

1

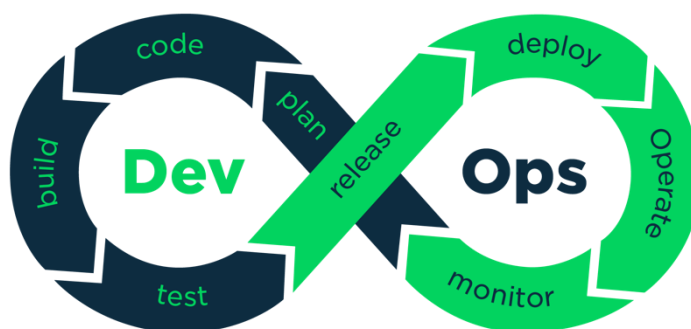
Lab

PHỤC VỤ MỤC ĐÍCH GIÁO DỤC  
FOR EDUCATIONAL PURPOSE

# AUTOMATING EVERYTHING AS CODE

DevOps/CI-CD/Git

Thực hành Lập trình An toàn & Khai thác lỗ hổng phần mềm



Lưu hành nội bộ

<Ng nghiêm cấm đăng tải trên internet dưới mọi hình thức>

## A TỔNG QUAN

### A.1 Mục tiêu

- Giới thiệu

### A.2 Thời gian thực hành

- Thực hành tại lớp: **5** tiết tại phòng thực hành.
- Hoàn thành báo cáo kết quả thực hành: tối đa **13** ngày.

### A.3 Môi trường thực hành

Sinh viên cần chuẩn bị trước máy tính với môi trường thực hành như sau:

1 PC cá nhân với hệ điều hành tự chọn, trong đó có 1 máy ảo **Linux** cài đặt Docker (hoặc Windows Subsystem Linux version2)

### A.4 Các tài nguyên được cung cấp sẵn

- Dành cho Phần B.2: source ứng dụng **sample-app.zip**.
- Dành cho Phần B.3: **unittest.tar.gz**

## B THỰC HÀNH

### B.1 Quản lý phiên bản phần mềm với Git

**Yêu cầu 1.** Sinh viên thực hiện các bước bên dưới để thiết lập hệ thống quản lý phần mềm với Git trên máy ảo, báo cáo kết quả của các bước và trả lời các câu hỏi.

#### B.1.1 Thiết lập Git Repository

**Bước 1.** Cấu hình thông tin người sử dụng và liên kết tài khoản trong local repository.

Lưu ý: sử dụng user.name là **tên nhóm sinh viên**, ví dụ “Nhóm 01” và email của đại diện nhóm.

```
$ git config --global user.name "<username>"
$ git config --global user.email <email>
```

Ví dụ:

```
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ git config --global user.name "Nhóm 00"
ubuntu@ubuntu:~$ git config --global user.email 13520260@gm.uit.edu.vn
```

**Bước 2.** Kiểm tra lại thông tin người sử dụng với lệnh

```
$ git config --list
```

```
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ git config --list
user.name=Nhóm 00
user.email=13520260@gm.uit.edu.vn
ubuntu@ubuntu:~$
```

**Bước 3.** Tạo thư mục có tên **NhómX**, với X là số thứ tự nhóm ở dạng 2 chữ số, và di chuyển đến thư mục vừa tạo

```
$ mkdir NhómX && cd NhómX
```

**Bước 4.** Trong thư mục **NhómX**, tiếp tục tạo thư mục **git-intro** và di chuyển vào

```
$ mkdir git-intro && cd git-intro
```

Đây là thư mục sẽ được sử dụng làm Git repository cục bộ trên máy sinh viên.

```
ubuntu@ubuntu: ~/Nhóm00/git-intro
ubuntu@ubuntu:~$ mkdir Nhóm00 && cd Nhóm00
ubuntu@ubuntu:~/Nhóm00$ mkdir git-intro && cd git-intro
ubuntu@ubuntu:~/Nhóm00/git-intro$
```

**Bước 5.** Khởi tạo thư mục hiện tại (**git-intro**) dưới dạng Git repository với lệnh:

```
$ git init
```

Output của lệnh trên cho biết ta đã tạo một local repository được chứa trong thư mục **.git**. Đây là nơi chứa tất cả lịch sử thay đổi của code.

Có thể xem lại các thư mục đã được tạo sau lệnh **git init** bằng lệnh:

```
$ ls -a
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git init
Initialized empty Git repository in /home/ubuntu/Nhom00/git-intro/.git/
ubuntu@ubuntu:~/Nhom00/git-intro$ ls -a
.  ..  .git
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 6.** Xem trạng thái của repository cục bộ với lệnh

```
$ git status
```

Lệnh này cho phép kiểm tra các tệp nào đã bị thay đổi, hữu ích khi ta làm việc trong dự án và chỉ muốn commit một vài tệp tin chứ không phải toàn bộ.

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
ubuntu@ubuntu:~/Nhom00/git-intro$
```

Output của lệnh git status hiển thị các tệp tin đã sửa đổi trong thư mục, được chuẩn bị cho lần commit sắp tới. Ở ví dụ trên, một số thông tin hiển thị bao gồm:

- Nhánh (branch) đang làm việc: master
- Chưa có commit nào, commit này là Initial commit (lần đầu)
- Không có gì thay đổi trong commit.

### B.1.2 Staging và Committing một tập tin trên Repository

**Bước 1.** Trong thư mục **git-intro**, tạo 1 tệp tin tên **README.MD**, có nội dung là tên nhóm và danh sách thành viên của nhóm.

**Bước 2.** Kiểm tra tệp tin vừa tạo.

Kiểm tra tệp tin đã được tạo trong thư mục với lệnh:

```
$ ls -la
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ ls -la
total 16
drwxrwxr-x 3 ubuntu ubuntu 4096 Sep 11 08:50 .
drwxrwxr-x 3 ubuntu ubuntu 4096 Sep 11 08:47 ..
drwxrwxr-x 7 ubuntu ubuntu 4096 Sep 11 08:48 .git
-rw-rw-r-- 1 ubuntu ubuntu  76 Sep 11 08:50 README.MD
ubuntu@ubuntu:~/Nhom00/git-intro$
```

Kiểm tra nội dung của tệp tin với lệnh:

```
$ cat README.MD
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ cat README.MD
Nhom 00
20520xxx - Tran Van A
20520yyy - Bui Thi B
20520zzz - Nguyen Ngoc C
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 3.** Kiểm tra trạng thái Repository sau khi tạo tập tin.

```
$ git status
```

**Minh chứng cho thấy Git đã tìm thấy 1 tập tin mới nhưng chưa được giám sát?**

**Bước 4.** Staging tập tin – Đưa tập tin mới vào vùng staging

Trước khi commit, tập tin cần được đưa vào vùng staging, sử dụng lệnh

```
$ git add README.MD
```

Lệnh này tạo một snapshot cho tệp. Mọi thay đổi của tệp này sẽ được ghi nhận để chuẩn bị commit.

**Bước 5.** Kiểm tra trạng thái Repository sau khi ‘stage’ tập tin.

```
$ git status
```

**Minh chứng cho thấy Git thấy file mới README.MD trên vùng staging?**

**Bước 6.** Commit tập tin

Đối với các thay đổi đã được đưa vào vùng ‘staging’, cần thực hiện tiếp bước Commit để Git thay đổi những thay đổi đó, với lệnh git commit như sau:

*Lưu ý: thay NhomX tương ứng với nhóm.*

```
$ git commit -m "Committing README.MD from NhomX to begin tracking changes"
```

Trong đó:

- Tùy chọn -m cho phép thêm thông điệp giải thích những thay đổi đang thực hiện.
- Lưu ý ở output bên dưới, số và chữ được highlight là commit ID. Mọi commit được xác định bằng một hàm băm SHA1 duy nhất. Commit ID là 7 ký đầu tiên trong commit hash.

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git commit -m "Committing README.MD fr
om Nhom00 to begin tracking changes"
[master (root-commit) 3680d92] Committing README.MD from Nhom00 to begin
tracking changes
1 file changed, 4 insertions(+)
create mode 100644 README.MD
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 7.** Xem lịch sử commit

```
$ git log
```

Lệnh này hiển thị tất cả các commit trong lịch sử của nhánh hiện tại. Theo mặc định, tất cả các commit là được thực hiện cho nhánh master.

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git log
commit 3680d92243b0209b1971d54bf04eef4bd11e0aac (HEAD -> master)
Author: Nhom 00 <13520260@gm.uit.edu.vn>
Date:   Sun Sep 11 08:54:01 2022 -0700

    Committing README.MD from Nhom00 to begin tracking changes
ubuntu@ubuntu:~/Nhom00/git-intro$
```

### B.1.3 Sửa đổi tập tin và theo dõi các thay đổi

#### Bước 1. Sửa đổi tập tin README.MD

Chèn thêm 1 dòng bất kỳ vào cuối tập tin README.MD, sử dụng lệnh **echo** và dấu ">>" như bên dưới.

Lưu ý: Dùng nhảm dấu ">" sẽ ghi đè lên tập tin hiện có.

```
$ echo "I am beginning to understand Git" >> README.MD
```

#### Bước 2. Xem lại nội dung tập tin đã chỉnh sửa.

```
$ cat README.MD
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ echo "I am beginning to understand Git" >> README.MD
ubuntu@ubuntu:~/Nhom00/git-intro$ cat README.MD
Nhom 00
20520xxx - Tran Van A
20520yyy - Bui Thi B
20520zzz - Nguyen Ngoc C
I am beginning to understand Git
ubuntu@ubuntu:~/Nhom00/git-intro$
```

#### Bước 3. Kiểm tra thay đổi đối với repository

```
$ git status
```

Minh chứng cho thấy Git đã thấy các thay đổi mới chưa được commit?

#### Bước 4. Stage và commit tập tin đã thay đổi

Sử dụng các lệnh:

```
$ git add README.MD
$ git commit -m "NhomX Added additional line to file"
```

Git đưa ra các output như thế nào để hiển thị các thay đổi trong tập tin vừa commit?

#### Bước 5. Xem lại commit vừa thực hiện trong repository

```
$ git log
```

```

ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git log
commit 26e0a0c7dc35afc054d67a503fbc72e0053dabe5 (HEAD -> master)
Author: Nhom 00 <13520260@gm.uit.edu.vn>
Date:   Sun Sep 11 08:56:58 2022 -0700

    Nhom00 Added additional line to file

commit 3680d92243b0209b1971d54bf04eef4bd11e0aac
Author: Nhom 00 <13520260@gm.uit.edu.vn>
Date:   Sun Sep 11 08:54:01 2022 -0700

    Committing README.MD from Nhom00 to begin tracking changes
ubuntu@ubuntu:~/Nhom00/git-intro$

```

Cho biết thông tin về commit vừa thực hiện: Commit ID, thời gian commit, thông điệp commit?

**Bước 6.** So sánh các commit. Khi có nhiều commit trong log, có thể so sánh khác biệt giữa hai commit bằng lệnh:

```
$ git diff <commit ID 1> <commit ID 2>
```

Sinh viên thử so sánh 2 commit đã tạo, giải thích ngắn gọn ý nghĩa các ký hiệu trong output?

```

ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git diff 3680d92 26e0a0c
diff --git a/README.MD b/README.MD
index a7f8f07..aa208a2 100644
--- a/README.MD
+++ b/README.MD
@@ -2,3 +2,4 @@ Nhom 00
 20520xxx - Tran Van A
 20520yyy - Bui Thi B
 20520zzz - Nguyen Ngoc C
+I am beginning to understand Git
ubuntu@ubuntu:~/Nhom00/git-intro$

```

### B.1.4 Branches và Merging (nhánh và hợp nhất)

#### a) Làm việc trong branch

Khi một repository được tạo ra, tất cả các tập tin sẽ được đưa vào một nhánh chính là master. Việc phân nhánh sẽ giúp kiểm soát và thực hiện thay đổi trong một vùng mà không hưởng nhánh chính, tránh code bị ghi đè không mong muốn.

**Bước 1.** Tạo branch mới

Tạo branch mới **feature** với câu lệnh:

```
$ git branch feature
```

**Bước 2.** Kiểm tra branch hiện tại

```
$ git branch
```



Lệnh **git branch** (không kèm theo tên branch) sẽ hiển thị tất cả các branch cho repository này. Trong đó, nhánh có ký hiệu dấu \* phía trước là nhánh hiện tại đang làm việc (đang checkout).

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git branch feature
ubuntu@ubuntu:~/Nhom00/git-intro$ git branch
  feature
* master
ubuntu@ubuntu:~/Nhom00/git-intro$
```

### Bước 3. Chuyển (checkout) branch mới

Để chuyển sang nhánh mới, ví dụ nhánh **feature**, sử dụng lệnh:

```
$ git checkout <branch-name>
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git checkout feature
Switched to branch 'feature'
```

### Bước 4. Kiểm tra lại branch hiện tại

Chạy lại lệnh

```
$ git branch
```

**Dấu hiệu nào cho thấy đã chuyển sang nhánh mới là feature?**

### Bước 5. Thay đổi tập tin README.MD, stage và git trên branch feature

**Sinh viên báo cáo các lệnh và kết quả thực hiện các bước:**

- Thêm một đoạn text mới dạng "... from branch feature" vào tập tin README.MD.
- Stage tập tin vừa thay đổi trong branch feature.
- Kiểm tra trạng thái repository sau khi stage tập tin.
- Commit tập tin trong branch feature với thông điệp "Added a third line in feature branch".
- Xem lịch sử các commit đã thực hiện, ***minh chứng cho thấy có 1 commit tại nhánh feature?***

### b) Merge các thay đổi từ các branch vào branch master

#### Bước 1. Chuyển (checkout) sang branch master

```
$ git checkout master
```

**Bước 2.** Kiểm tra file README.MD ở nhánh master để xem nội dung của nó có bị ảnh hưởng bởi thay đổi ở nhánh feature.

**Bước 3.** Merge tất cả các nội dung tập tin thêm từ branch feature sang master

Các branch thường được sử dụng khi triển khai các tính năng mới hoặc sửa lỗi. Các thay đổi có thể được gửi cho các thành viên trong nhóm phát triển xem xét và thống nhất, sau đó có thể gộp vào branch chính master.



Để merge nội dung từ 1 branch về master có thể sử dụng lệnh:

```
$ git merge <branch-name>
```

Trong đó, branch-name là branch muốn pull về branch hiện tại.

Báo cáo kết quả thực hiện câu lệnh merge branch feature về branch master?

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git merge feature
Updating 26e0a0c..94e4c34
Fast-forward
 README.MD | 1 +
 1 file changed, 1 insertion(+)
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 4.** Kiểm tra nội dung của file README.MD khi merge

Mở tập tin README.MD, so sánh nội dung của nó với kết quả sau lần thay đổi cuối ở branch master ở phần **Bước 2**?

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ cat README.MD
Nhom 00
20520xxx - Tran Van A
20520yyy - Bui Thi B
20520zzz - Nguyen Ngoc C
I am beginning to understand Git
Hello world from branch feature
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 5.** Xoá branch

Kiểm tra branch feature vẫn còn tồn tại bằng lệnh

```
$ git branch
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git branch
feature
* master
ubuntu@ubuntu:~/Nhom00/git-intro$
```

Có thể xoá branch bằng lệnh sau, trong đó <branch-name> là tên branch muốn xoá. Lưu ý, phải chuyển sang branch khác trước khi muốn xoá 1 branch, Git không cho phép xoá branch đang sử dụng.

```
$ git branch -d <branch-name>
```

Xoá branch feature và kiểm tra lại branch này còn tồn tại không?

### B.1.5 Xử lý xung đột khi merge

Đôi khi sẽ có trường hợp xảy ra conflict (xung đột) khi merge. Ví dụ, khi ta thực hiện các thay đổi chồng chéo đối với tập tin, có thể Git sẽ không tự động merge những thay đổi này được.

**Bước 1.** Tạo branch test mới và chuyển sang branch test vừa tạo

```
$ git branch test
$ git checkout test
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git branch test
ubuntu@ubuntu:~/Nhom00/git-intro$ git checkout test
Switched to branch 'test'
```

**Bước 2.** Chỉnh sửa nội dung tập tin README.MD với lệnh sau:

```
$ sed -i 's/feature/test/' README.MD
```

Giải thích ý nghĩa của dòng lệnh trên? Minh chứng sự thay đổi trước và sau khi chạy dòng lệnh?

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ sed -i 's/feature/test/' README.MD
ubuntu@ubuntu:~/Nhom00/git-intro$ cat README.MD
Nhom 00
20520xxx - Tran Van A
20520yyy - Bui Thi B
20520zzz - Nguyen Ngoc C
I am beginning to understand Git
Hello world from branch test
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 3.** Thực hiện stage, commit branch test

```
$ git commit -a -m "Change feature to test"
```

Lưu ý: option -a chỉ ảnh hưởng đến các tập tin đã được sửa đổi và xóa. Nó không ảnh hưởng đến các tập tin mới.

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git commit -a -m "Change feature to test"
[test 1bfc348] Change feature to test
1 file changed, 1 insertion(+), 1 deletion(-)
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 4.** Chuyển sang branch master và thực hiện chỉnh sửa tập tin README.MD.

```
$ git checkout master
$ sed -i 's/feature/master/' README.MD
```

Nội dung tập tin README.MD thay đổi như thế nào?

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git checkout master
Switched to branch 'master'
ubuntu@ubuntu:~/Nhom00/git-intro$ sed -i 's/feature/master/' README.MD
ubuntu@ubuntu:~/Nhom00/git-intro$ cat README.MD
Nhom 00
20520xxx - Tran Van A
20520yyy - Bui Thi B
20520zzz - Nguyen Ngoc C
I am beginning to understand Git
Hello world from branch master
```

**Bước 5. Stage và commit branch master**

```
$ git commit -a -m "Changed feature to master"
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git commit -a -m "Change feature to master"
[master ea56d52] Change feature to master
1 file changed, 1 insertion(+), 1 deletion(-)
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 6. Merge hai branch test và master**

```
$ git merge test
```

Kết quả không thể merge do có xung đột.

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git merge test
Auto-merging README.MD
CONFLICT (content): Merge conflict in README.MD
Automatic merge failed; fix conflicts and then commit the result.
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 7. Tìm xung đột**

- Xem các commit. Lưu ý rằng phiên bản HEAD là branch master, thông tin này sẽ hữu ích trong bước tiếp theo.

```
$ git log
```

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ git log
commit ea56d52fea25752055be093a98afd94eadf7d1a4 (HEAD -> master)
Author: Nhom 00 <13520260@gm.uit.edu.vn>
Date: Sun Sep 11 09:18:46 2022 -0700

    Change feature to master
```

- Tập README.MD sẽ chứa thông tin tìm ra xung đột. Phiên bản HEAD (branch master) cho chứa từ "master" đang xung đột với phiên bản branch test là từ "test".

```
ubuntu@ubuntu: ~/Nhom00/git-intro
ubuntu@ubuntu:~/Nhom00/git-intro$ cat README.MD
Nhom 00
20520xxx - Tran Van A
20520yyy - Bui Thi B
20520zzz - Nguyen Ngoc C
I am beginning to understand Git
<<<<<< HEAD
Hello world from branch master
=====
Hello world from branch test
>>>>>> test
ubuntu@ubuntu:~/Nhom00/git-intro$
```

**Bước 8. Chỉnh sửa tập tin README.MD để xóa đoạn xung đột**

Mở tập README.MD và xoá đoạn highlight với các ký hiệu <<, ==, >> và 1 trong 2 dòng được thêm ở branch master hoặc test (có thể tùy chọn). Lưu lại và kiểm tra kết quả.

```
...
I am beginning to understand Git
<<<<<<< HEAD
... from branch master
=====
... from branch test
>>>>>> test
```

**Bước 9.** Thực hiện stage, commit và kiểm tra commit branch master

```
$ git add README.MD
$ git commit -a -m "Manually merged from test branch"
```

**Kết quả commit? Kiểm tra log commit của branch master?**

### B.1.6 Tích hợp Git với Github

Hiện tại, những thay đổi chỉ được lưu trữ tại máy. Git chạy cục bộ và không yêu cầu bất kỳ máy chủ hoặc dịch vụ lưu trữ đám mây nào. Git cho phép người dùng lưu trữ cục bộ và quản lý tập tin.

Việc sử dụng GitHub sẽ giúp làm việc nhóm tốt hơn và tránh tình trạng mất dữ liệu. Có một số dịch vụ Git khá phổ biến bao gồm GitHub, Stash từ Atlassian và GitLab.

**Yêu cầu 2.** Sinh viên thực hiện các bước bên dưới để thiết lập hệ thống quản lý phần mềm với Git kết hợp GitHub, báo cáo kết quả của các bước.

- Bước 1: Tạo tài khoản GitHub
- Bước 2: Đăng nhập tài khoản GitHub
- Bước 3: Tạo Repository tên **NT521-TeamX** có chế độ Private
- Bước 4: Tạo thư mục **NT521-TeamX** trên máy ảo
- Bước 5: Sao chép tập tin **README.MD** vào thư mục vừa tạo
- Bước 6: Khởi tạo Git repository từ thư mục **NT521-TeamX**

Lưu ý: Cần điều chỉnh user.name và user.email cho khớp với tài khoản GitHub.

- Bước 7: Trỏ Git repository đến GitHub repository

Gợi ý: sử dụng lệnh **git remote add origin <GitHub link>**

- Bước 8: Stage, commit README.MD với thông điệp "Added to NT521-TeamX"
- Bước 9: Kiểm tra log commit
- Bước 10: Gửi (push) tập tin từ Git lên GitHub với lệnh

Gợi ý: dùng lệnh **git push origin master**, dùng token thay vì password.

- Bước 11: Kiểm tra tập tin trên GitHub

## B.2 Test Python Function với unittest

Sử dụng unittest để kiểm tra function có chức năng tìm kiếm đệ quy JSON object. Hàm trả về giá trị được gắn thẻ bằng một khoá cố định. Lập trình viên thông thường thực hiện hành động trên JSON object trả về bởi lời gọi API.

Bài test này sẽ sử dụng 3 tập tin như sau:

Tập tin	Chú thích
<b>recursive_json_search.py</b>	Đoạn mã này gồm function json_search() mà ta muốn test
<b>test_data.py</b>	Đây là dữ liệu mà hàm json_search() đang tìm kiếm
<b>test_json_search.py</b>	Đây là tập ta sẽ tạo để kiểm tra hàm json_search() trong tập mã recursive_json_search.py.

**Bước 1.** Xem tập tin test\_data.py

Mở *unittest/test\_data.py* và kiểm tra nội dung của nó. Dữ liệu JSON này là điển hình của dữ liệu trả về. Dữ liệu mẫu đủ phức tạp để trở thành một bài test tốt. Ví dụ, nó có các loại dict và list xen kẽ.

```
GNU nano 4.8 test_data.py
key1 = "issueSummary"
key2 = "XY&^$#@!1234%^&"

data = {
    "id": "AWcvsjx864kVeDHDi2gB",
    "instanceId": "E-NETWORK-EVENT-AWcvsjx864kVeDHDi2gB-1542693469197",
    "category": "Warn",
    "status": "NEW",
    "timestamp": 1542693469197,
    "severity": "P1",
    "domain": "Availability",
    "source": "DNAC",
    "priority": "P1",
    "type": "Network",
    "title": "Device unreachable",
    "description": "This network device leaf2.abc.inc is unreachable from controller.",
    "actualServiceId": "10.10.20.82",
    "assignedTo": "",
    "enrichmentInfo": {
        "issueDetails": {
```

**Bước 2.** Tạo hàm json\_search() mà ta sẽ kiểm tra.

Mở tập tin **unittest/recursive\_json\_search.py** và thêm hàm bên dưới.

```
from test_data import *
def json_search(key,input_object):
    ret_val=[]
    if isinstance(input_object, dict): # Iterate dictionary
        for k, v in input_object.items(): # searching key in the dict
            if k == key:
                temp={k:v}
                ret_val.append(temp)
            if isinstance(v, dict): # the value is another dict so repeat
```

```

        json_search(key,v)
    elif isinstance(v, list): # it's a list
        for item in v:
            if not isinstance(item, (str,int)): # if dict or list
                repeat
                    json_search(key,item)
            else: # Iterate a list because some APIs return JSON object in a list
                for val in input_object:
                    if not isinstance(val, (str,int)):
                        json_search(key,val)
        return ret_val
print(json_search("issueSummary",data))

```

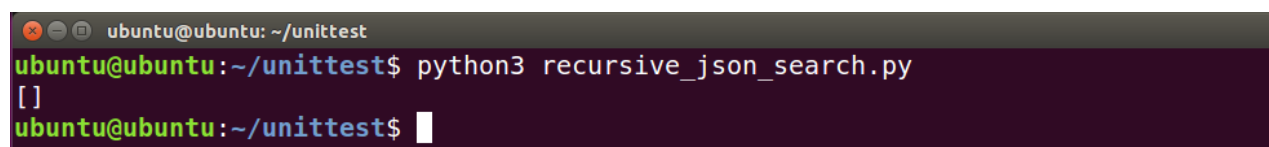
Hàm `json_search` nhận một khoá (key) và một JSON object làm tham số đầu vào, sau đó trả về list các cặp key/value. Phiên bản hiện tại của function cần kiểm tra để xem nó có hoạt động đúng như dự định hay không.

Hoạt động của function này:

- Trước tiên nhận dữ liệu test.
- Tìm kiếm dữ liệu phù hợp các biến key trong tập tin `test_data.py`. Nếu tìm ra một kết quả phù hợp, nó sẽ nối dữ liệu phù hợp vào list.
- Hàm `print()` ở cuối dùng để in list kết quả cho ví dụ `key = "issueSummary"`

**Bước 3.** Lưu và chạy code trên.

```
$ python3 recursive_json_search.py
```



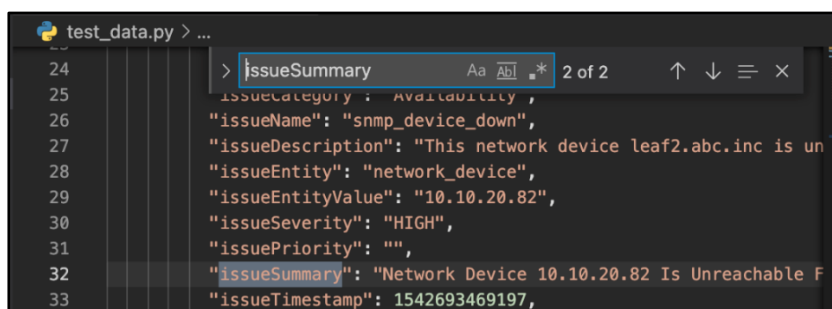
```

ubuntu@ubuntu:~/unittest$ python3 recursive_json_search.py
[]
ubuntu@ubuntu:~/unittest$

```

Kết quả chạy hàm `json_search()` cho thấy, key "issueSummary" không có trong dữ liệu JSON được trả về từ việc gọi API.

**Bước 4.** Kiểm tra lại kết quả của hàm `json_search()` bằng mở tập tin `test_data.py` và tìm kiếm "issueSummary".



```

24  > issueSummary
25  "issueCategory": "Availability",
26  "issueName": "snmp_device_down",
27  "issueDescription": "This network device leaf2.abc.inc is un
28  "issueEntity": "network_device",
29  "issueEntityValue": "10.10.20.82",
30  "issueSeverity": "HIGH",
31  "issuePriority": "",
32  "issueSummary": "Network Device 10.10.20.82 Is Unreachable F
33  "issueTimestamp": 1542693469197,

```

⇒ Đoạn code lỗi.



**Bước 5.** Tạo một unit test để kiểm tra function có hoạt động đúng như dự kiến

- Mở tập tin **test\_json\_search.py**
- Dòng đầu tiên thêm thư viện unittest

```
import unittest
```

- Thêm các dòng import function đang testing cũng như dữ liệu JSON mà hàm sử dụng

```
from recursive_json_search import *
from test_data import *
```

- Thêm đoạn mã vào class json\_search\_test. Mã tạo ra subclass TestCase của unittest framework. Class định nghĩa một số phương pháp kiểm tra được sử dụng trên function json\_search() trong recursive\_json\_search.py. Lưu ý mỗi phương thức test bắt đầu với **test\_**, cho phép unittest framework tự động khám phá. Thêm các dòng sau vào cuối test\_json\_search.py

```
class json_search_test(unittest.TestCase):
    '''test module to test search function in `recursive_json_search.py`'''
    def test_search_found(self):
        '''key should be found, return list should not be empty'''
        self.assertTrue([]!=json_search(key1,data))
    def test_search_not_found(self):
        '''key should not be found, should return an empty list'''
        self.assertTrue([]==json_search(key2,data))
    def test_is_a_list(self):
        '''Should return a list'''
        self.assertIsInstance(json_search(key1,data),list)
```

- Trong code unittest, đang sử dụng ba phương pháp để test function tìm kiếm.
  1. Đưa ra một key tồn tại sẵn trong JSON object, xem đoạn code có thể tìm thấy key như vậy không.
  2. Đưa một key không tồn tại trong JSON object, xem liệu code có xác nhận rằng không có key nào tìm được.
  3. Kiểm tra xem function có trả về một list như mong đợi hay không.

Để tạo các test, đoạn code sử dụng phương thức assert được tích hợp trong class unittest TestCase để kiểm tra các điều kiện. Phương thức assertTrue(x) sẽ kiểm tra một điều kiện có đúng không, assertIsInstance(a,b) kiểm tra xem a có phải là một thể hiện của kiểu b hay không. Thể hiện kiểu ở đây là list.

Lưu ý mỗi phương thức đều có chú thích trong nháy đôi ("), điều này bắt buộc nếu kiểm tra thông tin output của phương pháp test khi chạy.



- Cuối tập tin, thêm phương thức `unittest.main()`, điều này cho phép unittest chạy từ command line

```
if __name__ == '__main__':
    unittest.main()
```

### Bước 6. Chạy test để xem kết quả

- Chạy code test.
- Quan sát kết quả: Đầu tiên ta thấy list trống, thứ hai ta thấy **.F.**. Dấu chấm có nghĩa test passed và điểm F có nghĩa là test failed. Do đó, phép thử thứ nhất đã passed, phép thử thứ hai test failed, và phép thử thứ ba cũng test passed.

```
ubuntu@ubuntu: ~/unittest
ubuntu@ubuntu:~/unittest$ python3 test_json_search.py
[]
.F.
=====
FAIL: test_search_found ( __main__.json_search_test)
key should be found, return list should not be empty
=====
Traceback (most recent call last):
  File "test_json_search.py", line 9, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true
=====
Ran 3 tests in 0.001s
FAILED (failures=1)
```

- Để liệt kê từng bài test và kết quả của nó, thêm tùy chọn `-v` trong lệnh unittest.

```
ubuntu@ubuntu: ~/unittest
ubuntu@ubuntu:~/unittest$ python3 -m unittest -v test_json_search.py
[]
test_is_a_list (test_json_search.json_search_test)
Should return a list ... ok
test_search_found (test_json_search.json_search_test)
key should be found, return list should not be empty ... FAIL
test_search_not_found (test_json_search.json_search_test)
key should not be found, should return an empty list ... ok
=====
FAIL: test_search_found (test_json_search.json_search_test)
key should be found, return list should not be empty
=====
Traceback (most recent call last):
  File "/home/ubuntu/unittest/test_json_search.py", line 9, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true
=====
Ran 3 tests in 0.003s
FAILED (failures=1)
```

### Bước 7. Điều tra và sửa lỗi trong đoạn mã `recursive_json_search.py`

**key should be found, return list should not be empty ... FAIL** => Khoá không tìm được. Tại sao? Ta nhìn vào đoạn code của hàm đệ quy thấy rằng `ret_val=[]` đang được

thực thi lặp đi lặp lại mỗi lần hàm được gọi. Điều này làm cho list luôn rỗng và làm mất kết quả tích lũy từ lệnh `ret_val.append(temp)` đang thêm vào danh sách `ret_val`.

```
def json_search(key, input_object):
    ret_val=[]
    if isinstance(input_object, dict): # Iterate dictionary
        for k, v in input_object.items(): # searching key in the dict
            if k == key:
                temp={k:v}
                ret_val.append(temp)
```

**Yêu cầu 3.** Sinh viên sửa lại lỗi theo gợi ý trên và thực thi lại unittest. Nếu còn lỗi, tiếp tục chỉnh sửa đến khi pass cả 3 phép thử; trình bày step-by-step có minh chứng.

#### Gợi ý:

- Dòng `ret_val=[]` bị ghi đè giá trị rỗng mỗi lần gọi đệ quy hàm, gây ra lỗi, cần di chuyển `ret_val` ra bên ngoài hàm `json_search`.
- Lưu ý khi di chuyển `ret_val`, nếu biến này trở thành 1 biến toàn cục (global variable), trong 1 lần chạy file, gọi hàm `json_search` nhiều lần sẽ khiến lần gọi sau sử dụng lại kết quả `ret_val` của lần gọi trước. Vậy cần sửa như thế nào?

## C YÊU CẦU & ĐÁNH GIÁ

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Sinh viên báo cáo kết quả thực hiện và nộp bài bằng **1 trong 2 hình thức**:

### C.1 Cách 1: Báo cáo trực tiếp trên lớp

Báo cáo trực tiếp kết quả thực hành (có hình ảnh minh họa các bước) với GVTH trong buổi học, trả lời các câu hỏi và giải thích các vấn đề kèm theo.

### C.2 Cách 2: Nộp file báo cáo

Báo cáo cụ thể quá trình thực hành (có hình ảnh minh họa các bước), trả lời các câu hỏi và giải thích các vấn đề kèm theo.

**Đặt tên file báo cáo theo định dạng như mẫu:**

**[Mã lớp]-LabX\_MSSV1-MSSV2-MSSV3**

Ví dụ: *[NT521.012.ATCL.1]-Lab1\_21520xxx-21520yyy-21520zzz.*

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp báo cáo trên theo thời gian đã thống nhất tại website môn học.

## D ĐÁNH GIÁ

- Sinh viên hiểu và tự thực hiện được bài thực hành, đóng góp tích cực tại lớp.
- Báo cáo trình bày chi tiết, giải thích các bước thực hiện và chứng minh được do nhóm sinh viên thực hiện.
- Hoàn tất nội dung cơ bản và thực hiện nội dung *mở rộng*

**Kết quả thực hành cũng được đánh giá bằng kiểm tra kết quả trực tiếp tại lớp vào cuối buổi thực hành hoặc vào buổi thực hành thứ 2.**

**Lưu ý:** Bài sao chép, nộp trễ, "*gánh team*", ... sẽ được xử lý tùy mức độ.

**HẾT**

*Chúc các bạn hoàn thành tốt!*