SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2024-25

BCA 2nd Sem

Web Technology (UCA24M01J)

Lab Manual

Lab 1: Learning to Work with Linux Server

Title: Introduction to Linux Server

Aim: To familiarize students with the Linux server environment, basic commands, and file system navigation.

Procedure:

1. Log in to the Linux server.
2. Practice basic commands: ls, cd, pwd, mkdir, rmdir, touch, cp, mv, rm.
3. Explore the file system structure.
4. Use the man command to learn about command options.

Source Code: (Commands used in the lab)

```
# Example commands:
ls -l
cd /home/user
pwd
mkdir my_directory
touch my_file.txt
cp my_file.txt my_file_copy.txt
mv my_file.txt new_file.txt
rm new_file.txt
rmdir my_directory
```

Input: (Commands entered by the user)

```
# Example input:
cd /var/www/html
mkdir test_site
cd test_site
touch index.html
```

Expected Output: (What the server should return)

```
# Example output:
# ls -l  (lists files and directories with permissions)
# pwd     (/var/www/html)
# (Successful creation of directory and file)
```

Lab 2: Working with Files and Directory Commands

Title: File and Directory Management in Linux

Aim: To gain proficiency in using Linux commands for managing files and directories.

Procedure:

1. Create, copy, move, and delete files and directories using commands like cp, mv, rm, mkdir, and rmdir.
2. Use wildcards (*, ?) to manipulate multiple files.
3. Explore file permissions using chmod and chown.
4. Use find command to locate files.

Source Code:

```
# Example commands
mkdir dir1 dir2
touch file1.txt file2.txt file3.log
cp file*.txt dir1/
mv file2.txt dir2/newfile2.txt
rm file3.log
rmdir dir1
chmod 755 script.sh
chown user:group myfile.txt
find / -name "my*"
```

Input:

```
# Example input
mkdir documents
cd documents
touch report.txt, presentation.ppt, image.jpg
cp *.txt backup/
```

Expected Output:

```
# Example output
# (Directories and files created successfully)
# (Files copied to backup directory)
# (Permissions changed)
# (Files found by find command)
```

Lab 3: Working with File Commands, Creating and Modifying Files Using VI Editor

Title: File Manipulation and VI Editor

Aim: To learn how to create and modify files using the vi editor and other file-related commands.

Procedure:

1. Create new files using touch or redirecting output (>).
2. Modify existing files using the vi editor: learn basic commands for inserting, deleting, and saving text.
3. Use commands like cat, more, and less to view file contents.
4. Use echo to add content to a file.

Source Code:

```
# Example commands and vi usage
touch newfile.txt
vi newfile.txt  # (Demonstrate basic vi commands)
cat newfile.txt
more longfile.txt
less longfile.txt
echo "Hello, world!" >  newfile.txt
echo "Additional line" >> newfile.txt
```

Input:

```
# Example input
vi myfile.txt
# (Inside vi:  i <some text> Esc :wq)
```

Expected Output:

```
# Example output
# (File created)
# (File content displayed by cat, more, less)
# (File modified by echo)
```

Lab 4: Writing Simple PHP Programs

Title: Introduction to PHP Programming

Aim: To write and execute basic PHP scripts.

Procedure:

1. Set up a PHP development environment (e.g., using XAMPP, WAMP, or a Linux server with PHP).
2. Create a simple PHP file (e.g., info.php).
3. Write PHP code to display output using echo or print.
4. Use phpinfo() to display server information.
5. Run the PHP script from a web browser or command line.

Source Code:

```
// Example PHP code (info.php)
<?php
echo "Hello, PHP World!";
echo "<br>";
phpinfo();
?>
```

Input: (Accessing the PHP file through a web browser)

```
# Example input (URL in browser)
http://localhost/info.php
```

Expected Output:

```
# Example output
# (Web page displaying "Hello, PHP World!" and PHP configuration information)
```

Lab 5: Operators & Control Statements

Title: PHP Operators and Control Flow

Aim: To implement PHP programs using various operators and control statements.

Procedure:

1. Write PHP scripts using arithmetic, comparison, logical, and assignment operators.
2. Implement conditional statements: if, else, elseif, switch.
3. Use looping structures: for, while, do-while, foreach.

Source Code:

```
// Example PHP code
<?php
$x = 10;
$y = 5;
echo "Sum: " . ($x + $y) . "<br>";
if ($x > $y) {
  echo "$x is greater than $y <br>";
} else {
  echo "$y is greater than or equal to $x <br>";
}

for ($i = 0; $i < 5; $i++) {
  echo "Iteration: $i <br>";
}

$colors = array("red", "green", "blue");
foreach ($colors as $value) {
  echo "Color: $value <br>";
}
?>
```

Input: (Accessing the PHP file through a web browser)

```
 # Example input (URL in browser)
http://localhost/operators.php
```

Expected Output:

```
# Example output
#  Sum: 15
#  10 is greater than 5
#  Iteration: 0
#  Iteration: 1
#  Iteration: 2
#  Iteration: 3
#  Iteration: 4
#  Color: red
#  Color: green
#  Color: blue
```

Lab 6: Embedding PHP Script in HTML

Title: Embedding PHP in HTML

Aim: To integrate PHP code within HTML pages to create dynamic content.

Procedure:

1. Create an HTML file.
2. Embed PHP code within the HTML file using <?php ... ?> tags.
3. Use PHP to generate dynamic HTML content (e.g., displaying the current date/time, retrieving data from a form).

Source Code:

```
// Example HTML with embedded PHP
<!DOCTYPE html>
<html>
<head>
  <title>Dynamic Web Page</title>
</head>
<body>
  <h1>Welcome!</h1>
  <p>The current time is: <?php echo date("Y-m-d H:i:s"); ?></p>
  <form method="post">
    <input type="text" name="username" placeholder="Enter your name">
    <input type="submit" value="Greet">
  </form>
  <?php
  if (isset($_POST['username'])) {
    $name = $_POST['username'];
    echo "<h2>Hello, $name!</h2>";
  }
  ?>
</body>
</html>
```

Input: (Accessing the HTML file through a web browser and submitting the form)

```
# Example input
# 1. Open http://localhost/mypage.html
# 2. Enter "John" in the form and submit.
```

Expected Output:

```
 # Example output
# (Web page displaying the current date/time)
# (After submitting the form with "John", the page displays "Hello, John!")
```

Lab 7: Passing Parameters to a Function

Title: PHP Functions with Parameters

Aim: To define and call PHP functions, passing parameters to them.

Procedure:

1. Define PHP functions that accept arguments.
2. Call the functions with different parameter values.
3. Use type hinting (if applicable) to enforce data types.
4. Explore passing parameters by value and by reference.

Source Code:

```
// Example PHP code
<?php
function greet($name, $message = "Hello") {
  echo "$message, $name!<br>";
}

greet("Alice");
greet("Bob", "Good morning");

function add(&$num) {
  $num += 10;
}
$my_num = 20;
add($my_num);
echo "Number: $my_num"; // Output: Number: 30
?>
```

Input: (Accessing the PHP file through a web browser)

Expected Output:

```
# Example output
# Hello, Alice!
# Good morning, Bob!
# Number: 30
```

Lab 8: Functions & Strings

Title: PHP Functions and String Manipulation

Aim: To work with PHP functions and perform operations on strings.

Procedure:

1. Use built-in PHP string functions (e.g., strlen(), strtoupper(), strtolower(), strpos(), substr()).
2. Create custom functions to manipulate strings.
3. Combine functions to achieve complex string operations.

Source Code:

```
// Example PHP code
<?php
$text = "This is a string.";
echo "Length: " . strlen($text) . "<br>";
echo "Uppercase: " . strtoupper($text) . "<br>";
echo "Position of 'is': " . strpos($text, "is") . "<br>";
echo "Substring: " . substr($text, 5, 2) . "<br>";

function reverse_string($str) {
  return strrev($str);
}
echo "Reversed: " . reverse_string($text);
?>
```

Input: (Accessing the PHP file through a web browser)

Expected Output:

```
# Example output
# Length: 17
# Uppercase: THIS IS A STRING.
# Position of 'is': 2
# Substring: is
# Reversed: .gnirts a si sihT
```

Lab 9: String Manipulation

Title: Advanced String Manipulation in PHP

Aim: To perform more complex string operations using PHP.

Procedure:

1. Use functions like str_replace(), trim(), explode(), and implode().
2. Work with regular expressions for pattern matching and replacement (using preg_match(), preg_replace()).
3. Handle character encoding issues.

Source Code:

```
<?php
$text = "  Hello, world!  ";
echo "Trimmed: '" . trim($text) . "'<br>";
echo "Replaced: " . str_replace("world", "PHP", $text) . "<br>";

$tags = "apple,banana,orange";
$fruits = explode(",", $tags);
print_r($fruits);  // Output: Array ( [0] => apple [1] => banana [2] =>
orange )
echo "<br>";

$numbers = array(1, 2, 3);
$list = implode("-", $numbers);
echo "List: " . $list . "<br>"; // Output: List: 1-2-3

$email = "test@example.com";
if (preg_match("/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/", $email))
{
  echo "Valid email address.";
} else {
  echo "Invalid email address.";
}
?>
```

Input:

Expected Output:

```
# Example output
# Trimmed: 'Hello, world!'
# Replaced:   Hello, PHP!
# Array ( [0] => apple [1] => banana [2] => orange )
# List: 1-2-3
# Valid email address.
```

Lab 10: Arrays

Title: Working with Arrays in PHP

Aim: To create, manipulate, and iterate through arrays in PHP.

Procedure:

1. Create indexed and associative arrays.
2. Access and modify array elements.
3. Use array functions: count(), sort(), asort(), ksort(), array_push(), array_pop(), array_merge().
4. Iterate through arrays using for and foreach loops.

Source Code:

```
<?php
$colors = array("red", "green", "blue");
echo "First color: " . $colors[0] . "<br>";
$colors[] = "yellow";  // Add to the end
print_r($colors); echo "<br>";

$ages = array("Peter" => 32, "John" => 28, "Mary" => 35);
echo "John's age: " . $ages["John"] . "<br>";
$ages["Mary"] = 36;
print_r($ages); echo "<br>";

echo "Number of colors: " . count($colors) . "<br>";
sort($colors);  // Sort alphabetically
print_r($colors); echo "<br>";

foreach ($ages as $name => $age) {
  echo "$name is $age years old <br>";
}
?>
```

Input:

Expected Output:

```
# Example output
# First color: red
# Array ( [0] => red [1] => green [2] => blue [3] => yellow )
# John's age: 28
# Array ( [Peter] => 32 [John] => 28 [Mary] => 36 )
# Number of colors: 4
# Array ( [0] => blue [1] => green [2] => red [3] => yellow )
# Peter is 32 years old
# John is 28 years old
# Mary is 36 years old
```

Lab 11: Arrays & Objects

Title: Arrays and Objects in PHP

Aim: To work with arrays in conjunction with objects in PHP.

Procedure:

1. Create classes and objects.
2. Store objects in arrays.
3. Access object properties within arrays.
4. Use foreach loops to iterate through arrays of objects.

Source Code:

```
<?php
class Person {
  public $name;
  public $age;
  public function __construct($name, $age) {
    $this->name = $name;
    $this->age = $age;
  }
  public function getInfo() {
    return "Name: " . $this->name . ", Age: " . $this->age;
  }
}

$people = array();
$people[] = new Person("Alice", 30);
$people[] = new Person("Bob", 40);
$people[] = new Person("Charlie", 25);

foreach ($people as $person) {
  echo $person->getInfo() . "<br>";
}

?>
```

Input:

Expected Output:

```
# Example output
# Name: Alice, Age: 30
# Name: Bob, Age: 40
# Name: Charlie, Age: 25
```

Lab 12: Introspection and Serialization

Title: PHP Introspection and Serialization

Aim: To understand how to inspect PHP variables and objects, and how to serialize data.

Procedure:

1. Use functions like var_dump(), gettype(), is_array(), is_object().
2. Use get_class_methods(), get_class_vars() to inspect classes.
3. Serialize and unserialize data using serialize() and unserialize().

Source Code:

```
<?php
class MyClass {
  public $prop1 = "hello";
  private $prop2 = "world";
  public function myMethod() {
    echo "Method called";
  }
}

$obj = new MyClass();
var_dump($obj); echo "<br>";
echo gettype($obj) . "<br>";
echo "Is array? " . is_array($obj) . "<br>";
echo "Is object? " . is_object($obj) . "<br>";

print_r(get_class_methods("MyClass")); echo "<br>";
print_r(get_class_vars("MyClass")); echo "<br>";

$serialized_data = serialize($obj);
echo "Serialized: " . $serialized_data . "<br>";
$unserialized_data = unserialize($serialized_data);
var_dump($unserialized_data);
?>
```

Input:

Expected Output:

```
# Example output
# object(MyClass)#1 (2) { ["prop1"]=> string(5) "hello"
["prop2":"MyClass":private]=> string(5) "world" }
# object
# Is array?
# Is object? 1
# Array ( [myMethod] => myMethod )
# Array ( [prop1] => hello [prop2] => world )
# Serialized:
O:7:"MyClass":2:{s:5:"prop1";s:5:"hello";s:10:"MyClassprop2";s:5:"world";}
# object(MyClass)#2 (2) { ["prop1"]=> string(5) "hello"
["prop2":"MyClass":private]=> string(5) "world" }
```

Lab 13: Creating Database and Table

Title: Creating Database and Table in MySQL

Aim: To create a database and table using MySQL.

Procedure:

1. Connect to the MySQL server using a client (e.g., mysql command-line client, phpMyAdmin).
2. Create a new database using the CREATE DATABASE statement.
3. Select the database using the USE statement.
4. Create a table with appropriate columns and data types using the CREATE TABLE statement.
5. Define a primary key for the table.

Source Code:

```
-- Example SQL commands
CREATE DATABASE my_database;
USE my_database;
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
SHOW DATABASES;
SHOW TABLES;
DESCRIBE users;
```

Input: (SQL commands executed in MySQL client)

```
# Example input
# (The SQL commands above)
```

Expected Output: (Output from MySQL client)

```
# Example output
# (Database created successfully)
# (Table created successfully)
# (Output of SHOW DATABASES, SHOW TABLES, DESCRIBE users)
```

Lab 14: Working with Various MySQL Queries

Title: MySQL Queries

Aim: To write and execute various SQL queries to interact with a database.

Procedure:

1. Insert data into a table using the INSERT statement.
2. Retrieve data from a table using the SELECT statement with WHERE clause, ORDER BY, and LIMIT.
3. Update data in a table using the UPDATE statement.
4. Delete data from a table using the DELETE statement.
5. Use aggregate functions (e.g., COUNT(), AVG(), MAX(), MIN(), SUM()).
6. Perform joins between tables.

Source Code:

```
-- Example SQL queries
USE my_database;
INSERT INTO users (username, email, password) VALUES
('john_doe', 'john.doe@example.com', 'password123'),
('jane_smith', 'jane.smith@example.com', 'password456');

SELECT * FROM users;
SELECT username, email FROM users WHERE username = 'john_doe';
SELECT * FROM users ORDER BY created_at DESC;
SELECT * FROM users LIMIT 1;

UPDATE users SET email = 'new_email@example.com' WHERE username = 'john_doe';
DELETE FROM users WHERE username = 'jane_smith';

SELECT COUNT(*) FROM users;
SELECT AVG(age) from profiles;
```

Input: (SQL queries executed in MySQL client)

Expected Output: (Output from MySQL client)

```
# Example output
# (Data inserted successfully)
# (Query results)
# (Data updated successfully)
# (Data deleted successfully)
# (Count of records)
```

Lab 15: Developing Simple Database Applications

Title: Simple Database Application with PHP and MySQL

Aim: To develop a basic web application that interacts with a MySQL database.

Procedure:

1. Create a database and table (as in previous labs).
2. Write PHP scripts to connect to the database using mysqli or PDO.
3. Implement functionality to:

   Display data from the table.

   Insert new data into the table (e.g., through a form).

   Update existing data.

   Delete data.

4. Create HTML forms for user input.

Source Code: (This will involve multiple PHP files and SQL)

```
// Example PHP code (connect.php)
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "my_database";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
?>

// Example PHP code (index.php - display data)
<?php
include 'connect.php';
$sql = "SELECT id, username, email FROM users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  while($row = $result->fetch_assoc()) {
    echo "ID: " . $row["id"]. " - Name: " . $row["username"]. " - Email: " .
$row["email"]. "<br>";
  }
} else {
  echo "No users found";
}
$conn->close();
?>

// Example PHP code (add_user.php - insert data)
<?php
include 'connect.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $username = $_POST['username'];
  $email = $_POST['email'];
  $password = password_hash($_POST['password'], PASSWORD_DEFAULT); // Hash
the password

  $sql = "INSERT INTO users (username, email, password) VALUES ('$username',
'$email', '$password')";

  if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
  } else {
    echo "Error: " . $sql . "<br>" . $conn->error;
  }
}
$conn->close();
?>

// Example HTML (add_user_form.html)
<form action="add_user.php" method="post">
  Username: <input type="text" name="username"><br>
  Email: <input type="text" name="email"><br>
  Password: <input type="password" name="password"><br>
  <input type="submit" value="Add User">
</form>
```

Input: (Data entered through HTML forms)

Expected Output: (Web pages displaying data and success/error messages)

```
# Example output
# (List of users from the database)
# (Success message after adding a user)
      # (Error message if there's a problem)
```