SCHOOL OF ENGINEERING, KMITL

//WK04

Advance Computer Programming

For FAs

Topic of intermediate Python Programming

Dr. Sarucha Yanyong,
Department of Robotics and AI Engineering

Advance
Computer
Programming
for RAI

Function

def

;

Purpose

The def keyword is used to define a function in Python. Functions allow you to encapsulate reusable blocks of code, making your programs modular and easier to manage.

Syntax

```
def function_name(parameters):
    # Code block
    return value (optional)
```

def

;

Example

```
def greet(name):
    message = f"Hello, {name}!"
    return message


# Usage
print(greet("Alice"))  # Output: Hello, Alice!
```

In this example, the greet function takes a name parameter, constructs a greeting message, and returns it. The function is then called with the argument "Alice".

Advance
Computer
Programming
for RAI

Anonymous Functions

lambda functions, also known as anonymous functions, allow the creation of small, unnamed functions for short-term use.

lambda

;

Syntax

```
lambda arguments: expression
```

Regular function VS Lambda function

```
# Regular function
def add(x, y):
    return x + y
```

```
# Lambda function
add = lambda x, y: x + y
```

** x, y are defined as input argument for function add.

```
# Regular function
def square(x):
    return x*x
```

```
# Lambda function
square = lambda x: x*x
```

** x is defined as input argument for function square.

lambda

Use cases

Using lambda with map()

To modify element in list with specific algorithm

Example

```
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
# squared = [1, 4, 9, 16, 25]
```

lambda

Use cases

Using lambda with filter()

To filter element in list with specific algorithm

Example

```
numbers = [1, 2, 3, 4, 5]
evens = list(filter(lambda x: x % 2 == 0, numbers))
# evens = [2, 4]
```

lambda

Use cases

Using lambda with sort

To sort element in list with specific algorithm

Example

```
# List of dictionaries
students = [
    {"name": "Alice", "grade": 85},
    {"name": "Bob", "grade": 92},
    {"name": "Charlie", "grade": 78}
]
```

Sort the nestest list of dictionary

```
# Output:
# [
# {'name': 'Charlie', 'grade': 78},
# {'name': 'Alice', 'grade': 85},
# {'name': 'Bob', 'grade': 92}
# ]
```

lambda

Use cases

Using lambda

with sort

Syntax

```
sorted(iterable, key=lambda x: expression)
```

Example

```
sorted = sorted(students, key=lambda student: student["grade"])
```

lambda

Use cases

Example

```
# List of dictionaries
students = [
    {"name": "Alice", "grade": 85},
    {"name": "Bob", "grade": 92},
    {"name": "Charlie", "grade": 78}
]

# Sorting the list of dictionaries by the 'grade' key
sorted_students = sorted(students, key=lambda student: student["grade"])

# Output the sorted list
print(sorted_students)

# Output:
# [{'name': 'Charlie', 'grade': 78}, {'name': 'Alice', 'grade': 85},
{'name': 'Bob', 'grade': 92}]
```

In this example, the lambda function lambda student: student["grade"] is used to extract the grade value from each dictionary in the list. The sorted() function then sorts the list of dictionaries based on these grade values in ascending order.

lambda

with dictionary

Example

```
student = {
 "student_id": 1,
 "grade": 90
}
criteria = lambda student: True if student["grade"] > 80 else False
print(student, criteria(student))
```

Explaination

The given code defines a dictionary named student with keys student_id and grade, holding values 1 and 90 respectively. It then defines a lambda function criteria that takes a student dictionary as input and returns True if the student's grade is greater than 80; otherwise, it returns False. Finally, the code prints the student dictionary and the result of applying the criteria function to the student, which in this case will output True because the student's grade is 90, which is greater than 80.

lambda

with list of dictionary

Example

```
# List of student dictionaries
students = [
{"student_id": 1, "grade": 90},
{"student_id": 2, "grade": 75},
{"student_id": 3, "grade": 85},
{"student_id": 4, "grade": 60},
{"student_id": 5, "grade": 95}
]

criteria = lambda student: True if student["grade"] > 80 else False
iltered_students = list(filter(criteria, students))
# Samilar to
# filtered_students = [student for student in students if criteria(student)]

print(students, iltered_students)
```

Explaination

In this example, we define a lambda function criteria that checks if a student's grade is above 80. Using the filter function, we apply this criteria to the list of students, which returns an iterator of students meeting the criteria. We convert this iterator to a list using the list function, resulting in filtered_students, which contains only the students with grades above 80.

Advance
Computer
Programming
for RAI

Importing Modules in Python

Understand the importance and usage of importing modules in Python to enhance code reusability and organization.

Import

module vs package

Modules

- A single file containing Python code (e.g., module.py).
- Can be imported using

```
import module_name
```

Packages

- A directory containing multiple modules and sub-packages.
- Contains an __init__.py file to be recognized as a package.
- Can be imported using

```
import package_name
```

or

```
from package_name import module_name
```

Import

module vs package

Modules are suitable for smaller projects where a single file can logically encapsulate related functions and classes.

Packages are ideal for larger projects that require multiple modules grouped together, making the codebase more organized and maintainable.

Import

module vs package

directory structure

Modules

```
project/
    main.py
    my_module.py
```

Packages

```
project/
    main.py
    my_package/
        __init__.py
        module1.py
        module2.py
```

** The directory structure for both modules and packages when main.py is the main program.

Module

Creating

Directory structure

project/

main.py

my_module.py

1. Create file my_module.py

(next to the main.py)

```
def greet(name):
 return f"Hello, {name}!"
```

2. Import into main.py

```
import my_module
print(my_module.greet("Alice")) # Outputs: Hello, Alice!
```

Package

Creating

Directory structure

```
project/
    main.py
    my_package/
        __init__.py
        module1.py
        module2.py
```

1. Create file __init__.py as empty file

2. Create file my_package/module1.py

```python
def greet(name):
    return f"Hello, {name}!"
```

3. Create file my_package/module2.py

```python
def bye(name):
    return f"Bye, {name}!"
```

4. Import into main.py

```python
from my_package import greet, bye


print(module1.greet("Alice")) # Outputs: Hello, Alice!
print(module2.bye("Alice"))   # Outputs: Bye, Alice!
```

Git Repositories

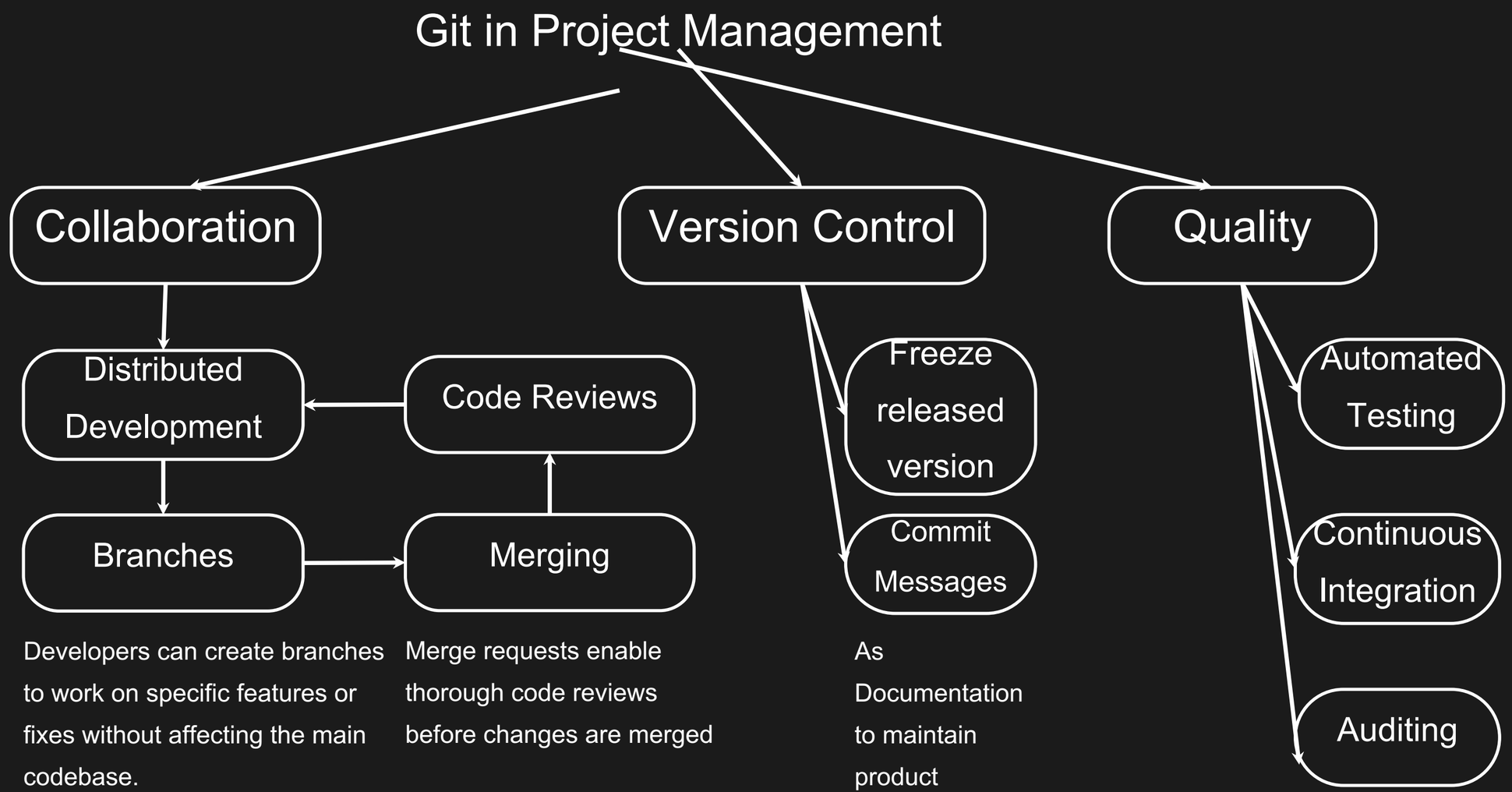
A Git Repository is a virtual storage of your project. It allows you to save versions of your code, which you can access, update, and manage over time. Repositories can be local (on your computer) or remote (on GitHub or other platforms), enabling both individual and collaborative work.

Advance Computer Programming for RAI

Git

repository

Why?



Git

Init

1. Create a new repository on GitHub named adv_compro_week04
2. Clone the repository to your local machine

```
git clone https://github.com/your-username/adv_compro_week04.git
```

3. Navigate into the repository

```
cd adv_compro_week04
```

4. Create/move some file to this directory. Ex. README.md
5. Commit change

```
git add README.md
git commit -m "Initial commit with README.md"
```

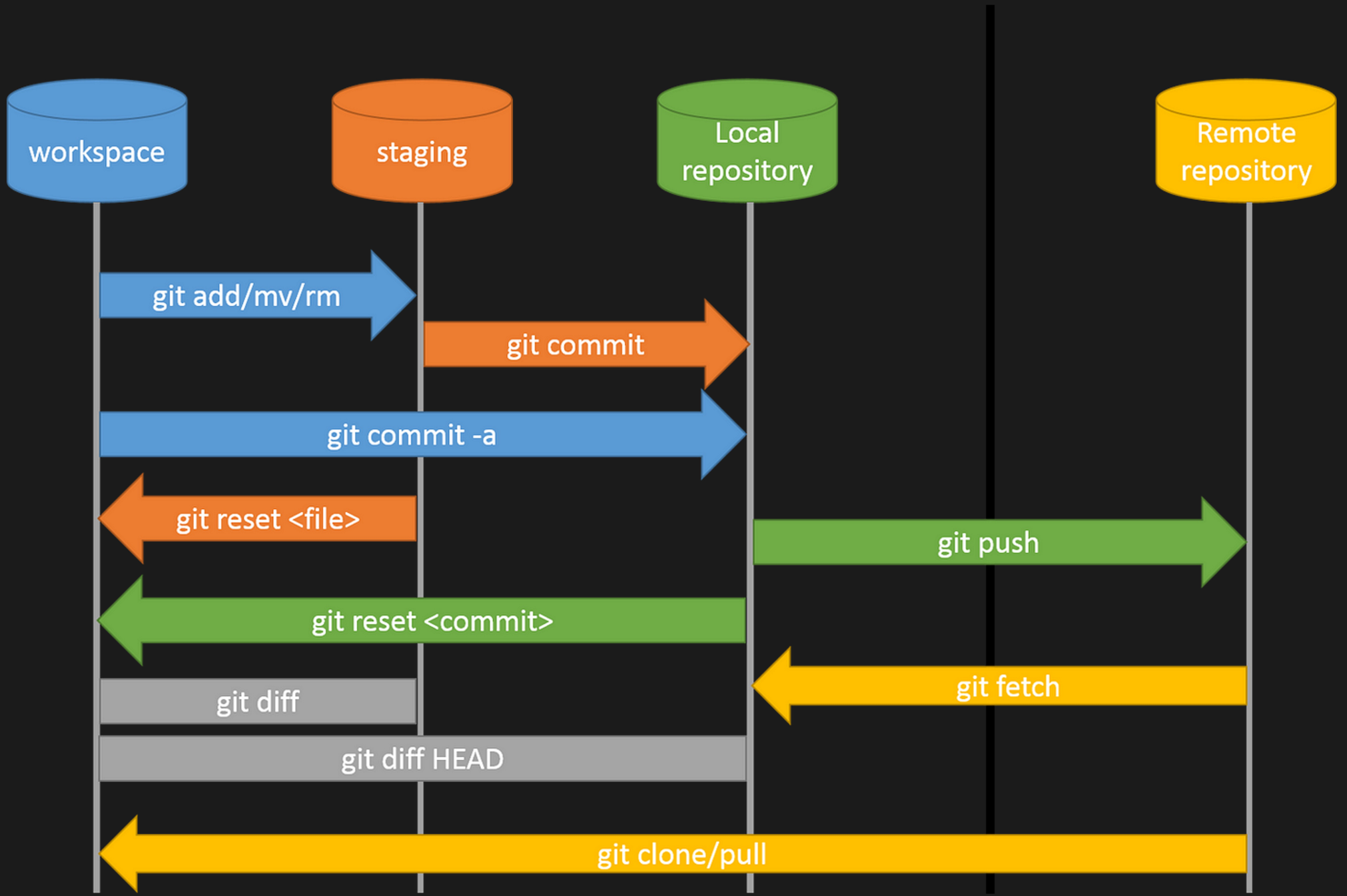
6. Push the commit to the remote repository on GitHub

```
git push origin main
```

Advance
Computer
Programming
for RAI

Git

Overview



Reference: https://github.com/Svendolin/All-about-GIT

Advance Computer Programming for RAI

Git

Commands

Command	Description	Example
git init	Initialize a new Git repository	git init
git clone <url>	Clone an existing repository	git clone https://github.com/your-repo.git
git status	Check the status	git status
git add <file>	Add a file to the staging area	git add README.md
git commit -m "message"	Commit changes in staging area	git commit -m "Initial commit"
git branch	List all branches in the repository	git branch
git branch <name>	Create a new branch	git branch feature-branch
git checkout <branch>	Switch to a different branch	git checkout feature-branch
git merge <branch>	Merge a branch into the current branch	git merge feature-branch
git push	Push changes to the remote repository	git push origin main
git pull	Fetch and merge changes from the remote repository	git pull origin main

Done!

Dr. Sarucha Yanyong

Department of Robotics and AI Engineering

School of Engineering, KMITL

EMAIL ADDRESS sarucha.ya@kmitl.ac.th

WEBSITE https://syanyong.github.io

Q/A