Lightswitch: A Beginner's Guide to Scripting

Chapter 1: Introduction

Welcome to the Lightswitch Scripting Guide! Lightswitch is a simple yet powerful scripting language that lets you automate tasks, create functions, and build dynamic scripts. This guide will help you learn how to use Lightswitch effectively, starting from the basics and progressing to more advanced topics!

Chapter 2: Installation and Setup

To install Lightswitch, you need to make the `lightswitch.install.sh` script executable and run it. This script will automatically handle the installation process for you.

Steps:

1. Download the `lightswitch.install.sh` file by running “git clone https://github.com/asher37/Lightswitch
2. Enter the lightswitch directory “cd Lightswitch”
3. Make the script executable:
   ```
   chmod +x lightswitch.install.sh
   ```
4. Run the script:
   ```
   ./lightswitch.install.sh
   ```
5. Follow the on-screen instructions to complete the installation.

Once the installation is complete, you're ready to start scripting with Lightswitch!

Chapter 3: Basic Syntax and Structure

In this chapter, we'll cover the fundamental syntax and structure of Lightswitch scripts. Lightswitch is simple and intuitive, allowing you to write scripts quickly and easily.

1. Variables

Variables in Lightswitch are used to store data that can be accessed later. You can assign a value to a variable using the `set` command.

```
set x 5
```

2. Printing Values

You can print the value of a variable or any text using the `print` command.

```
print "Hello, World!"
```

3. Functions

Functions are reusable blocks of code that can be called with arguments.

```
function greet(name) {
    print "Hello, $name!"
}

greet "Alice"
```

Chapter 4: Control Structures

Lightswitch includes several control structures for conditional logic and loops.

1. If-Else Statements

You can make decisions in your script using the `if` statement.

```
set x 10
if $x -eq 10 {
    print "x is 10"
} else {
    print "x is not 10"
}
```

2. Loops

Lightswitch supports `for`, `while`, and `until` loops.

```
# For loop
for i in 1 2 3 4 5 {
    print $i
}

# While loop
set i 1
while $i -le 5 {
    print $i
    set i $((i + 1))
}
```

Chapter 5: Functions and Libraries

Functions are a key feature of Lightswitch. You can define your own functions to simplify your scripts and make them more modular. Additionally, you can create libraries of functions and reuse them across different scripts.

Example: Creating a Function

```
function add(a, b) {
    set result $((a + b))
    print "Result: $result"
}

add 3 4
```

Example: Creating a Library

```
# my_library.lightswitch
function greet() {
    print "Hello, $1!"
}
```

Chapter 6: Debugging and Troubleshooting Your Lightswitch Scripts

Debugging and troubleshooting are essential skills for any developer. In this chapter, we'll cover common issues you might encounter while working with Lightswitch scripts and provide techniques to identify and fix those issues.

1. Basic Debugging Techniques

You can use `print` statements to check the values of variables or trace the flow of your script.

```
function calculate_area() {
    local length=$1
    local width=$2
    print "Debug: length=$length, width=$width"
    local area=$((length * width))
    print "Area: $area"
}
```

2. Using the `set -x` Option for Detailed Trace Output

To trace the script's execution, use `set -x`.

```
set -x

function multiply() {
    local a=$1
    local b=$2
    local result=$((a * b))
    print "Result: $result"
}

multiply 3 4
```

3. Using Logs for Advanced Debugging

```
function log_message() {
    local message=$1
    echo "$(date +'%Y-%m-%d %H:%M:%S') - $message" >> /var/log/lightswitch.log
}
```

4. Common Issues and Solutions

- **Unexpected behavior with loops**: Ensure your loop terminates correctly.
- **Variable scope problems**: Check if your variables are being overwritten unintentionally.
- **Command not found errors**: Ensure system paths are set correctly.

Chapter 7: Advanced Scripting Techniques

In this chapter, we'll explore advanced scripting techniques that will help you take your Lightswitch scripts to the next level.

1. Creating Modular Scripts with Libraries

You can create reusable libraries and source them in your main script.

```
source "my_library.lightswitch"
greet "Alice"
```

2. Using Functions as Arguments

You can pass functions as arguments to other functions.

```
function run_task() {
    local task_function="$1"
    local task_argument="$2"
    $task_function "$task_argument"
}

function print_task() {
    print "Executing task: $1"
}

run_task "print_task" "Backup files"
```

3. Complex Data Structures

Lightswitch supports arrays and associative arrays.

```
# Array Example
function array_example() {
    local arr=("apple" "banana" "cherry")
    for item in "${arr[@]}"; do
        print "Item: $item"
    done
}

# Associative Array Example
function associative_array_example() {
    declare -A fruits
    fruits["apple"]="green"
    fruits["banana"]="yellow"
    for fruit in "${!fruits[@]}"; do
        print "$fruit is ${fruits[$fruit]}"
    done
}
```

4. Error Handling with Custom Messages

Handle errors by checking conditions and printing custom error messages.

```
function check_disk_space() {
    local threshold=80
    local disk_usage=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')

    if [[ "$disk_usage" -ge "$threshold" ]]; then
```

```
        print "Error: Disk space is above $threshold%. Current usage: $disk_usage%"
        exit 1
    else
        print "Disk space is healthy. Current usage: $disk_usage%"
    fi
}

check_disk_space
```

5. Automating System Maintenance

Automate system tasks like updates.

```
function update_system() {
    sudo apt update && sudo apt upgrade -y
    print "System update complete."
}

update_system
```

Chapter 8: Conclusion

In this guide, you've learned how to use Lightswitch to automate tasks, create functions, handle errors, and implement complex logic. By now, you should be comfortable writing Lightswitch scripts and using them for a variety of tasks.

Key Takeaways:

- Mastered basic syntax, variable assignments, and control structures.
- Explored advanced features like functions, error handling, and pattern matching.
- Gained troubleshooting and debugging skills.
- Learned how to modularize your scripts using libraries.

That wraps up the Lightswitch Scripting Guide. We hope you’ve found this guide useful and that it helps you as you build more complex scripts using Lightswitch. Continue exploring Lightswitch and experimenting with new features to make your scripts even more powerful!

This should now be easy to copy into LibreOffice Writer. Let me know if you'd like any other adjustments or additions!