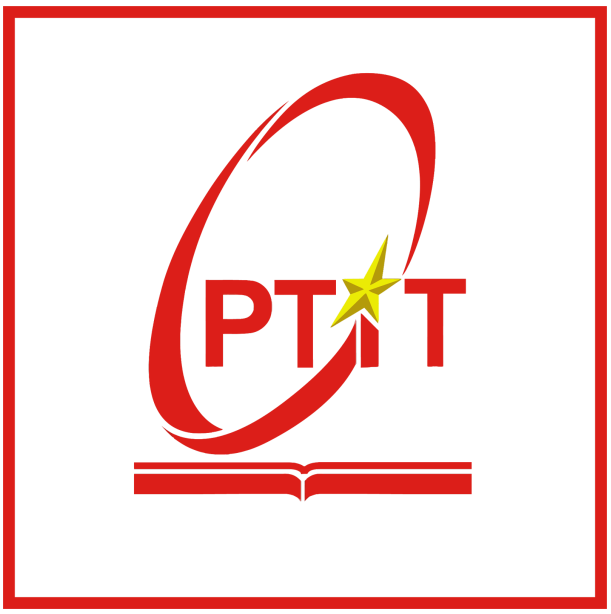
BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO THỰC TẬP CƠ SỞ TUẦN 2

TÌM HIỂU VỀ EJS VÀ GIT, GITHUB

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

- Nguyễn Quang Trung - B22DCDT321

MỤC LỤC

I. **Embedded JavaScript**

II. **JavaScript Object Notation**

III. **Git, Github**

I. Embedded JavaScript

EJS (Embedded JavaScript Templates) là một công cụ templating (Templating languages) phía server được sử dụng trong phát triển web để tạo nội dung HTML động. Nó cho phép nhúng mã JavaScript trực tiếp vào HTML, giúp kết hợp dữ liệu từ server với giao diện người dùng một cách linh hoạt. EJS thường được tích hợp với Node.js và các framework như Express.js, phù hợp cho các ứng dụng web cần hiển thị dữ liệu động.

Các đặc điểm nổi bật của EJS:

- **Cú pháp đơn giản**: Sử dụng cấu trúc HTML thông thường kết hợp với các thẻ EJS đặc biệt.
- **Nhúng mã JavaScript**: Cho phép chèn logic như vòng lặp, điều kiện vào HTML thông qua các thẻ <% %>, <%= %>,...
- **Hỗ trợ Partial Views**: Tái sử dụng các thành phần HTML (như header, footer) bằng cách include file.
- **Tích hợp dễ dàng với Express.js**: Thiết lập nhanh chóng thông qua middleware.

Cú pháp của EJS:

Thẻ	Mục đích	Ví dụ
<% %>	Thực thi code JS (không xuất kết quả)	<% for (let i=0; i<5; i++) { %>
<%= %>	Xuất giá trị (escape HTML)	<%= user.email %>
<%- %>	Xuất giá trị nguyên bản (không escape)	<%- include('header') %>
<%# %>	Chú thích	<%# Đây là comment %>

→ Ứng dụng:

- **Xây dựng trang web động**: Hiển thị dữ liệu từ CSDL (danh sách sản phẩm, bài viết).
- **Tái sử dụng component**: Sử dụng partials để quản lý header, footer.
- **Prototyping nhanh**: Tạo MVP (Minimum Viable Product) với ít code phức tạp.

→ So sánh với các loại Templating Languages khác:

- **Pug (Jade)**: Cú pháp ngắn gọn nhưng cần học lại cách viết HTML.
- **Handlebars**: Tập trung vào logic đơn giản, ít linh hoạt hơn EJS.
- **EJS**: Phù hợp cho người muốn giữ nguyên cấu trúc HTML truyền thống.

→ Ví dụ sử dụng:

Example

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Usage

```
let template = ejs.compile(str, options);
template(data);
// => Rendered HTML string

ejs.render(str, data, options);
// => Rendered HTML string

ejs.renderFile(filename, data, options, function(err, str){
    // str => Rendered HTML string
});
```

```
// index.js
```

```
app.get("/", (req, res) => {
   res.render("index.ejs");
});
```

```
<!-- index.ejs -->
```

```
<% if (fruits) { %>
   <ul>
      <% fruits.forEach((fruit) => { %>
         <li>
            <%= fruit %>
         </li>
      <% }) %>
   </ul>
<% } %>
```

→ Nếu như không có data ?

Trong đoạn mã trên, chúng ta mong đợi một mảng có tên là fruits để lặp qua trong tệp index.ejs. Tuy nhiên, khi kiểm tra index.js, chúng ta nhận ra rằng mặc dù đã render index.ejs, nhưng lại quên truyền dữ liệu vào.

Đây là một lỗi phổ biến có thể xảy ra khi:

- Dữ liệu bị hỏng hoặc không được truyền từ backend.
- Có lỗi trong mã JavaScript khiến biến fruits không được gửi đến template EJS.

Trong JavaScript thông thường, chúng ta có thể xử lý tình huống này bằng cách sử dụng câu lệnh if để kiểm tra xem biến fruits có tồn tại hay không trước khi thực thi các đoạn mã tiếp theo. Tuy nhiên, với EJS, cách tiếp cận này không hoạt động như mong đợi.

Lý do là vì trong EJS, khi gặp một biến không tồn tại, thay vì kiểm tra trước khi sử dụng, nó sẽ ngay lập tức cố gắng truy cập biến đó. Nếu biến chưa được định nghĩa, điều này dẫn đến lỗi và khiến trang web bị sập.

Nói cách khác, **EJS không tự động kiểm tra xem một biến có tồn tại trước khi sử dụng nó, mà thay vào đó, khi không tìm thấy biến, nó sẽ báo lỗi thay vì tiếp tục thực thi mã**. Đây là một đặc điểm của EJS mà có thể gây bất ngờ nếu bạn quen với cách JavaScript thông thường hoạt động.

Locals trong EJS:

Biến gán vào res.locals có thể truy cập ở mọi template EJS mà không cần truyền qua res.render().

→ **Sử dụng res.locals để thiết lập biến cục bộ**

- **res.locals** là một đối tượng trong Express, chứa các biến có thể truy cập trong tất cả các template mà không cần truyền mỗi lần.
- Tất cả các biến truyền vào res.render() đều trở thành locals trong template EJS

```
app.use((req, res, next) => {
  res.locals.siteName = "My Website";
  next();
});

app.get("/", (req, res) => {
  res.render("index", { title: "Trang chủ", user: "Trung" });
});
```

```
<h1>Chào mừng, <%= user %>!</h1>
<p>Website: <%= siteName %></p>
```

→ Bây giờ, trong index.ejs, ta có thể dùng siteName mà không cần truyền vào res.render() mỗi lần

Static files:

Static files (tệp tĩnh) bao gồm các tệp như:

- CSS (stylesheet)
- JavaScript (client-side)
- Hình ảnh (images, icons)

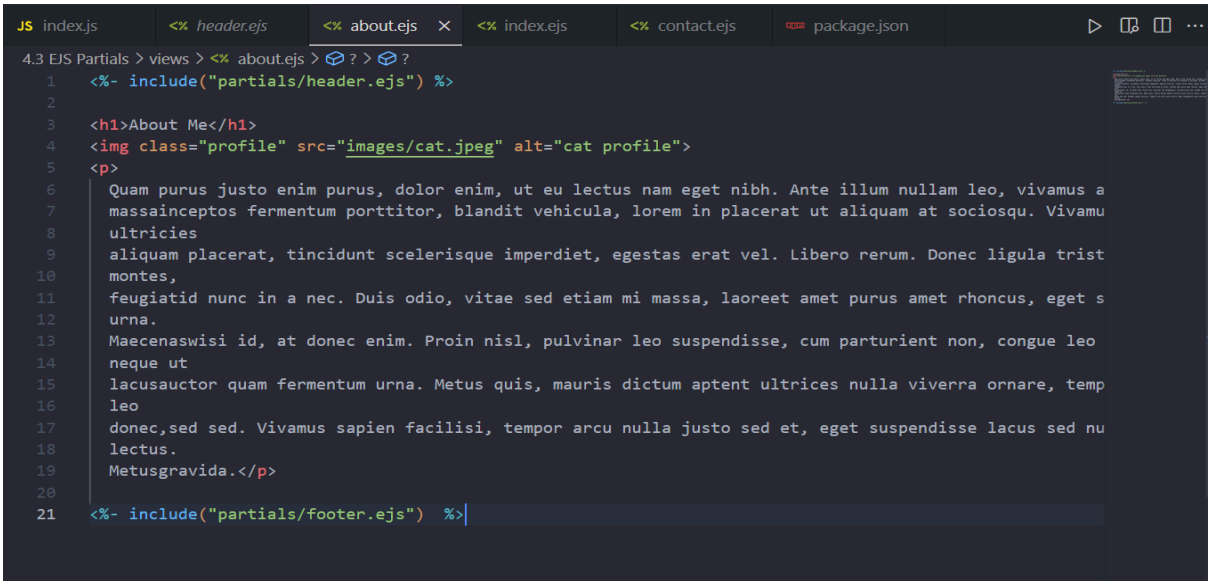
- Fonts
- Tệp khác (PDF, JSON, v.v.)

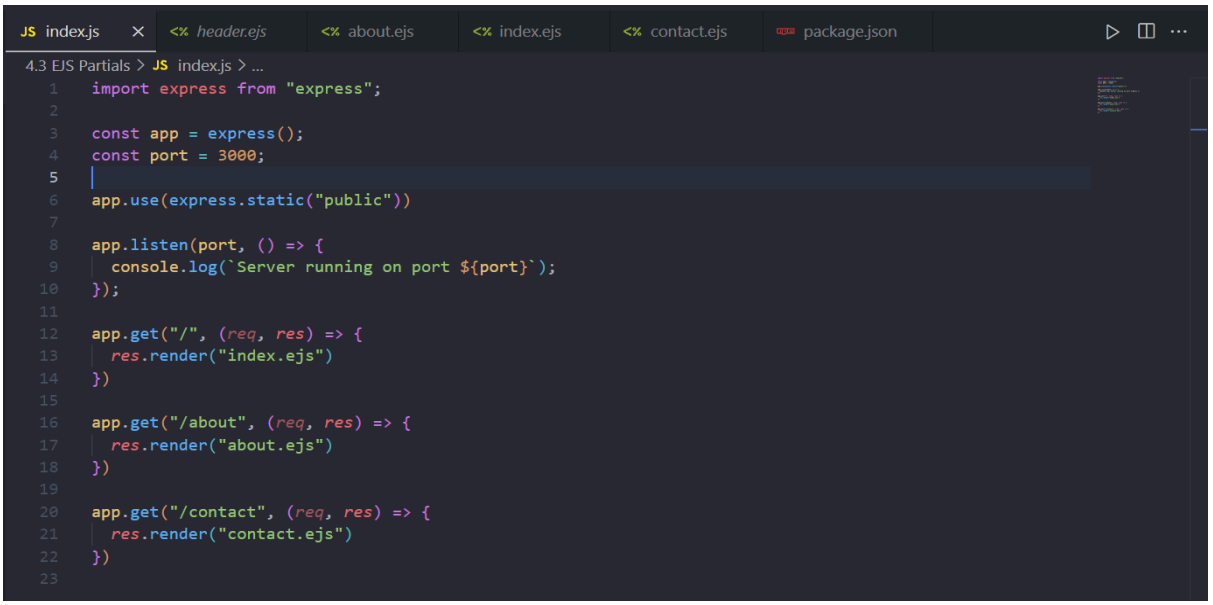
Express không tự động phục vụ các tệp này, bạn cần chỉ định thư mục chứa chúng bằng middleware express.static().

VD: **app.use(express.static("public"))**

→ Có thể sử dụng EJS để tách header và footer của trang web ra thành 2 trang .ejs riêng, sau đó sử dụng <%- %> để link vào các trang khác.

→ Các file .ejs nên được đặt trong thư mục views (theo mặc định của Express)





→ Bài tập thực hành:

Passing data: Sử dụng EJS đếm số ký tự của tên trong form.

- Cài đặt npm, body-parser, ejs và express

package.json

4.2+Passing+Data > package.json > ...

```
{
  "name": "4.2-passing-data",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.3",
    "ejs": "^3.1.10",
    "express": "^4.21.2"
  }
}
```

- Trong file index.ejs tạo ra một form gồm 2 input

index.ejs

4.2+Passing+Data > views > index.ejs > ...

```
<html lang="en">
<body>

    <% if (locals.nameLenLen) { %>

      <h1>Your name has <%= locals.nameLenLen%> characters.</h1>


    <% } else { %>
      <h1>Enter your name below 👇</h1>
    <% } %>

  <!-- Write your code here -->
  <form action="/submit" method="POST">
    <input type="text" name="fName" placeholder="First name">
    <input type="text" name="lName" placeholder="Last name">
    <input type="submit" value="OK">
  </form>
</body>

</html>
```

- Route /submit và truyền giá trị tới index.ejs qua biến nameLenLen

II. JavaScript Object Notation

JSON (JavaScript Object Notation) là một định dạng trao đổi dữ liệu nhẹ gọn, dễ đọc và ghi, được sử dụng rộng rãi trong lập trình và truyền dữ liệu giữa các hệ thống.

Cấu Trúc JSON: JSON bao gồm hai cấu trúc dữ liệu chính:

- Object (Đối tượng): Chứa các cặp key-value, được bao bọc bởi dấu ngoặc nhọn `{}`.
- Array (Mảng): Chứa danh sách giá trị, được bao bọc bởi dấu ngoặc vuông `[]`.

Các Kiểu Dữ Liệu JSON

- String (Chuỗi): "Hello"
- Number (Số): 123, 45.6
- Boolean (Luận lý): true, false
- Null (Giá trị rỗng): null
- Object (Đối tượng): { "name": "John", "age": 30 }
- Array (Mảng): ["Apple", "Banana", "Cherry"]

Ứng Dụng Của JSON

- Web API: JSON là định dạng chính trong trao đổi dữ liệu giữa client và server.
- Lưu Trữ Dữ Liệu: JSON thường được dùng làm file lưu trữ cấu hình hoặc dữ liệu.
- Cơ Sở Dữ Liệu NoSQL: Nhiều hệ quản trị cơ sở dữ liệu như MongoDB sử dụng JSON làm định dạng dữ liệu.

Các phương thức thao tác JSON trong ngôn ngữ lập trình

- JavaScript: JSON.parse(), JSON.stringify().

- Python: Dùng thư viện json với json.loads(), json.dumps().
- Java: Dùng thư viện Jackson hoặc Gson để xử lý JSON.

→ JSON có thể lồng nhau, tức là một đối tượng có thể chứa một mảng hoặc một đối tượng khác.

Các chuẩn mở rộng của JSON

- JSON Schema: Xác định cấu trúc JSON hợp lệ.
- JSON Web Token (JWT): Dùng để xác thực trong hệ thống web.

Nhược điểm của JSON

- Không hỗ trợ kiểu dữ liệu ngày tháng trực tiếp.
- Không hỗ trợ chú thích (comments), khác với XML.
- Không bảo mật nếu không được mã hóa (có thể dễ bị tấn công Injection).

III. Git và Github

Lệnh git:

- **Register user git:** Khai báo user.name và user.email khi dùng Git Bash là cần thiết vì Git cần thông tin này để ghi lại danh tính của người thực hiện các hành động trên kho lưu trữ (repository). Khi bạn thực hiện các thao tác như commit, Git sẽ gắn kèm thông tin này vào lịch sử commit.

  ```
  git config --global user.name trung
  git config --global user.email trung@mail.com
  ```

  Tham số --global áp dụng cấu hình này cho tất cả các dự án Git trên máy tính của bạn.

  Kiểm tra tên, email người dùng hiện tại:

  ```
  git config user.name
  git config user.email
  ```

- **Lệnh cd - Di chuyển đến một thư mục, repo, ... cụ thể:**

  Lệnh cd (Change Directory) được dùng để di chuyển giữa các thư mục trong hệ thống tệp. Khi làm việc với Git, bạn cần dùng cd để di chuyển vào thư mục chứa repository Git của mình trước khi chạy các lệnh Git khác.

  ```
  cd tên_thư_mục
  cd /c/Users/Trung/Documents/GitHub/MyProject
  cd ..  (Quay trở lại thư mục cha)
  cd ~  (Quay trở về thư mục gốc)
  ```

- **Lệnh ls - Liệt kê các thư mục, file, ... trong thư mục**

  Lệnh ls hiển thị danh sách tệp và thư mục có trong thư mục hiện tại.

- **mkdir, touch - Tạo đường dẫn, thư mục và file mới.**

- **rm, rmdir - Xóa đường dẫn, thư mục và file mới.**

- **Lệnh khởi tạo repo : git init**

Các khái niệm cơ bản về Git:

- **Repository - repo, repos:**
  - Container cho codes, project, …
  - Có 2 loại repo: repo local và repo remote
  - File .git trong thư mục là thứ sẽ track những thay đổi trong repo

git add . có thể không thêm các tệp bị xóa.
git add --all chắc chắn sẽ thêm tất cả thay đổi (cả xóa, chỉnh sửa, thêm mới).

Màu xanh: file mới
Màu vàng: file đã có sẵn, đã có thay đổi gì đó trong file

git log xem lịch sử commit
git log --oneline

- **Undoing commit:**
  - **Checkout**
  - **Revert**
  - **Reset**

Tổng quan lại cách sử dụng git:

Bước 1: Khởi tạo tên người dùng và email
Bước 2: Trỏ con trỏ đến folder mình muốn tạo repo local, tạo repo
Bước 3: Đưa file muốn commit lên local repo (đưa vào hàng chờ)
Bước 4: Commit lên local repo
Bước 5: Kết nối local repo với remote repo
Bước 7: Push code lên remote repo
Bước 8: Tạo pull request

Commit histories, head, branch, clone repo.

git log Kiểm tra lịch sử commit

git log --oneline

HEAD là một con trỏ (pointer) đặc biệt đại diện cho **commit hiện tại** mà bạn đang làm việc.

Trong Git, **upstream branch** (nhánh thượng nguồn) là nhánh trên **remote repository** mà nhánh local của bạn theo dõi. Nó giúp Git biết nhánh nào trên remote sẽ nhận code khi bạn push hoặc từ đâu pull code về.

Nhánh (Branch) trong git

- Kiểm tra nhánh:

  git branch

- Tạo nhánh mới:

  git branch *tên_nhánh_mình_muốn_tạo*

- Di chuyển giữa các nhánh:

  git checkout *tên_nhánh_muốn_chuyển*

Git Clone

git clone <url>

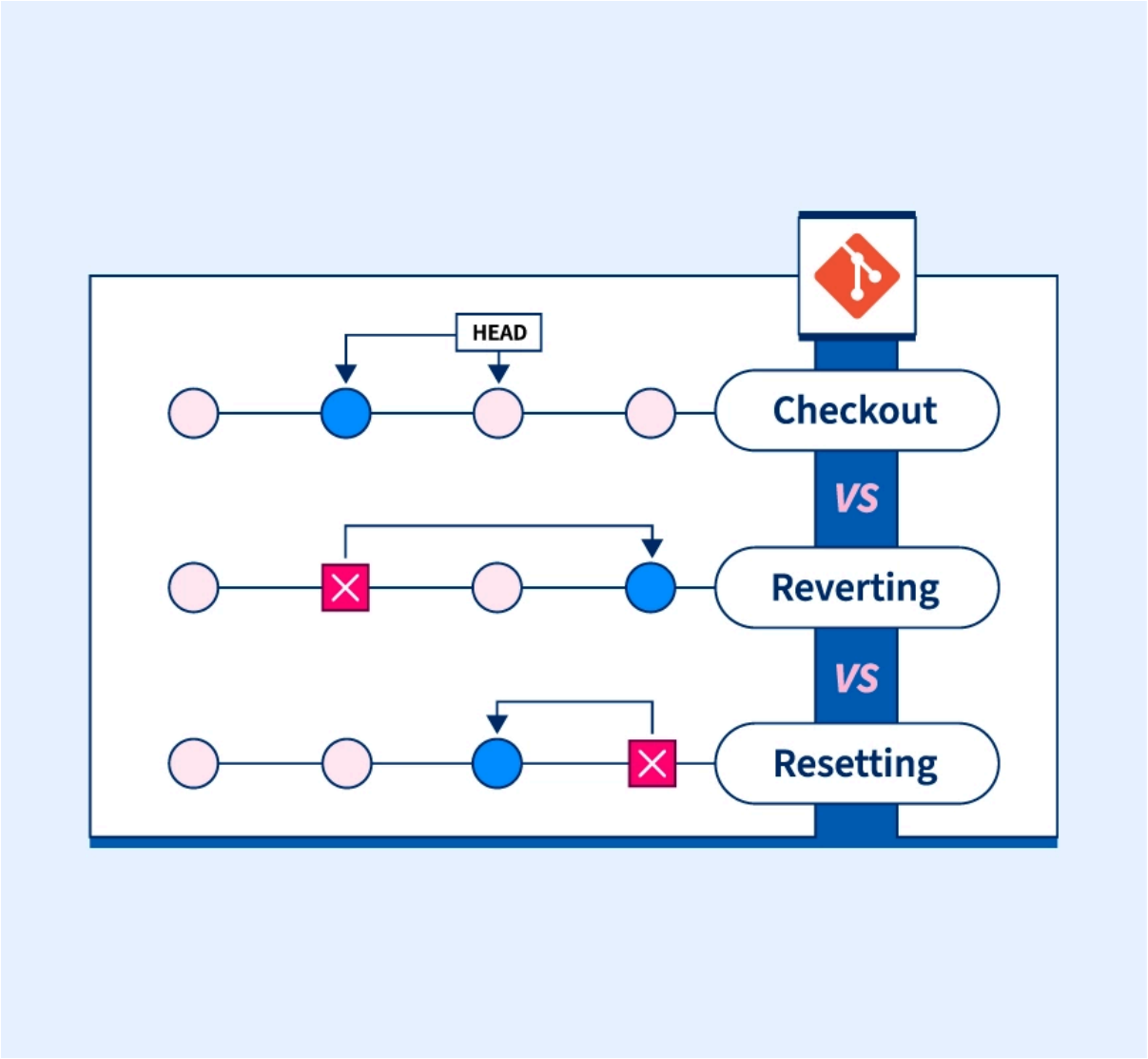
Sau khi clone về trên máy sẽ tự có local repo.

NOTE !

- Trạng thái Detached Head: Điều này có nghĩa là bạn không còn ở trên một nhánh nào nữa, mà chỉ đang xem lại trạng thái của repository tại commit đó.
- git switch -c <*tên-nhánh-mới*>
- **Câu lệnh push pull đúng:**

  git pull/push <*tên_repo_remote*> <*gửi*>:<nhận>

UNDO CHANGES - CHECKOUT, REVERT và RESET



1.A. Checkout: checkout nhánh và checkout commit.

- Lệnh để chuyển đổi nhánh.
- Để tạo và chuyển sang nhánh mới.
- Khôi phục file về trạng thái của commit gần nhất.
- Chuyển về một commit cụ thể (chế độ detached HEAD).

1.B. Revert: đảo ngược (undo) một commit bằng cách tạo một commit mới.

git revert *id_commit*

Dùng revert trong đa số trường hợp không phải commit lại.

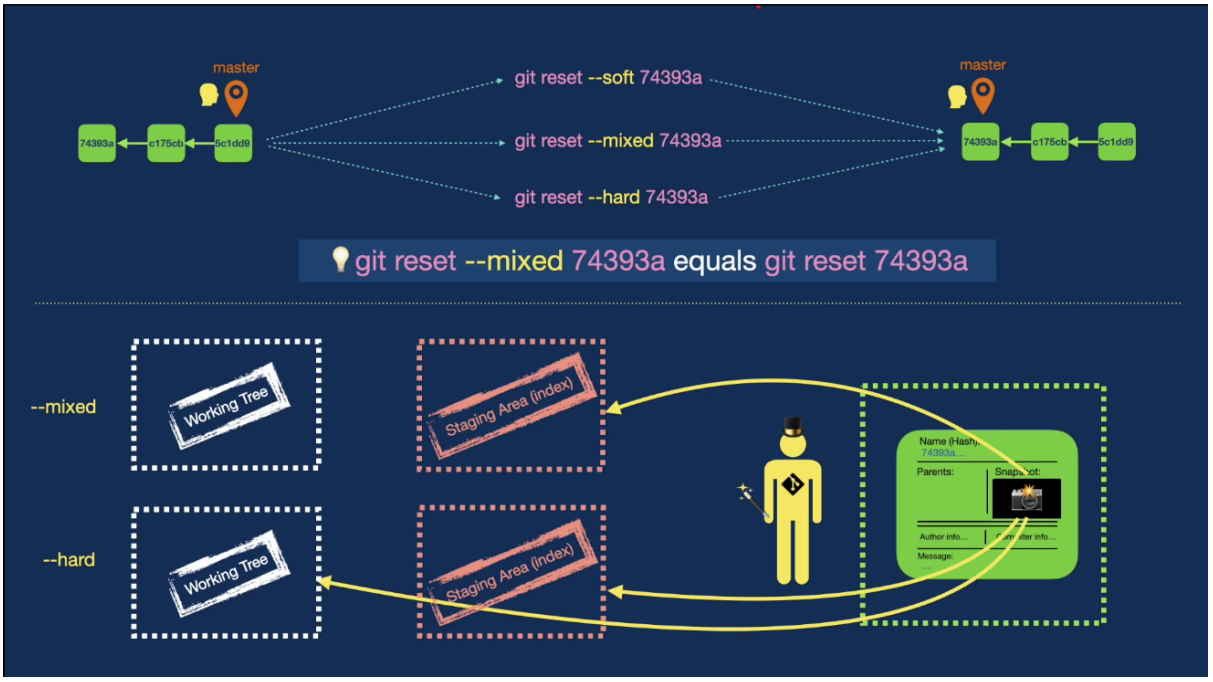
1.C. Reset: quay lại trạng thái trước của code bằng cách xóa hoặc giữ thay đổi tùy vào chế độ reset.

🔥 So sánh `git reset` với `git revert`

Lệnh	Hành động	Ảnh hưởng đến lịch sử commit	Ảnh hưởng đến working directory
`git reset --soft`	Quay lại commit cũ, giữ thay đổi trong staging	❌ Xóa commit nhưng không mất code	❌ Không thay đổi
`git reset --mixed`	Quay lại commit cũ, bỏ staging nhưng giữ code	❌ Xóa commit nhưng không mất code	❌ Không thay đổi
`git reset --hard`	Quay lại commit cũ và xóa sạch thay đổi	🚨 Xóa commit vĩnh viễn	🚨 Xóa tất cả thay đổi chưa commit
`git revert`	Tạo commit mới để đảo ngược commit cũ	✅ Giữ lịch sử commit	❌ Không thay đổi file đã commit

👉 Dùng `git reset` khi làm việc trên local repo, chỉ cần chỉnh sửa commit cá nhân.

👉 Dùng `git revert` khi làm việc với repository chung, cần giữ lịch sử commit rõ ràng.



References:

- Di chuyển giữa các nhánh:
  Git - 1x: Di chuyển giữa các nhánh - git checkout
- Di chuyển giữa các commit:
  Git - 16: Lệnh git checkout [mã commit]