

Використання Github Pages

Github – сервіс віддалених репозиторіїв для розподіленої системи контролю версій (СКВ) Git. Github надає можливість безкоштовного хостингу статичних web-ресурсів, **GitHub Pages**. Слід розрізняти сам Github та Github Pages, як лише одну з його можливостей.

Для роботи буде потрібно ПК із встановленою СКВ Git. Графічний клієнт не потрібний, **робота ведеться у консольній оболонці** (залежно від налаштувань при інсталяції – консоль Git, Git Bash, або просто консоль самої ОС – cmd або PowerShell).

Робота передбачає використання **робочого каталогу** (де, власне, вами ведеться робота над вашим проектом і знаходяться файли з вихідним кодом), **локального репозиторію** СКВ Git (де зберігаються «комміти» – версії вашого вихідного коду) і **віддаленого репозиторію** на сервісі Github. Загальний принцип простий – після внесення змін у файли з вихідним кодом потрібно створити новий комміт (версію) у локальному репозиторії, після чого дані з нього відправляються у віддалений репозиторій. А там вже завдяки сервісу Github Pages здійснюється публікація вашого проекту як повноцінного web-ресурсу.

Можливо два шляхи початку роботи – або спочатку створити локальний репозиторій (команда *git init*) та підключити до нього також створений віддалений репозиторій (*git remote add*), або, навпаки, **спочатку створити віддалений репозиторій**, потім його клонувати собі локально, тим самим отримавши готовий пустий репозиторій на локальному ПК, вже прив'язаний до віддаленого – так саме пустого, це ж клон. Обидва способу рівноцінні, але при використанні клонування буде трішки менше команд, тож його і розглянемо докладніше.

Оскільки неможна працювати з віддаленим репозиторієм, якщо його поки не існує – скористаймося сервісом Github. Якщо немає облікового запису на Github – **реєструйтесь**. Це не буде зайвим і за межами навчання в університеті. Обов'язково запам'ятайте **параметри входу**, вони знадобляться в майбутньому.

Після успішної реєстрації свого акаунту, можна через web-інтерфейс створити новий порожній репозиторій:



При створенні строго необхідно вказати тільки **назву** (у прикладі це *kiuki22*):


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

 knureigs

Repository name *

/ kiuki22

✓ kiuki22 is available.

Great repository names are short and memorable. Need inspiration? How about [miniature-computing-machine](#) ?

Description (optional)

Example for first lecture.

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

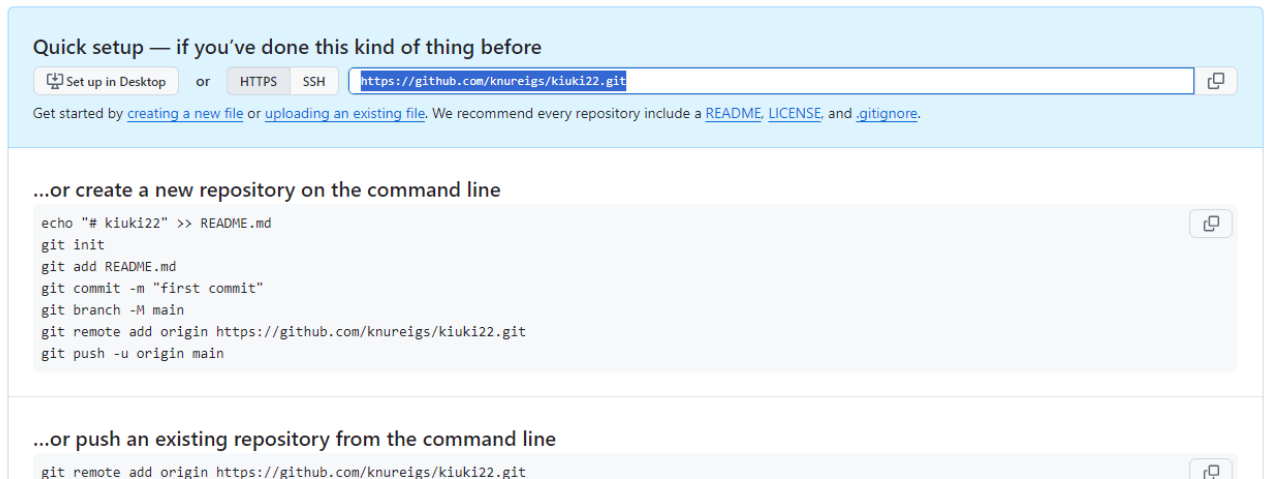
Initialize this repository with:

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Використання файлу **.gitignore**, що містить перелік файлів, що ігноруються СКВ, для даної лабораторної не є необхідним. Чудовим при створенні репозиторію є вказівка ліцензії та опису проєкту – але це також необов'язково.

Після створення репозиторія проєкта, Github сам підказує, що оскільки репо пустий, то доречними будуть такі команди для використання на локальному ПК:



Тут, власне, є все необхідне для створення локального репозиторію «власноруч» – команда `git init` для ініціалізації, `git remote add` для вказівки віддаленого репо – але ми підемо простішим шляхом, і з підказок Github скористаємось тільки **адресою репозиторія**.

На локальному ПК, де буде вестися ваша робота, перейдемо до каталогу, де плануєте розміщувати папку вашого проєкту. І виконуємо **клонування**:

```
PS D:\Programming\workspaceVSC\ITech1> git clone https://github.com/knureigs/kiuki22.git
Cloning into 'kiuki22'...
warning: You appear to have cloned an empty repository.
```

Git попередив, що клонуємо пустий репозиторій. Так, це так і планувалося. Наповнимо його! Чим? Це вже не питання роботи з СКВ, але у контексті лабораторних робіт за курсом «Internet-технології» у вас, напевно, буде щонайменше один файл на мові розмітці HTML. У прикладі так і зроблено – у робочому каталозі доданий файл **index.html**, у якому написано щось по темі лабораторної роботи.

Не буде зайвим (взагалі ніколи не буде зайвим, буквально) – завжди перевіряти стан свого локального репозиторію. Для цього використовується команда `git status`:

```
PS D:\Programming\workspaceVSC\ITech1\kiuki22> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.html

nothing added to commit but untracked files present (use "git add" to track)
```

СКВ нам підказує, що поки що комітів немає – жодної версії не збережено. Але присутній файл `index.html`, який системі невідомий, і, напевно, який слід **проіндексувати** – щоб Git розумів, з яких файлів складатися буде версія вихідного коду вашого проєкту. Це дуже важливий аспект, оскільки у реальних проєктах файлів з вихідним кодом на порядки більше, ніж у прикладі для лабораторної роботи. Але у нас поки що все просто, і користуємося підказкою самої СКВ – використати команду **git add**. Для якої можна вказати

конкретний файл, який індексуємо, або вказати «.» – просто крапку, щоб проіндексувати взагалі все, що там можна індексувати.

Вказуємо конкретний файл для індексації (саме він у нас потрапить до майбутнього «комміту», тобто «зліпку», або збереженої версії вихідного коду) і перевіряємо знову стан – як писав раніше, це ніколи зайвим не буде. Не знаєте, що робити – перевіряйте стан репозиторія. Це дозволить уникнути купи помилок.

```
PS D:\Programming\workspaceVSC\ITech1\kiuki22> git add index.html
PS D:\Programming\workspaceVSC\ITech1\kiuki22> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   index.html
```

Що змінилось? Файл був раніше вказаний червоним кольором, тепер зелений. СКВ натікає, що можна вже зробити комміт – Git, після індексації, знає, з чого сема, з яких файлів, цей комміт буде складатися. Власне, індексація буде необхідна після кожного комміту – версій у проєкта, як правило, значно більше однієї.

На цьому етапі у нас ініціалізований (точніше, клонований) локальний репозиторій (сховище даних про версію вашого проєкту) та додані всі файли з робочої папки (всі – тобто один) до контрольованих. Тобто, СКВ відстежує індексовані файли, але поки що нічого фактично не зберігає – версій поки немає. Для заповнення репозиторію необхідно виконати комміт (фіксацію змін у робочій папці):

```
git commit -m "commit description"
```

Дуже раджу саме такий формат використання команди `git commit` – де ключ «-m» дозволяє безпосередньо у рядку команди вказати назву комміту, без використання окремих редакторів (тим паче, що за замовчуванням для цього використовується не самий легкий у використанні `vim`). Як називати комміт? Суворих вимог немає, але є рекомендації щодо свідомого неймінгу і навіть стандарти іменування. У цьому ж прикладі назва комміту говорить, що була додана певна функціональність, яка реалізована у файлі `index.html`:

```
PS D:\Programming\workspaceVSC\ITech1\kiuki22> git commit -m "feat: add index.html"
[master (root-commit) 5b6c8b4] feat: add index.html
1 file changed, 11 insertions(+)
create mode 100644 index.html
```

До речі, якщо Git використовувався на даному ПК вперше, то на цьому етапі комміту не відбудеться, а буде виведено повідомлення про помилку – із запитом даних того, хто саме робить цей комміт (тобто хто відповідає за зміни у вихідному коді, настільки важливі, що їх вирішили зберегти у окремій версії). І запропоновано вирішення проблеми – шляхом вказівки двох команд, що заповнюють конфігураційний файл СКВ.

```
git config --global user.name "ваше_ім'я"
```

```
git config --global user.email "ваша_пошта"
```

Щоб встановити налаштування тільки для даного поточного проєкту, потрібно прибрати ключ «--global». Інакше вказані вами налаштування будуть використовуватися за замовчуванням для всіх проєктів цього ПК – тобто, при фіксації змін (виконанні команди `commit`) на всіх проєктах, а не тільки ваших, за замовчуванням фігуруватимуть ваші дані.

На цьому етапі – після успішного комміту – поточний стан проєкту збережено у локальному репозиторії. Перша версія історії розвитку вашого вихідного коду є.

Тепер потрібно передати свої локальні комміти у віддалений репозиторій, на Github. Завдяки тому, що ми свій локальний репо взяли буквально шляхом клонування з Github, у нас вже є прив'язка – і не потрібна команда `git remote add`.

Тож просто виконуємо відправку – команда **git push**:

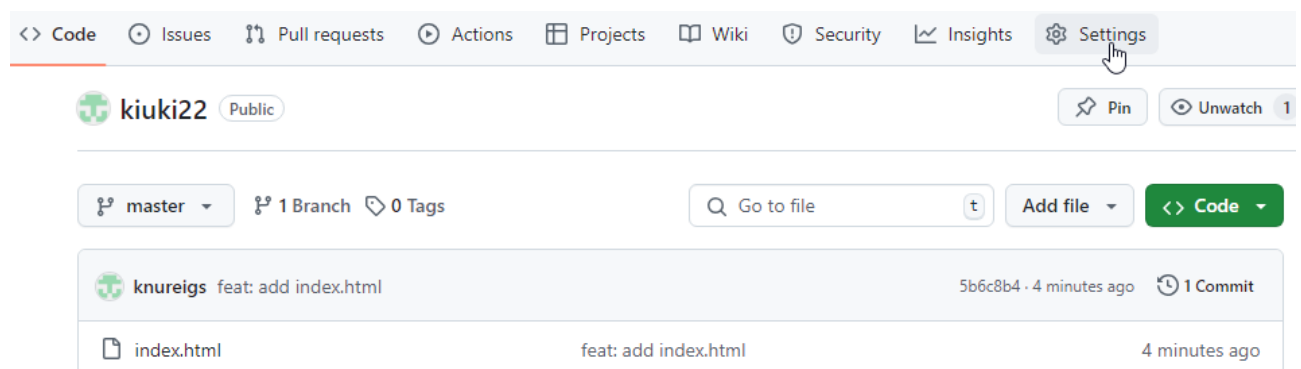
```
PS D:\Programming\workspaceVSC\ITech1\kiuki22> git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 387 bytes | 129.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/knureigs/kiuki22.git
* [new branch]      master -> master
```

А якщо б йшли не через клонування, то команда відправки була б трішки складнішою (але тільки на перший раз, далі так саме просто): **git push -u origin master**, тобто через вказівку гілки-джерела і назви (псевдоніма) для віддаленого репозиторію.

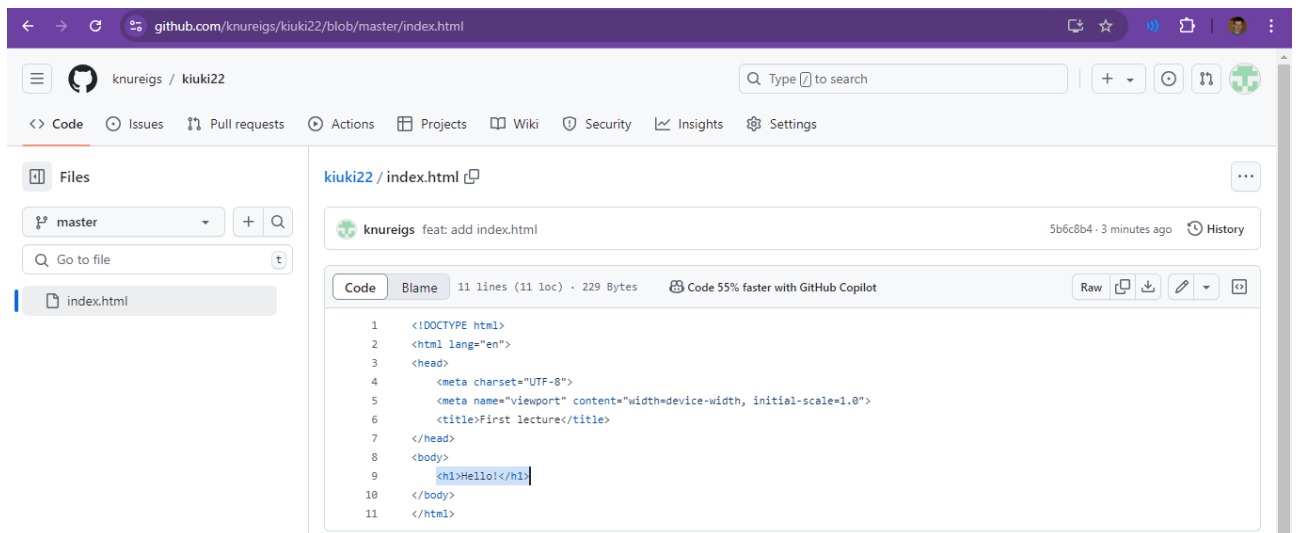
При роботі з віддаленим репозиторієм можливі запити **логіна та пароля**, при цьому пароль зазвичай вводиться користувачем у прихованому режимі. Альтернативне рішення – що дозволяє уникнути проблем з авторизацією під час відправлення змін у віддалений репозиторій – передбачає використання SSH. Згенерувати ключ можна засобами утиліти Git GUI, після чого ключ вказується в налаштуваннях віддаленого репозиторію.

Перевірити, чи актуальна версія локального репозиторію порівняно з віддаленим, можна командою **git remote show origin**. Ця команда повертає багато корисної інформації, але головне буде наприкінці – «up to date», якщо локальний репозиторій є актуальним у порівнянні з віддаленим (що не означає, що він актуальний у порівнянні з вашою робочою папкою – для цього скористайтеся командою `git status`) або "Out of date" – якщо ваша локальна версія встигла застаріти.

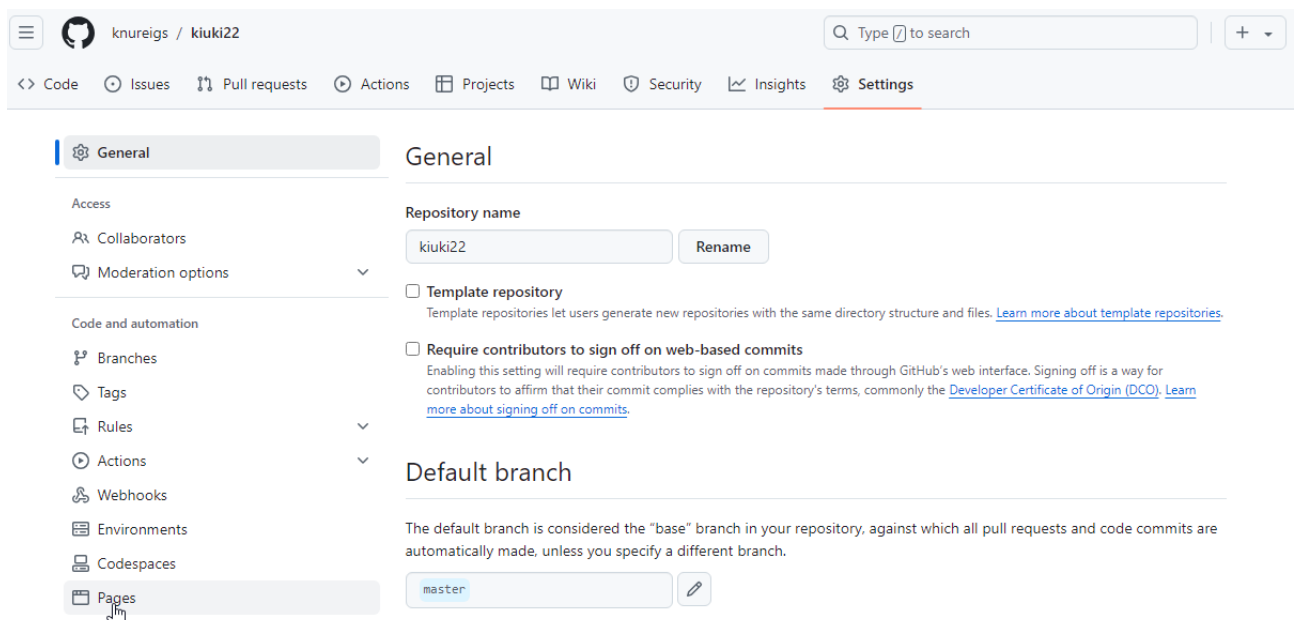
Тепер віддалений репозиторій заповнений, він відповідає локальній версії вихідного коду проєкту, і в контексті лабораторної роботи потрібно відкрити доступ за посиланням (як до звичайного ресурсу), використовуючи GitHub Pages. Повертаємось на <https://github.com> у свій репозиторій:



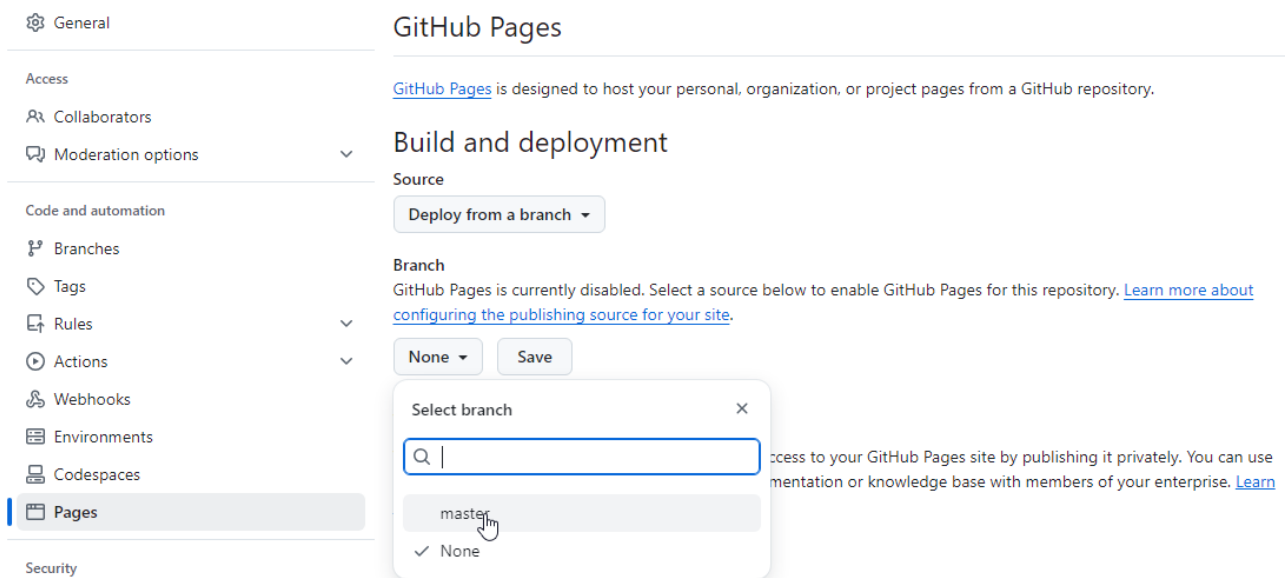
Тут тепер можна побачити, що репозиторій не пустий, у ньому є один файл, який був доданий у комміті під назвою «feat: add index.html». Якщо подивитися всередину файлу, який зараз зберігається у віддаленому репо – то це буде виглядати як звичайний текстовий файл з вихідним кодом:



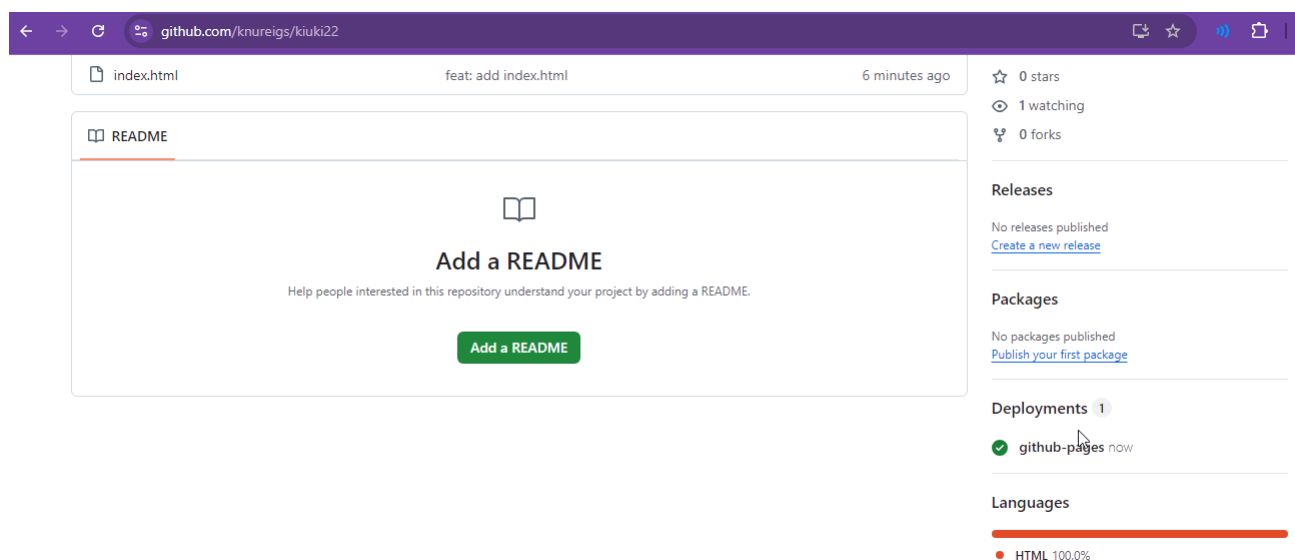
Для того, щоб можна було його сприймати як web-ресурс, потрібно скористатися сервісом GitHub Pages. Вкладка settings:



Знаходимо налаштування «Pages» і там у списку, що випадає, обираємо гілку мастер (або мейн) для публікації і тиснемо «Save»:



Все! Через декілька хвилин на сторінці репозиторія з'явиться пункт «Розгортання» - «Deployments»:



Через який і можна дізнатися посилання на свій ресурс, опублікований у мережі Інтернет:

Deployments

All deployments

Environments

github-pages

Manage environments

github-pages deployments

Latest deployments

✓ github-pages

Last deployed now

<https://knureigs.github.io/kiuki22/>

Filter

Filter deployments

1 deployments

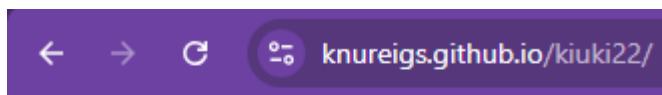
✓

feat: add index.html

Active

Deployed to github-pages by knureigs via pages-build-deployment #1

Перевіряємо доступ. Якщо у вас файл називався інакше, ніж «index.html» - не забуваємо зкорегувати адресу. Також оновлення виконується не миттєво, декілька хвилин при спробах доступу ви можете замість сайту бачити помилку 404. Через декілька хвилин – проста, але повноцінна HTML-сторінка відображається як слід:



Hello!

Далі – згадуємо, що лабораторна присвячена не тільки використанню СКВ Git, сервісу Github та можливості Github Pages, і реалізуємо у вихідному коді ті вимоги, що вказані у лабораторній відповідної тематиці. І після змін у вихідному коді – індексація (git add), комміт (git commit) та відправка (git push). І так при збереженні кожної версії.