неї
- Ініціалізуйте репозиторій за допомогою команди
  git init
- Переконайтеся, що утворилася директорія .git

Додаємо файл

- 1. Створимо текстовий файл:
  echo "My first line in project" > file.txt
- 2. Виконаємо git status
  Чому файл позначений як “untracked”?
- 3. Додаємо файл до індексу:
  git add file.txt
- 4. Знову виконаємо git status
- 5 . Дивимося проіндексовані зміни:
  git diff --cached

Збережемо зміни в історії

- Створюємо коміт:
  git commit -m "My first commit"
- Перевірте, що коміт є в історії:
  git log
- Перегляньте інформацію про останній коміт:
  git show

Додамо ще один коміт

- Додамо рядок у існуючий файл:
  echo "London is the capital of GB" >> file.txt
- Подивимося непроіндексовані зміни:
  git diff
  - Порівняйте вивід з результатом git diff --cached
- Створимо коміт: git commit -am "Capital added“
  - Чи вдалося нам зробити коміт без додавання до Index?
    Або ми автоматично додали зміни перед комітом?
- Перевіримо в історії, чи ми зробили коміт:
  git log

Виправимо попередній коміт

- В останньому коментарі ми зафіксували додавання рядка *"London is the capital of GB"* у файл file.txt , але забули додати до файлу, що *“Kyiv is the capital of Ukraine"*.

  Виправте останній коміт (не додаючи новий) та у повідомленні коміту вкажіть, що було додано не одну столицю, а декілька.

  (відповідь є на наступному слайді)

Виправимо попередній коміт (рішення)

- Додамо новий рядок:
  echo “Kyiv is the capital of Ukraine” >> file.txt
- Додамо зміни в stage:
  git add file.txt
- Додамо оновлений stage та змінимо commit message:
  git commit --amend
- Переконаємося, що комітів, як і раніше, два і нам дійсно вдалося змінити останній коміт, не створюючи новий:
  git log
- Команда git show , покаже нам зміни у останньому коміті

Додамо Францію

- Додайте до кінця файлу file.txt рядок "Paris is the capital of France" та зафіксуйте ці зміни, зробивши коміт. У повідомленні коміту вкажіть "Another capital was added".
- Використовуйте команди git log та git show, щоб перевірити зроблений коміт.

Скасуємо другий коміт

- Знайдіть hash другого комміту із git log
- Використовуйте команду git revert 1f829c0 щоб скасувати коміт.
  - ℹ Використовуйте перші символи свого хешу (рекомендується вказувати перші 7 символів)
  - Також можна скористатися HEAD~1
- Якщо все правильно, то має виникнути конфлікт

Вирішуємо конфлікт

- Дивимося, де у нас стався конфлікт командою git status
- Дивимося рядки, що конфліктують, командою cat file.txt

```
My first line in project
<<<<<<< HEAD
London is the capital of GB
Kyiv is the capital of Ukraine
Paris is the capital of France
=======
>>>>>>> parent of 7bf6180... Capital added
```

- *Git позначає спеціальними маркерами рядки, в яких стався конфлікт*

Продовжуємо

- Повідомимо Git, що ми вирішили конфлікт - додамо файл до stage командою git add file.txt
- Продовжимо процес скасування коміту - git revert --continue
- Перевіримо, що створився новий коміт - git log
- Подивимося, що змінилося в останньому коментарі - git show

Працюємо з .gitignore

- Створіть два файли з однаковим розширенням, наприклад:

  echo "this is a garbage file" | tee garbage1.tmp garbage2.tmp

- Перевірте, що Git бачить ці файли командою git status
- Створіть файл .gitignore та вкажіть у ньому, щоб Git ігнорував усі файли з розширенням .tmp, оскільки ми не хочемо їх відстежувати.
- Знову зробіть команду git status - файли з розширенням .tmp повинні зникнути з виводу.
- Зробіть коміт з .gitignore

Завдання на розуміння Index

- Зробіть так, щоб у виводі команди git status файл file.txt був присутній у двох секціях одночасно:
  Changes to be committed та *Changes not staged for commit*.
- Порівняйте виведення команд git diff та git diff –cached

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   file.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   file.txt
```

Продовження роботи

- Результат потрібно закомітити, щоб робоча директорія була чистою

```
$ git status
On branch master
nothing to commit, working tree clean
```

Робота з гілками

- Створення та видалення гілок
- Мердж гілок
- Переміщення (rebase) гілок
- Клонування віддаленого репозиторію
- Pull і Push у віддалений репозиторій

Створимо гілку

- Створіть гілку з ім'ям first - git branch first
- Перегляньте список локальних гілок - git branch
  (зірочкою)буде позначено поточну гілку)

```
$ git branch
    first
  * master
```

- Перейдіть на гілку first - git checkout first

Створимо ще одну гілку

- Створимо і переключимося на гілку однією командою (порівняйте, як булоз гілкою **first**) - git checkout -b second
- Знову перевіримо поточну гілку, але тепер командою git status
- І знову подивимося список локальних гілок - git branch

Змінимо гілку second

- Створіть новий файл, наприклад:

```
echo "Let's change the branch" > branch.txt
```

- Зробіть коміт з повідомленням *“Branch was changed ”*

Зробимо коміт на першій гілці

- Перейдіть на гілку **first**, створіть файл first.txt з будь-яким вмістом і зробіть коміт.
- Намагайтеся зробити самостійно. Відповіді наведено на наступному слайді.

Підказка:

- Треба переключитися, створити файл та зробити коміт:

```
$ git checkout first
Switched to branch 'first'

$ echo "And this branch too" > first.txt

$ git add first.txt

$ git commit -m "First branch changed"
[first 2e54f37] First branch changed
1 file changed, 1 insertion(+)
create mode 100644 first.txt
```

Дивимося результат

- Подивимося граф змін - git log --all --decorate --oneline—graph
- Створіть alias для цієї команди, вона нам ще знадобиться:

git config --global alias.g 'log --all --decorate --oneline --graph‘

```
$ git g

* 2e54f37 (HEAD -> first) First branch changed
| * f279210 (second) Branch changed
|/
* 17d8431 (master) .gitignore added
* f9e503c Revert "Capitals added"
* de140fc France added
* 66bca8c Capitals added
* bf79e90 My first commit
```

Змерджимо гілки

- Перейдемо до гілки **master** та перевіримо наявність файлів:

  ```
  git checkout master
  ls
  ```

- Змерджимо гілку **first** (зверніть увагу – у нас з’явився файл first.txt):

```
$ git merge first
Updating 17d8431..2e54f37
Fast-forward
first.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 first.txt
```

Зробимо **rebase** гілки second

- Перейдемо до гілки **second** та перевіримо наявність файлів:

  git checkout second

- Виконаємо **rebase**:

  git rebase master

  First, rewinding head to replay your work on top of it...
  Applying: Branch changed

Гілка second змінила свою базу

- Що ми отримали:

```
$ git g
* b77f3f6 (HEAD -> second) Branch changed
* 2e54f37 (master, first) First branch changed
* 17d8431 .gitignore added
* f9e503c Revert "Capitals added"
* de140fc France added
* 66bca8c Capitals added
* bf79e90 My first commit
```

- Лінійна історя
- Гілки **master** та **first** співпадають
- Гілка **second** на 1 коміт попереду **master**

Видалимо гілку

- Видаляємо гілку **first**:

```
$ git branch –d first

Deleted branch first (was 2e54f37).
```

- Перевіримо:

```
$ git branch

  master
* second
```