Project Explanation (OS)

This script is a comprehensive shell script for managing orders, customer interactions, and administrative tasks for a fictional restaurant, "Rahman Fast Food Restaurant." Here's a line-by-line breakdown:

Welcome Message and Menu Display

```
echo "_____________________________Welcome TO RAHMAN FAST FOOD RESTAURANT
_____________________________"
echo
"_____________________________________________________________________________________
_____________________"
echo "_____________________________________________MENU
ITEM_____________________________________________"
echo " Item No_________FOOD
Name_________________Size_____________________________Price_____"
echo "   1          Chicken pot pie           6/8/10/12 inches
150/170/200/250 Tk"
...
```

- **Purpose**: Displays a welcome message and the menu with item numbers, sizes, and prices for customer reference.

Order Confirmation

```
echo "Do you want to order (yes=1/no=2): "
read user_input
```

- **Purpose**: Prompts the user to start an order by entering `1` for yes or `2` for no.
- **`read`**: Captures user input and stores it in the `user_input` variable.

Customer Details

```
if [ "$user_input" -eq 1 ]; then
    read -p "Please Enter your name: " name
    read -p "Please Enter your address: " address
    read -p "Please Enter your phone number: " phone_number
    read -p "Please Enter your email: " email
```

- **Purpose**: Collects customer details like name, address, phone number, and email if the user decides to place an order.

Item Selection

```
echo "Please enter item number:"
read item_no
case $item_no in
    1) item_name="Chicken pot pie";;
    ...
    *) echo "Invalid item selected"; exit 1;;
esac
```

- **Purpose**: Prompts the user to choose an item by entering its number, assigns the corresponding item name, and exits if the input is invalid.

Size and Price Selection

```
echo "Please choose the size:"
read size_choice
case $item_no in
    1)
        case $size_choice in
            1) price=150;;
            ...
            *) echo "Invalid size"; exit 1;;
        esac
        ;;
    ...
esac
```

- **Purpose**: Prompts the user to select a size and assigns the corresponding price based on the menu.

Inventory Check

```
declare -A inventory
inventory=( [1]=50 [2]=100 [3]=70 [4]=60 [5]=80 [6]=200 [7]=150 [8]=30
[9]=100 [10]=90 )
if [ "${inventory[$item_no]}" -ge "$quantity" ]; then
    inventory[$item_no]=$((inventory[$item_no] - quantity))
    echo "Inventory updated. Remaining ${item_name}s: ${inventory[$item_no]}"
else
    echo "Sorry, insufficient inventory for $item_name. Only
${inventory[$item_no]} left."
    exit 1
fi
```

- **Purpose**: Checks if sufficient stock is available for the chosen item and updates inventory if possible. Exits if inventory is insufficient.

Total Price and Discounts

```
total_price=$((price * quantity))
if (( total_price > 1000 )); then
    discount=$(( total_price / 10 ))
    total_price=$(( total_price - discount ))
    echo "10% discount applied! Total price: $total_price Tk"
fi
```

- **Purpose**: Calculates the total price and applies a 10% discount if the total exceeds 1000 Tk.

Gift Wrapping

```
read -p "Do you want gift wrapping? (yes=1/no=2): " gift_choice
if [ "$gift_choice" -eq 1 ]; then
    read -p "Enter occasion (e.g., Birthday, Anniversary): " occasion
    wrapping_charge=50
    total_price=$((total_price + wrapping_charge))
    echo "Gift wrapping charge added. Updated total price: $total_price Tk"
fi
```

- **Purpose**: Offers gift wrapping for an additional charge and updates the total price if selected.

Customization Options

```
read -p "Do you want to customize your order? (yes=1/no=2): " custom_choice
if [ "$custom_choice" -eq 1 ]; then
    echo "Available Customizations: 1=Extra Toppings (50 Tk), 2=Less Salt,
3=Extra Spicy, 4=Add Cheese (30 Tk)"
    read -p "Choose your customization: " custom_option
    case $custom_option in
        1) total_price=$((total_price + 50)); echo "Added Extra Toppings (50
Tk)";;
        ...
    esac
fi
```

- **Purpose**: Allows customers to customize their order with additional charges or preferences.

Payment Method

```
echo "Please choose a payment method: 1=Cash, 2=Card, 3=Online Payment"
read payment_method
case $payment_method in
    1) echo "You chose cash payment."; payment_mode="Cash";;
    ...
esac
```

- **Purpose**: Asks the customer to choose a payment method and saves their choice.

Saving Order and Receipt Generation

```
echo
"$name,____$address,____$phone_number,____$email,____$item_name,____$total_pr
ice" >> restaurant.txt
pandoc receipt.txt -o receipt.pdf
```

- **Purpose**: Saves the order details to `restaurant.txt` and generates a receipt PDF using `pandoc`.

Admin Panel

```
read -sp "Enter admin password to access admin panel: " admin_password
if [ "$admin_password" == "admin123" ]; then
    echo "Welcome, Admin!"
    echo "1. View all orders"
    ...
```

- **Purpose**: Provides admin functionalities like viewing orders, feedback, total sales, employee management, and feedback analysis.

This code is a robust example of combining shell scripting with conditional logic, file operations, and user interaction. Let me know if you'd like detailed explanations for specific sections!

Or,

Certainly, let's break down the given Bash script line by line:

1. Welcome Message and Menu:

- `#!/bin/bash`: Specifies that this script should be executed by the Bash shell.
- `echo ...`: Displays welcome messages and the menu of available food items with their prices and sizes.

2. User Input:

- `read user_input`: Prompts the user to enter 1 (yes) to order or 2 (no) to exit.

3. Order Processing (if user_input is 1):

- **Customer Details:**
  - `read -p ...`: Prompts the user to enter their name, address, phone number, and email.
- **Item Selection:**
  - `read item_no`: Prompts the user to enter the item number from the menu.
  - `case $item_no in ...`: Assigns the corresponding item name to the `item_name` variable based on the entered `item_no`.
- **Size Selection:**
  - `read size_choice`: Prompts the user to choose the size of the selected item.
  - `case $item_no in ...`: Assigns the corresponding price to the `price` variable based on the selected `item_no` and `size_choice`.
- **Quantity:**
  - `read -p ...`: Prompts the user to enter the quantity of the selected item.
- **Inventory Check:**
  - `declare -A inventory`: Declares an associative array to store the inventory of each item.
  - Checks if the requested quantity is available in the inventory. If not, displays an error and exits.
  - Updates the inventory with the remaining quantity.

- **Price Calculation and Discounts:**
  - Calculates the `total_price` by multiplying the `price` with the `quantity`.
  - Applies a 10% discount if the `total_price` exceeds 1000 Tk.
- **Gift Wrapping:**
  - Asks the user if they want gift wrapping.
  - If yes, adds a wrapping charge to the `total_price`.
- **Customization:**
  - Asks the user if they want to customize their order.
  - If yes, allows the user to select customization options and adjusts the `total_price` accordingly.
- **Payment Method:**
  - Asks the user to choose a payment method (Cash, Card, Online).
  - Stores the chosen `payment_mode` in a variable.
- **Save Order:**
  - Appends the order details (name, address, phone number, email, item_name, total_price) to a file named `restaurant.txt`.
- **Feedback:**
  - Asks the user for feedback.
  - If provided, appends the feedback to a file named `feedback.txt`.
- **Receipt Generation:**
  - Creates a temporary file (`receipt.txt`) with the order details.
  - Uses `pandoc` to convert the `receipt.txt` to a PDF file.
- **Order Cancellation:**
  - Asks the user if they want to cancel the last order.
  - If yes, saves the last order to `cancelled_order.txt` and removes it from `restaurant.txt`.
- **Table Reservation:**
  - Asks the user if they want to reserve a table.
  - If yes, collects reservation details and appends them to `reservations.txt`.

- **Order Status Tracking:**
    - Simulates order preparation and delivery with `sleep` commands.
- **Estimated Delivery Time:**
    - Displays an estimated delivery time.
- **Order History:**
    - Asks the user to enter their phone number or email to retrieve their order history.

4. Admin Panel:

- Prompts the admin to enter a password.
- If the password is correct:
    - Displays a menu of admin options:
        - View all orders
        - View feedback
        - View total sales
        - Employee Management
        - Feedback Analysis
    - Executes the selected admin option.
- If the password is incorrect:
    - Displays an error message.

5. Exit:

- If the user chooses not to order (`user_input` is 2), the script exits with a thank-you message.

Key Improvements:

- **Error Handling:** The script includes basic error handling for invalid inputs, insufficient inventory, and invalid admin passwords.
- **Inventory Management:** Implements a simple inventory system to track the availability of items.
- **Discounts:** Applies a discount for orders exceeding a certain amount.

- **Customization Options:** Allows users to customize their orders.
- **Gift Wrapping:** Provides the option for gift wrapping with an additional charge.
- **Payment Methods:** Offers multiple payment options.
- **Receipt Generation:** Generates a PDF receipt for each order.
- **Order Cancellation:** Allows users to cancel their last order.
- **Table Reservation:** Enables users to reserve tables.
- **Order Status Tracking:** Simulates order preparation and delivery.
- **Order History:** Allows users to view their order history.
- **Admin Panel:** Provides basic admin functionalities for viewing orders, feedback, sales, managing employees, and analyzing feedback.

This script provides a basic framework for a restaurant order management system. It can be further enhanced by adding features like order confirmation emails, delivery tracking, loyalty programs, and more sophisticated inventory management.

Or,(Main)

Let's dive deeper into **specific parts of the script**. Below is a detailed breakdown of key sections, with further explanation for each concept and functionality.

Welcome Message

```
echo "_____Welcome TO RAHMAN FAST FOOD RESTAURANT
_____"
```

- This line uses the `echo` command to display a welcome message.
- It uses underscores (_) and other characters to format the text visually, making the welcome message stand out.

Menu Display

```
echo " Item No_____FOOD
Name_____Size_____Price_____"
echo "   1          Chicken pot pie          6/8/10/12 inches
150/170/200/250 Tk"
```

- Displays a formatted list of menu items with details like **Item Number**, **Food Name**, **Available Sizes**, and **Prices**.
- Each row uses spacing or underscores to visually separate data points.

Order Decision

```
echo "Do you want to order (yes=1/no=2): "
read user_input
```

- Asks the user whether they want to place an order.
- The `read` command stores the user's input into the variable `user_input`. Here:
  - `1` represents "yes" (proceed with order).
  - `2` represents "no" (exit the program).

Customer Information Collection

```
read -p "Please Enter your name: " name
```

- The `-p` flag allows `read` to display a prompt message alongside collecting input.
- Stores the entered name in the `name` variable. The same logic applies for `address`, `phone_number`, and `email`.

Menu Item Selection

```
read item_no
case $item_no in
    1) item_name="Chicken pot pie";;
    ...
    10) item_name="Bread sticks";;
    *) echo "Invalid item selected"; exit 1;;
esac
```

1. **Purpose**: Prompts the user to select a menu item.
2. **How It Works**:
    - `case` matches `item_no` against possible valid options (`1` to `10`).
    - Based on the match, assigns a corresponding `item_name`.
    - The `*)` part serves as the "default" case, displaying an error if the input doesn't match any valid option.
3. **Why `exit 1`?** Exits the script with status `1`, indicating an error occurred.

Size and Price Selection

```
case $item_no in
    1)
        case $size_choice in
            1) price=150;;
            2) price=170;;
            3) price=200;;
            4) price=250;;
            *) echo "Invalid size"; exit 1;;
        esac
        ;;
    # Add similar cases for other items
esac
```

1. **Purpose**: Assigns a price based on the selected size.
2. **How It Works**:
    - Outer `case`: Matches the selected menu item (`item_no`) to process size-specific logic.
    - Inner `case`: Matches the size (`size_choice`) and assigns the appropriate price to `price`.
3. **Example**:
    - If `item_no=1` (Chicken pot pie) and `size_choice=3`, it sets `price=200`.

Inventory Management

```
declare -A inventory
inventory=( [1]=50 [2]=100 [3]=70 [4]=60 [5]=80 [6]=200 [7]=150 [8]=30
[9]=100 [10]=90 )
```

1. **Purpose**: Tracks the stock levels for each menu item.
2. **How It Works**:
    - `declare -A` creates an associative array (key-value pairs).
    - Keys (`1` to `10`) represent menu item numbers, and values (e.g., `50`, `100`) indicate available quantities.
3. **Example**:
    - `inventory[1]` returns `50`, meaning 50 units of Chicken pot pie are in stock.

Stock Check and Update

```
if [ "${inventory[$item_no]}" -ge "$quantity" ]; then
    inventory[$item_no]=$((inventory[$item_no] - quantity))
    echo "Inventory updated. Remaining ${item_name}s: ${inventory[$item_no]}"
else
    echo "Sorry, insufficient inventory for $item_name. Only
${inventory[$item_no]} left."
    exit 1
fi
```

1. **Purpose**: Ensures the requested quantity is available.
2. **Logic**:
    - `-ge`: Checks if the available stock (`inventory[$item_no]`) is greater than or equal to the requested quantity (`quantity`).
    - If sufficient stock exists:
        - Decreases inventory by the ordered amount (`inventory[$item_no] - quantity`).
        - Displays updated inventory.
    - If insufficient:
        - Displays a message and exits the script.

Discount Application

```
if (( total_price > 1000 )); then
    discount=$(( total_price / 10 ))
    total_price=$(( total_price - discount ))
    echo "10% discount applied! Total price: $total_price Tk"
fi
```

1. **Purpose**: Applies a 10% discount if the total price exceeds 1000 Tk.
2. **Logic**:
    - `(( ))` is used for arithmetic comparisons.
    - If the condition is true:
        - Calculates a 10% discount (`total_price / 10`).
        - Subtracts the discount from `total_price`.

Gift Wrapping Option

```
if [ "$gift_choice" -eq 1 ]; then
    wrapping_charge=50
    total_price=$((total_price + wrapping_charge))
    echo "Gift wrapping charge added. Updated total price: $total_price Tk"
fi
```

1. **Purpose**: Adds a wrapping charge to the total if the user opts for gift wrapping.
2. **How It Works**:
    - Checks if `gift_choice` is `1` (yes).
    - Adds `50 Tk` to `total_price`.

Feedback System

```
if [ "$feedback_choice" -eq 1 ]; then
    read -p "Please enter your feedback: " feedback
    echo "$name: $feedback" >> feedback.txt
    echo "Thank you for your feedback!"
fi
```

1. **Purpose**: Collects user feedback and saves it to `feedback.txt`.
2. **How It Works**:
    - Prompts the user to enter feedback.
    - Appends the feedback with the user's name to `feedback.txt`.

Admin Panel

```
read -sp "Enter admin password to access admin panel: " admin_password
if [ "$admin_password" == "admin123" ]; then
    echo "Welcome, Admin!"
    ...
else
    echo "Invalid password."
fi
```

1. **Purpose**: Provides administrative features to manage orders, feedback, and employees.
2. **Logic**:
    - Prompts for a password in silent mode (`-sp` hides input).
    - Grants access if the password matches `admin123`.

Employee Management

```
case $emp_choice in
    1) echo "$emp_name, $emp_role, $emp_salary" >> employees.txt ;;
    ...
esac
```

1. Allows the admin to:
    - Add, remove, view, or update employee records.
    - Changes are stored in `employees.txt`.

This covers **line-by-line functionality**, but let me know if you'd like clarification on a specific segment!