Assignment Description

Background info.

Bash, short for "Bourne Again Shell", is a challenging scripting language and one of the widely-used command-line shell and scripting languages in the world of DevOps (Development and Operations) and automated environment setup. DevOps is a set of practices that combine software development (Dev) and IT operations (Ops) to automate and streamline the software delivery and infrastructure management processes. Some examples of using bash in DevOps and Automatic Environment Setup:

1. **Integration**: Bash can be easily integrated into various DevOps tools and platforms. It can call APIs, execute commands, and parse data from tools like version control systems, continuous integration servers, and container orchestration platforms, enabling seamless integration into the DevOps toolchain.
2. **Infrastructure as Code (IaC)**: Bash is often used to write scripts that define infrastructure as code. IaC enables the creation and management of infrastructure resources (e.g., servers, databases, networking) using code. Bash scripts can interact with cloud providers' APIs or configuration management tools, making it easier to

Bash plays often a significant role in operating systems in several ways. Here are some examples:

1. **Scripting and Automation:** In the context of operating systems, Bash is a powerful scripting language. It helps to write Bash scripts to automate various tasks, such as file management, process control, and system administration. Bash scripts can be used to automate repetitive tasks and perform system-wide operations.
2. **Debugging and Troubleshooting:** Bash can be used for debugging and troubleshooting purposes. Bash scripts can assist in diagnosing issues, checking logs, and performing system health checks, which are vital skills for system administrators and developers working on operating systems.
3. **File System Manipulation:** Understanding how the file system works is a central component of operating system studies. Bash provides commands for navigating, creating, modifying, and deleting files and directories, making it a valuable tool for teaching file system concepts.

Your goal.

The goal of this assignment is for you to implement a *Bash* script to **set up**, **install**, **uninstall**, **test**, and **manage** two software packages (applications) on the standard virtual machine, provided during the course (Debian 11).
This assignment uses methods like many scripts from software/framework developers to make your OS ready for programming and testing. Your script will install the following two packages:

- Nosecrets (https://github.com/bartobri/no-more-secrets)
- Pywebserver (https://github.com/nickjj/webserver#installation).

The implementation

Use the **folder/file structure** and **the Bash template** provided to perform the following tasks/implementations (have a look at the template).

The template contains several TODOs to provide some guidance for the implementation.

Validation checks must be everywhere inside the Bash script:

- **User Input and Parameter Values Validation**: validate user and non-user-provided arguments everywhere to ensure they are valid commands and options. Provide feedback if the arguments are not valid or recognized.

The *Setup()* function is responsible for the following tasks:

- **Dependency Checking**: It checks for the existence of required system dependencies: `"unzip", "wget",` and `"curl"`. Your task is to write, extract, and download the two packages using `"unzip", "wget",` and `"curl".` If these dependencies are not installed, it provides instructions for installing them before starting executing `install_packages()`
- **Directory and File Structure Check**: It checks whether certain directories and files exist in the specified installation directory. If they don't exist, it creates the necessary directory structure.

The *Install_package(), Uninstall_nosecrets()* and *Uninstall_pywebserver*() functions are responsible of the following tasks:

- **Generic Package Installation**: The code defines a generic function called `install_package()` to download packages from URLs and unzip them. This function can be used for both packages. Specific actions for each package are handled in separate sections within the same function body.
- **Application-Specific Installation and uninstallation**: Depending on the application name (e.g., "nosecrets" or "pywebserver") and the installation option (e.g., `"--install", "--uninstall", "--test"`), the code installs, uninstalls, or tests the corresponding application. This includes downloading, unzipping, compiling, and configuring as needed.

- **Directory and File Structure Check**: It checks whether a dedicated subdirectory exists for each package in the specified installation directory. If they don't exist, it creates the necessary directory structure. For example, to install “nosecrets” and “pywebserver” the package directory structure is:

  `./apps/nosecrets/`

  `./apps/pywebserver/`

Handle_error(), *Rollback_nosecrets() and Rollback_pywebserver() functions are responsible of the following tasks:*

- **Error Handling**: The code provides error handling using the `handle_error()` function. If errors occur during any step of the process, it prints error messages and, if specified, executes commands to handle the error (e.g., installing missing dependencies).
- **Reverse incomplete steps**: Some set of steps need to be executed together. If the set cannot be completed for some reason throw an error. Both `rollback` functions should clean and reverse to the starting point of the set.

Test_nosecrets() and *Test_pywebserver()* functions are responsible for the following tasks:

- **Testing**: Depending on the application name (e.g., "nosecrets" or "pywebserver") and the installation option (e.g.`"--test")` tests need to be executed. The template contains two functions to test each package installation. Both packages that need to be installed provide instructions to test the installation on their repository website.
  1. Nosecrets URL: https://github.com/bartobri/no-more-secrets

     Test by running the command: `ls -l | nms`

  2. Pywebserver URL: https://github.com/nickjj/webserver#installation

     There are multiple ways to test. The expected way for this assignment is to read the values from "`test.json`" and pass it to the pywebserver. See the example on the repository website.

Usage of *Config.conf* and *test.json*:

- **Configuration File Usage**: The code reads configuration settings from a file named "`config.conf`" to determine the URLs for downloading specific applications. It also uses data available in "`test.json`" to test the "pywebserver".

Remove() function is responsible for the following task:

- **Dependency, file, and directory structure:** Every dependency, folder, and file that was installed or created in the setup needs to be removed.

It is **mandatory** to keep the existing template code (and functions) as given. However, you have the freedom to add additional functions and code to complete the application (all in the same file). For example, the template assumes that **error handling is always done** using the generic function `handle_error()`. If something goes wrong halfway during a set of steps the function rollback can be used to reverse the steps.

Usage.

Your implementation needs to be tested on Linux Debian 11.

To run and test your Bash script implementation, change the path in the command line to the folder where the script is located. The following commands and options are expected to be used ("# " represents the bash prompt):

To run the setup and install the required dependencies.

```
# ./assignment.sh setup
```

To install each package.

```
# ./assignment.sh nosecrets --install

# ./assignment.sh pywebserver --install
```

To test the installation of each package

```
# ./assignment.sh nosecrets --test

# ./assignment.sh pywebserver --test
```

To uninstall each package

```
# ./assignment.sh nosecrets --uninstall

# ./assignment.sh pywebserver --uninstall
```

To clean up the download folder and remove installed dependencies from the OS.

```
# ./assignment.sh remove
```

Tip: test your application extensively before submission to handle all types of errors. For example, run every command mentioned above but do not provide all arguments and handle each error you encounter. Remove required files during testing and handle printed errors accordingly. This will make your implementation robust.

Grading.

To pass the requirements below need to be satisfied (VLD):

- The implementation follows the provided structure in the template.
- Each mentioned command above works as expected.
- No errors during the execution of the bash script.
- Comments are added to explain the provided logic.
- Use proper access level to commands, files, and folders that are strictly needed e.g. u+x

If any of the following problems are present, the script will be regarded as insufficient (NVL):

- The file is corrupted/incomplete/gives errors on the first run (e.g.: end-line Windows corruption).
- The implementation does not follow the provided structure in the template.
- One of the commands in the description is not working as expected.
- The added code is not organized in functions. Or added logic is not readable and/or not commented on.
- Any fatal error (e.g.: wrongly implemented requirement, missed implementation, etc.). The sum of less significant errors might also cause failing results if the requirements are not met in that case.
- Plagiarism (AI-generated code is plagiarism too, any usage of **automated generation of code is forbidden**).
- Using the root (e.g.: *su* or *sudo*) permission in combination with a command that does not require it. Similarly, the use of higher access levels than strictly needed for files and folders for users and groups when created.
- The delivery does not contain the students' names and students' numbers of all the participants.
- Double delivery (2 students submit the same assignment).

If no delivery is provided or the delivery is late, the grade will be ND.

The script will be tested in the given environment and might also be tested in the delivery system for plagiarism/nonsense.

Submission

It is allowed to deliver in groups of *up to 2* students. Each student will be responsible for the whole assignment.
The reviewers will always have the possibility to have an **oral check to verify the integrity and or originality** of the delivery.

The instructions for the submission will be added here later when it is ready, please bear with it, we must make many new setups and experiments.
The upload will be via "codegrade", and mandatory ONLY for one participant of each group.

Deadline

The deadline for this upload is the **3rd of November at 11:30 P.M**.