Day 1: UNIX COMMANDS AND VIM

Introduction to UNIX Commands and VIM

UNIX Directory Tree:
In UNIX systems, the directory tree starts from the root directory ("/") and branches out to various subdirectories. It follows a hierarchical structure, with directories containing files or additional subdirectories.

Advantages of Command Line Interface (CLI) over Graphical User Interface (GUI):

- CLI uses textual commands to interact with the operating system, providing flexibility and control.
- CLI is generally faster and uses fewer system resources compared to GUI.
- CLI allows for complex tasks and automation that can be difficult or impossible to achieve with a GUI.
- CLI is inherently more secure than GUI, making it preferred in Linux environments.

Basic UNIX Commands:

- '**pwd**' (Print working directory): Prints the path of the current working directory, starting from the root.
- **ls** – (List files): Lists files and directories in the current location.
  - Usage: `ls -[Options] [path]` (default path is the working directory).
  - Options:
    - **l**: Lists files with additional metadata.
    - **a**: Shows hidden files (files whose name begins with ".").
    - **h**: Shows file sizes in MBs and GBs instead of bytes.
    - **t**: Sorts by date modified.
    - **r**: Lists files/directories in reverse order.
  - Example: `ls *.pdf` lists all PDF files in the directory.
- **cd** (Change directory): Navigates to a different directory.
  - Usage: `cd <dirname>`
  - `<dirname>` can be an absolute or relative path.
  - To go to the home directory, use `cd` without any argument.
- **mkdir** (Make directory): Creates a new directory.
  - Usage: `mkdir < newdirname >`
  - Requires write permission in the parent directory.
- **rmdir** (Remove directory): Deletes an empty directory.
  - Usage: `rmdir <dirname>`
  - Requires write permission in the parent directory.
  - To remove an entire subtree, use `rm -r <dirname>`.
- **cat** (Concatenate): Displays the contents of a text file.
- **head** and **tail**: Show the beginning or end of a text file.
- **man** (Manual): Provides command-specific documentation.

  - Example: `man ls` displays the manual for the `ls` command.
- **history**: Displays the command history in the terminal.

VIM: A Powerful Text Editor

- VIM is a popular text editor in UNIX systems with three modes: **Normal, Insert**, and **Visual**.
- Normal mode allows movement within the file without inserting text.
- Insert mode permits text entry at the cursor's position.
- Visual mode facilitates selecting blocks of text for editing.
- Notable commands in VIM:
- Normal mode movement: h, j, k, l.
- Insert mode text insertion: i, a, o.
- Visual mode text selection: v, V, Ctrl-v.
- Redirection and Pipes: Use symbols like '<', '>', '>>', and '|' to redirect input/output and chain commands together.
- File Permissions: Control read (r), write (w), and execute (x) access with the 'chmod' command.

Code Example:

```
# Change directory to the Documents folder
cd Documents

# List all files and directories
ls -l

# Create a new directory called 'my_folder'
mkdir my_folder

# Remove a directory named 'temp'
rmdir temp

# Display the contents

 of a text file named 'example.txt'
cat example.txt

# Move to the home directory
cd

# Open VIM and edit a file named 'my_file.txt'
vim my_file.txt
```

```
# Save and exit VIM
:wq

# View the manual for the 'ls' command
man ls

# Display the command history
history
```

VIM

VIM is a modal editor that provides different modes for efficient text editing. Understanding the various modes and their associated keybindings is essential to make the most out of VIM. This documentation will cover the different modes, their keybindings, and additional features such as redirection, file permissions, and more.

Modal Editing:

1. **Normal Mode**:
   - Movement:
     - h: Move left
     - j: Move down
     - k: Move up
     - l: Move right

2. **Insert Mode**:
   - Entering Insert Mode: Press 'i'
   - Basic Insertion:
     - i: Insert before the cursor
     - a: Append after the cursor
     - I: Insert at the beginning of the line
     - A: Append at the end of the line
     - o: Open a new line below the current one
     - O: Open a new line above the current one
     - r: Replace the character under the cursor
     - R: Replace characters continuously

3. **Visual Mode**:
   - Entering Visual Mode:
     - Character-based: Press 'v'
     - Line-based: Press 'V'
     - Paragraphs: Press Ctrl-v

  - Selection and Manipulation:
    - Cut: Press 'x'
    - Paste: Press 'p'
    - Copy: Press 'y'

Redirection and Pipes:

Redirection allows you to control the input and output of commands in VIM.

- Input Redirection:
  - '<' - Reads the input from a file

- Output Redirection:
  - '>' - Redirects the output to a desired file
  - '>>' - Appends to the end of a file
  - '|' - Links programs to pass output as input to another program

File Permissions:

File permissions control access rights to files and directories.

- Read: 'r'
- Write: 'w'
- Execute: 'x'

Changing File Permissions using 'chmod':
- Add permissions for other users:
  - chmod o+rw <filename> - Adds read and write permissions for other users
- Remove permissions for other users:
  - chmod o-rw <filename> - Removes read and write permissions for other users

Python Basics

This section will cover essential concepts and functionalities of Python, including printing, data types, augmented operators, boolean operations, strings, loops, conditional statements, lists, dictionaries, sets, tuples, functions, classes, and Numpy.

Printing:
The `**print()**` function is used to display output in Python. It can be used to print variables and text.
Example:
```
print("a =", a, "b =", b, "c =", c)
```

\`\`\`

or
```
print("a = {}; b = {}; c = {}".format(a, b, c))
```

Data Types:
Python supports various data types, including:
- **int**: Represents integers
- **strings**: Represents sequences of characters
- **lists**: Ordered collections of items
- **tuples**: Ordered, immutable collections of items
- **dictionaries**: Key-value pairs
- **sets**: Unordered collections of distinct elements

Augmented Operators:
Augmented operators combine assignment and arithmetic operations.
Example:
```
a += 1  # Increment a by 1
a -= 1  # Decrement a by 1
a *= 2  # Multiply a by 2
a /= 2  # Divide a by 2
```

Boolean Operations:
Python supports `**and**` and `**or**` operators for boolean operations.

Strings:
Strings represent sequences of characters and can be enclosed in double or single quotes.
Example:
```
s = "Strings123"
```
String Operations:
- Concatenation: Strings can be concatenated using the `+` operator.
  Example: `"a" + "b" = "ab"`
- Format method: String formatting can be done using the `.format()` method.
  Example:
  ```
  a = "{} {} {}".format(a, b, c)
  print(a)  # Output: a b c
  ```

- Additional string methods: **`capitalize()`**, **`upper()`**, **`rjust(n)`**, **`center(n)`**, **`replace('s', '(ab)')`**, **`strip()`**

Loops:
Python provides two main types of loops:
- `for` loop: Used to iterate over a sequence or range of values.
Example:
```
for i in range(1, 6):
    print(i)
```
- `while` loop: Executes a block of code as long as a condition is true.
Example:
```
while i > 0:
    i -= 1
    print(i)
```

Conditional Statements:
Conditional statements are used for decision-making in Python.
Example:
```
if <condition>:
    statements
elif <condition>:
    statements
else:
    statements
```

Lists:
Lists are ordered collections of items enclosed in square brackets.
Operations:
- **`append()`**: Adds an element to the end of a list.
- **`pop()`**: Removes and returns the last element of a list.
- **Slicing**: Accessing a portion of a list using indices. `list[2:4]` slices from index 2 to 4 (exclusive).
- **`enumerate()`**: Returns indices and corresponding values stored in the list.

Dictionaries:
Dictionaries store key-value pairs and are defined using curly braces.
Operations:
- Accessing elements: `<dict[<key>]>`

- **`get(key, default)`**: Returns the value of the key if it exists in the dictionary, otherwise returns the default value.
- **`items()`**: Returns key-value pairs in the dictionary.

Sets:
Sets are unordered collections of distinct elements enclosed in curly braces.
Operations:
- Adding elements: **`<set>.add(<element>)`**
- Removing elements: **`<set>.remove(<element>)`**

Tuples:
Tuples are immutable collections of elements enclosed in parentheses

.
Example:
```
my_tuple = (1, 2, 3)
```

Functions in Python:
Functions are defined using the `def` keyword.
Example:
```
def my_function():
    statements
    return value  # Optional return statement
```

Classes in Python:
Classes are used to create objects with their own properties and functions.
Example:
```
class Animal:
    def __init__(self, species):
        self.species = species

    def sound(self):
        print(self.species, "says cock-a-doodle-doo")
```

Numpy:

Numpy is a powerful library in Python for numerical computing. It provides efficient ways to work with arrays and perform mathematical operations on them. Here's a simplified and enhanced explanation of Numpy:

Numpy Arrays:

Numpy arrays are similar to Python lists but have some key advantages. They are homogeneous, meaning they can only store elements of the same data type. Numpy arrays can be one-dimensional (like a list), but they can also have multiple dimensions, allowing you to represent matrices and multidimensional data efficiently.

Array Creation:

To create a Numpy array, you can use the `numpy.array()` function and pass a Python list or another iterable as an argument. Numpy will automatically convert the input into an array.
Example:
```python
import numpy as np

my_array = np.array([1, 2, 3, 4])
```

Array Shape and Accessing Values:

You can determine the shape or dimensions of an array using the `shape` attribute. For one-dimensional arrays, the shape is the length of the array. For multidimensional arrays, it provides the size of each dimension.
Example:
```python
print(my_array.shape)  # Output: (4,)
```

You can access elements of a Numpy array using indexing, similar to Python lists. Additionally, you can use slicing to extract sub-arrays.
Example:
```python
print(my_array[0])  # Output: 1
print(my_array[1:3])  # Output: [2, 3]
```

Array Operations:

Numpy provides a wide range of mathematical operations that can be applied element-wise to arrays. These operations are optimized for performance and allow you to perform computations efficiently.
Example:
```python
```

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

# Element-wise addition
result = x + y  # Output: [5, 7, 9]

# Element-wise multiplication
result = x * y  # Output: [4, 10, 18]

# Dot product
result = np.dot(x, y)  # Output: 32
```

Array Functions:

Numpy provides several functions to create specific types of arrays easily. For example:
- `np.ones(shape)`: Creates an array filled with ones.
- `np.zeros(shape)`: Creates an array filled with zeros.
- `np.full(shape, value)`: Creates an array filled with a specified value.
- `np.eye(size)`: Creates an identity matrix of the given size.
- `np.random.random(shape)`: Creates an array filled with random values between 0 and 1.

Array Manipulation:

Numpy offers various functions to manipulate arrays, such as reshaping, transposing, and joining arrays.
These explanations provide a simplified overview of Numpy and its key features. Numpy offers a vast array of functionalities beyond what is covered here, including advanced mathematical operations, linear algebra, statistical functions, and more. Exploring the Numpy documentation and tutorials will help you harness the full power of this versatile library.

Digital Image and OpenCV

Digital Image:

A digital image is a representation of visual information in the form of discrete pixels. It can be categorized into different types based on the number of colors or shades it contains:

1. **Binary Image**: Contains only two colors, typically black and white, representing foreground and background.

2. **Grayscale Image**: Contains shades of gray ranging from black to white. Each pixel represents a single intensity value.

3. **Color Image**: Contains color information with multiple color channels (usually Red, Green, and Blue). Each pixel represents a combination of color intensities.

Storing Images:

Images can be stored using different color formats, which determine the range of values used to represent each color channel:

1. **8-bit Color Format**: Uses 8 bits of data to represent each color channel, allowing 256 different intensity levels (0-255).

2. **16-bit Color Format**: Uses 16 bits of data to represent each color channel, allowing a much larger range of color combinations (0-65535).

Color Channels:

Color spaces define how color information is represented and interpreted. Some commonly used color spaces are:

1. **RGB Color Space**: An additive color model where different intensities of Red, Green, and Blue are combined to create various shades of color.

2. **HSV Color Space**: Represents color information in a cylindrical representation of RGB color points. It consists of three components: Hue (0-179), Saturation (0-255), and Value (0-255).

3. **CMYK Color Space**: A subtractive color model commonly used in printing, representing colors using Cyan, Magenta, Yellow, and Key (Black) channels.

Basics of OpenCV:

OpenCV (Open Source Computer Vision) is a popular open-source library for computer vision and image processing in Python. It provides various functions and tools for working with digital images. Here are some essential concepts and functions in OpenCV:

1. **Reading an Image**: Use the `cv2.imread(img, flag)` function to read an image file. The `flag` parameter specifies the color format to load the image, such as `cv2.IMREAD_COLOR` for color images, `cv2.IMREAD_GRAYSCALE` for grayscale images, or `cv2.IMREAD_UNCHANGED` to load images including the alpha channel.

2. **Displaying an Image**: Use `cv2.imshow(window_name, image)` to display an image in a window. You can use `cv2.waitKey(value)` to introduce a delay in milliseconds and `cv2.destroyAllWindows()` to close or destroy all open windows.

3. **Resizing an Image**: Use `cv2.resize(src, dsize, fx, fy, interpolation)` to resize an image. You can specify the scaling factors `fx` and `fy` or provide the target size `dsize`. The `interpolation` parameter determines the method used for interpolation.

4. **Changing Color Space**: Use `cv2.cvtColor(src, code[, dst[, dstCn]])` to convert an image from one color space to another. The `code` parameter specifies the conversion type, and `dst` is an optional output image of the same size and depth as the source image.

5. **Flipping and Rotating Images**: Use `cv2.flip(src, flipCode[, dst])` to flip an image horizontally, vertically, or both. Use `cv2.rotate(src, rotateCode[, dst])` to rotate an image by a specified angle.

6. **Image Padding**: Use `cv2.copyMakeBorder(src, top, bottom, left, right, borderType, value)` to add padding or borders to an image. You can specify the width of borders on each side and the type of border using `borderType`. The `value` parameter sets the color of the border if the border type is `cv2.BORDER_CONSTANT`.

7. **Adding Custom Designs on Images**: OpenCV provides functions to draw shapes on images, including rectangles (`cv2.rectangle()`), lines (`cv2.line()`), circles (`cv2.circle()`), ellipses (`cv2.ellipse()`), and polylines (`cv2.polylines()`). These functions allow you to specify coordinates, sizes, colors, and thickness to create desired designs.

8. **Arithmetic Operations with Images**: OpenCV provides operations for image arithmetic, such as addition (`cv2.add()`), weighted addition (`cv2.addWeighted()`), subtraction (`cv2.subtract()`), bitwise AND (`cv2.bitwise_and()`), bitwise OR (`cv2.bitwise_or()`), and bitwise NOT (`cv2.bitwise_not()`). These operations enable you to combine or modify images based on pixel values.

Image Smoothing:

Image smoothing, or blurring, is a common technique to reduce noise or unwanted details in an image. OpenCV provides functions for image blurring using different filters:

1. **Averaging:** Use `cv2.blur(img, ksize)` to apply simple averaging to an image using a kernel of specified size (`ksize`). This operation replaces each pixel's value with the average value of its neighboring pixels.

2. **Gaussian Blur**: Use `cv2.GaussianBlur(src, dst, ksize, sigmaX, sigmaY, borderType)` to apply a Gaussian blur to an image. The `ksize` parameter defines the kernel size, and `sigmaX` and `sigmaY` control the standard deviation of the Gaussian distribution along the X and Y axes, respectively.

Edge Detection Methods:

Edge detection is a fundamental technique in computer vision for identifying boundaries or edges in an image. OpenCV provides the `cv2.Canny()` function for edge detection, which applies the Canny filter. The parameters include the input image, lower and upper threshold values for hysteresis thresholding, aperture size of the Sobel filter, and a boolean parameter for more precision in calculating edge gradient.

Corner Detection:

Corner detection is used to identify and locate corners or interest points in an image. OpenCV provides the `cv2.cornerHarris()` function for corner detection, which takes the input image, destination image for storing the corner responses, block size, aperture size of the Sobel operator, a free parameter for the Harris detector, and the border type.

These are some essential concepts and functions in digital image processing and OpenCV. By leveraging these capabilities, you can perform various operations, manipulate images, and extract valuable information from visual data.

Template Matching

Template Matching is a method used to search and locate a template image within a larger image. It involves comparing a template with different positions in an image to find the best match. OpenCV provides the `cv2.matchTemplate()` function to perform template matching.

```
result = cv2.matchTemplate(image, template, method)
```

- `image`: The input image in which the template will be searched.
- `template`: The template image that will be matched against the input image.
- `method`: The method used for template matching. It can take one of the following values:
- `cv2.TM_SQDIFF`: Squared difference between the template and the image.
- `cv2.TM_SQDIFF_NORMED`: Normalized squared difference.
- `cv2.TM_CCORR`: Cross-correlation between the template and the image.
- `cv2.TM_CCOEFF`: Correlation coefficient.

To find the best match, you can use the `cv2.minMaxLoc()` function, which returns the minimum and maximum values and their corresponding locations in the result.

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
```

The `min_loc` or `max_loc` can be used as the top-left corner coordinates to draw a rectangle around the detected template region.

Contours

Contours are curves that join continuous points of the same color or intensity. They are useful for shape analysis and object detection. OpenCV provides the `cv2.findContours()` function to find contours in an image.

```
contours, hierarchy = cv2.findContours(image, mode, method)
```

- `image`: The input image from which contours will be extracted.
- `mode`: The contour retrieval mode. It can be one of the following:
- `cv2.RETR_LIST`: Retrieves all contours without establishing any hierarchy.
- `cv2.RETR_EXTERNAL`: Retrieves only the external contours.
- `cv2.RETR_CCOMP`: Retrieves all contours and organizes them into a two-level hierarchy.
- `cv2.RETR_TREE`: Retrieves all contours and reconstructs a full hierarchy of nested contours.
- ‘method`: The contour approximation method. It can be one of the following:
- `cv2.CHAIN_APPROX_NONE`: Stores all the contour points.
- `cv2.CHAIN_APPROX_SIMPLE`: Compresses horizontal, vertical, and diagonal segments int
  their endpoints.

After finding the contours, you can draw them on the image using `cv2.drawContours()`.

```
cv2.drawContours(image, contours, contour_index, color, thickness)
```

- `image`: The image on which contours will be drawn.
- `contours`: The list of contours.
- `contour_index`: Index of the contour to draw (-1 to draw all contours).
- `color`: Color of the contour.
- `thickness`: Thickness of the contour lines.

You can also extract various properties from contours, such as area, perimeter, and contour approximation using functions like `cv2.contourArea()`, `cv2.arcLength()`, and `cv2.approxPolyDP()`.

Hough Line Transformation

The Hough Line Transformation is used to detect lines in binary images. It can be applied after performing edge detection on the image. OpenCV provides the `cv2.HoughLines()` function for this purpose.

```
lines = cv2.HoughLines(image, rho, theta, threshold)
```

- `image`: The input binary image containing edges.
- `rho`: The distance resolution in pixels.
- `theta`: The angular resolution in radians.
- `threshold

`: The minimum number of votes required to consider a line.

The function returns an array of lines, where each line is represented by ρ and θ values.

Video Processing

Video processing involves performing various operations on a sequence of frames to analyze or manipulate a video. You can apply the same image processing techniques to each frame of a video.

To capture a video, you can use the `cv2.VideoCapture()` function.

```
cap = cv2.VideoCapture(filename)
```

- `filename`: The name of the video file or device index.

To read frames from the video, you can use the `cap.read()` function.

```
ret, frame = cap.read()
```

- `ret`: Boolean value indicating if a frame was successfully read.
- `frame`: The captured frame.

To save a video, you can use the `cv2.VideoWriter()` function.

```
```

```
out = cv2.VideoWriter(filename, fourcc, fps, frameSize)
```

- `filename`: The name of the output video file.
- `fourcc`: Four-character code representing the video codec.
- `fps`: The frame rate of the output video.
- `frameSize`: The size of each frame.

You can then use the `out.write()` function to write frames to the output video.

```
out.write(frame)
```

Remember to release the video capture and writer objects after you finish processing the video.

```
cap.release()
out.release()
```

This covers the basic concepts and functions related to template matching, contours, Hough line transformation, and video processing in OpenCV.

Day 5: Path Planning Algorithms

Graphs

A graph is a mathematical structure that consists of a set of vertices (nodes) and a set of edges (connections) that link pairs of vertices together.

Breadth First Search (BFS)

BFS is an algorithm used for traversing or searching graph data structures. It starts at the tree root (or any arbitrary node) and explores all the nodes at the present depth before moving on to nodes at the next depth level. BFS uses a queue data structure for implementation.

Pseudocode:

```
function BFS(graph, start):
    create an empty queue
    enqueue start node into the queue
    mark start node as visited
    while queue is not empty:
        dequeue a node from the queue
        process the node

        for each neighbor of the node:
            if neighbor is not visited:
                mark neighbor as visited
                enqueue neighbor into the queue
```

Depth First Search (DFS)

DFS is another algorithm for traversing or searching graph data structures. It starts at the root node and explores as far as possible along each branch before backtracking.

Pseudocode:

```
function DFS(graph, start):
    create an empty stack
    push start node into the stack
    mark start node as visited

    while stack is not empty:
        pop a node from the stack
        process the node

        for each neighbor of the node:
            if neighbor is not visited:
                mark neighbor as visited
                push neighbor into the stack
```

Dijkstra's Algorithm

Dijkstra's algorithm is used to find the shortest paths between nodes in a weighted graph. It maintains a priority queue (often implemented using a min-heap) to select the node with the smallest known distance at each step.

Pseudocode:

```
function Dijkstra(graph, start):
    create a set of unvisited nodes
    set the distance of all nodes to infinity
    set the distance of the start node to 0

    while there are unvisited nodes:
        select the node with the smallest distance
        mark the node as visited

        for each neighbor of the node:
            calculate the distance from the start node to the neighbor
            if the calculated distance is smaller than the current distance:
                update the distance of the neighbor
                set the previous node of the neighbor to the current node
```

A* Algorithm

A* algorithm is an informed search algorithm that uses heuristic functions to find the shortest path from a start node to a goal node. It combines the advantages of both BFS and Dijkstra's algorithm by considering the estimated cost from the current node to the goal node.

Pseudocode:
```
function AStar(graph, start, goal):
    create open and closed lists
    add the start node to the open list

    while the open list is not empty:
        select the node with the lowest f value from the open list
        move the selected node to the closed list

        if the selected node is the goal node:
            return the path

        for each neighbor of the selected node:
            calculate the g value (cost from start to neighbor)
            calculate the h value (heuristic estimated cost from neighbor to goal)
            calculate the f value (sum of g and h)

            if the neighbor is not in the open or closed list:
                add the neighbor to the open list
            else if the neighbor is in the open list with a higher f value:
                update the neighbor's f value and parent
```

RRT Algorithm

Rapidly-Exploring Random Trees (RRT) is a probabilistic algorithm used for path planning in robotics. It generates a tree by randomly sampling points in the configuration space and expanding the tree towards those points.

RRT* Algorithm

RRT* is an extension of the RRT algorithm that improves the optimality of the generated path. It performs rewiring of the tree to find shorter paths by considering neighboring nodes.

This concludes the coverage of path planning algorithms. In the next section, we will explore various image processing techniques.

Day 6: Image Processing Techniques

Fourier Transform

Fourier Transform is a mathematical technique that transforms a time-domain signal into its frequency domain representation. It decomposes a signal into its constituent frequencies, allowing analysis and manipulation of the signal in the frequency domain.

Geometric Optics

Geometric Optics is a branch of optics that studies the behavior of light rays by considering them as rays rather than waves. It provides a simplified model for understanding the propagation of light in various optical systems.

Wavelet Transform and Ancillaries

Wavelet Transform is a mathematical technique used to decompose a signal into a set of wavelet functions. It offers advantages over the Fourier Transform, such as capturing localized information in both time and frequency domains.

Image Stitching

Image Stitching is the process of combining multiple overlapping images to create a single panoramic image. It involves finding corresponding points between images and warping them to align properly.

Stereovision

Stereovision is a technique that utilizes multiple cameras to extract 3D information from a scene. By analyzing the disparities between corresponding points in different camera views, depth information can be estimated.

Coordinate Stitching

Coordinate Stitching involves aligning and merging data from multiple coordinate systems or frames of reference. It is often used in robotics or mapping applications to combine data from different sensors or coordinate systems.

Orthorectification

Orthorectification is the process of removing distortions caused by terrain relief or sensor geometry from aerial or satellite images. It corrects the image to represent the Earth's surface as if it were viewed from directly above.

Crater Detection

Crater Detection is an image processing technique used in planetary science to identify and analyze impact craters on planetary surfaces. It involves image segmentation, feature extraction, and classification algorithms to identify crater candidates.