JC Scripts Collection

I hope you enjoy this collection of useful scripts! This repository contains various Bash scripts designed to simplify tasks, enhance productivity, and streamline workflows.

Table of Contents

Overview

> This repository includes scripts that automate tasks such as AWS profile management, device handling, and more. The scripts are organized into a compact library file (`scripts.ar`) to make distribution easy and efficient. Each script is documented and designed for ease of use, customization, and flexibility.

Prerequisites

> Before using this repository, make sure your environment meets the following requirements:

1. **Linux Operating System**

   Compatible with most Linux distributions (e.g., Ubuntu, Fedora, Debian, Arch).

2. **Required Tools**

- git

  > For cloning the repository and version control.

  ```
  sudo apt install git        # Debian/Ubuntu-based systems
  sudo dnf install git        # Fedora-based systems
  sudo pacman -S git          # Arch-based systems
  sudo apk add git            # Alpine
  ```

- ar

  > Utility: Used for creating the scripts library (scripts.ar).

```
sudo apt install binutils   # Debian/Ubuntu-based systems
sudo dnf install binutils   # Fedora-based systems
sudo pacman -S binutils     # Arch-based systems
sudo apk add binutils       # Alpine
```

- Bash Shell

  > Required to run the scripts. This is pre-installed on most Linux distributions.

- curl

  > Required for the one-liner installation method.

  Installing `curl`:

  - **Ubuntu/Debian-Based Systems**:

    ```
    sudo apt update
    sudo apt install curl -y
    ```

  - **Fedora/Red Hat-Based Systems**:

    ```
    sudo dnf install curl -y
    ```

  - **Arch-Based Systems**:

    ```
    sudo pacman -S curl
    ```

  - **Alpine Linux**:

    ```
    sudo apk add curl
    ```

3. **AWS**

   > In addition to the standard tools required for this project, users who intend to use the AWS scripts must install the AWS Command Line Interface (CLI). Below are the installation steps for various operating systems:

Prerequisites for Alpine Linux

> In addition to the general prerequisites for this project, users running Alpine Linux should ensure the following packages are installed:

Required Packages

1. file:

> Required for proper script detection.

```
sudo apk add file
```

2. Other Essential Tools (if not already installed)
   - bash (All scripts require Bash by default):

```
sudo apk add bash
```

   - findutils: To use a full-featured version of find.

```
sudo apk add findutils
```

3. Optional Tools:
   - column: For nicely formatted script lists (used with the -s option in the scripts command):

```
sudo apk add util-linux
```

Installing AWS CLI

> AWS CLI allows users to interact with AWS services directly from the command line. Follow the steps below to install it:

For Ubuntu and Debian-Based Systems

1. Update your package index:

```
sudo apt update
sudo apt install awscli -y
```

2. Verify the installation:

```
aws --version
```

For Fedora and Red Hat-Based Systems

1. Install AWS CLI via dnf:

```
sudo dnf install awscli -y
```

2. Check the version:

```
aws --version
```

For Arch-Based Systems

1. Install via pacman:

```
sudo pacman -S aws-cli
```

2. Confirm the installation:

```
aws --version
```

For Alpine Linux

1. Use the apk package manager:

```
sudo apk add aws-cli
```

2. Ensure AWS CLI is installed:

```
aws --version
```

Manual Installation

> Alternatively, users can manually download and install AWS CLI by following the official AWS documentation.

Getting Started

Steps to Set Up the Environment

1. Clone the repository:

```
git clone https://github.com/jcgarcia/jcscripts.git
mv jcscripts ~/.scripts && cd ~/.scripts
```

2. Run the setup script:

```
./installscripts
```

3. The setup script will:
    - Verify that all required components are present, including README.md, README.pdf, .jcscripts, installscripts, and lib/scripts.ar.
    - Extract scripts from the scripts.ar library into the directory specified in .jcscripts (default: ~/.scripts).
    - Make valid shell scripts executable.
    - Add the .scripts directory to your system's PATH for easy access.
4. After installation, you can call any script directly by its name from any location:

```
script_name
```

One-Liner Installation

To quickly install the JC Scripts Collection, run the following command:

```
bash <(curl -sSL
https://raw.githubusercontent.com/jcgarcia/jcscripts/main/installscripts)
```

> **Note**: Ensure you have internet access and `curl` installed on your system.

This command will:

1. Download and execute the `installscripts` script directly.
2. Set up the environment for the JC Scripts Collection.

After installation, you can use the scripts directly from any location.

Configurations

The `.jcscripts` file contains configuration options for the scripts repository. Below are the configurable values:

- **EDITOR**: Specifies the text editor for editing scripts (default: `vi`).
- **SCRIPTS_DIR**: Defines the directory where the scripts will be stored (default: `~/.scripts`).
- **SSH_KEY**: Path to the SSH key for accessing GitHub (default: `~/.ssh/personalkey.pem` you must generate your own key and change that in the config file).

Generating an SSH Key

To generate your own SSH key:

1. Run the provided script, and follow the prompts to create the key.

```
./generate_ssh_key
```

2. Follow the prompts to create the key.

- Copy the contents of the public key file (~/.ssh/personalkey.pem.pub):

```
cat ~/.ssh/personalkey.pem.pub
```

- Log in to GitHub and navigate to Settings > SSH and GPG Keys > New SSH Key. Adding a new SSH key to your GitHub account
- Paste the contents into the Key field, give it a meaningful title, and save.

Example .jcscripts file:

```
# Configuration for scripts
EDITOR="vi"
SCRIPTS_DIR="~/.scripts"
```

Available Scripts

Here are all the scripts included in this repository:

- **awsaccess**: Manage AWS Identity Center operations like users and groups.
- **awsaddprofile**: Interactively add new AWS profiles and validate login.
- **awslogin**: Dynamically sets AWS default profile after SSO login.
- **awsmoveaccount**: Move AWS accounts between organizational units.
- **awsnewaccount**: Create a new AWS account, wait for creation, and verify membership in the organization.
- **awsorg**: Fetch and display AWS Organization layout in multiple formats.

- **awsres**: Dynamically retrieve AWS resource information.
- **awsresources**: Manage AWS resources efficiently.
- **wslops**: wsl operations for your distros (install, remove, backup, restore).
- **connecttoaws**: Automate AWS connection processes.
- **createOrgAdm**: Automates the creation of an organization administrator profile.
- **createscripts**: Generate and manage new scripts using a predefined template.
- **device**: Dynamically list, mount, and unmount devices, including WSL-mounted drives.
- **edit**: Edit an existing script or create one if it doesn't exist.
- **getamiinfo**: Fetch and display information about AWS AMIs (Amazon Machine Images).
- **installscripts**: Automates the installation and setup of scripts from the repository.
- **listownamis**: Lists owned AMIs for your AWS account.
- **publish**: Creates a clean distribution package for the repository, ready to upload to GitHub.
- **newkey**: Automates the process of generating a new SSH key for secure access to GitHub and updates the `.jcscripts` configuration file with the new key path. This ensures seamless integration with the repository's settings.
- **scripts**: Lists all available scripts in the .scripts directory.
- **template**: A sample script template for creating new Bash scripts.

For detailed instructions on using each script, refer to the comments at the beginning of the script files. If you're unsure how to get started, see the Getting Started section above.

Libraries

`colorfunctions`

The `colorfunctions` library provides functions for colorized output using `tput`. It includes functions to print messages in standard and bright colors.

Functions

- **Standard Colors**:
  - `printred`: Print a message in red.
  - `printgreen`: Print a message in green.
  - `printyellow`: Print a message in yellow.
  - `printblue`: Print a message in blue.
  - `printmagenta`: Print a message in magenta.
  - `printcyan`: Print a message in cyan.

    - `printwhite`: Print a message in white.

- **Bright Colors**:

    - `printbrightblack`: Print a message in bright black (dark gray).
    - `printbrightred`: Print a message in bright red.
    - `printbrightgreen`: Print a message in bright green.
    - `printbrightyellow`: Print a message in bright yellow.
    - `printbrightblue`: Print a message in bright blue.
    - `printbrightmagenta`: Print a message in bright magenta.
    - `printbrightcyan`: Print a message in bright cyan.
    - `printbrightwhite`: Print a message in bright white.

`wslfunctions`

The `wslfunctions` library provides functions to manage WSL (Windows Subsystem for Linux) distributions.

Functions

- `convert_to_windows_path`: Convert a Linux path to a Windows path.
- `get_wsl_path`: Get the path to the `wsl.exe` executable.
- `get_powershell_path`: Get the path to the `powershell.exe` executable.
- `list_distributions`: List all WSL distributions.
- `list_online_numbered_distributions`: List available WSL distributions online.
- `install_distro`: Install a WSL distribution.
- `list_mumbered_installed_distributions`: List installed WSL distributions with numbers.
- `is_distro_running`: Check if a WSL distribution is running.
- `terminate_distro`: Terminate a running WSL distribution.
- `unregister_distro`: Unregister a WSL distribution.
- `validate_distro`: Validate the existence of a WSL distribution.
- `get_default_distro`: Get the default WSL distribution.
- `get_current_distro`: Get the currently running WSL distribution.
- `get_active_distro`: Get the active WSL distribution.
- `select_distro`: Prompt the user to select a WSL distribution.
- `get_windows_user`: Detect the Windows username.
- `clean_residual_files`: Remove residual files for a WSL distribution.
- `restore_distro`: Restore a WSL distribution from a backup.
- `backup_distro_function`: Backup a WSL distribution.
- `install_distro_function`: Install a WSL distribution interactively.
- `remove_distro_function`: Remove a WSL distribution interactively.

`awsfunctions`

The `awsfunctions` library provides functions to manage AWS operations.

Functions

- `aws_login`: Log in to AWS using SSO.
- `aws_add_profile`: Add a new AWS profile interactively.

- `aws_list_profiles`: List all AWS profiles.
- `aws_get_account_info`: Retrieve information about an AWS account.
- `aws_create_account`: Create a new AWS account.
- `aws_delete_account`: Delete an AWS account.
- `aws_list_resources`: List AWS resources for a given profile.
- `aws_backup_resources`: Backup AWS resources to a file.
- `aws_restore_resources`: Restore AWS resources from a backup file.
- `aws_validate_profile`: Validate the credentials of an AWS profile.

Contributing

> Contributions are welcome! If you'd like to contribute, please:

1. Fork this repository.
2. Create a new branch for your feature or bug fix.
3. Commit your changes and submit a pull request.

License

> This repository is released under the MIT License. Feel free to use and adapt the scripts as needed.

Additional Notes

> If the setup script detects that the environment has already been initialized, it will notify you and avoid reinitializing. To force reinstallation, delete the marker file:

```
rm ~/.scripts_installed
```

> To customize the editor or scripts directory, modify .jcscripts before running installscripts.

Fixing PATH Export Issue on Alpine Linux

If the `PATH` export is not working on Alpine Linux, follow these steps to ensure the `.scripts` directory is added to your `PATH`:

1. Open the shell configuration file for editing. Depending on your shell, this could be:
   - For `bash`:

     ```
     nano ~/.bashrc
     ```

   - For `ash` (default shell on Alpine):

     ```
     nano ~/.profile
     ```

2. Add the following line to export the `.scripts` directory to your `PATH`:

```
export PATH="$HOME/.scripts:$PATH"
```

3. Save the file and reload the shell configuration:
   - For `bash`:

```
source ~/.bashrc
```

   - For `ash`:

```
source ~/.profile
```

4. Verify that the `.scripts` directory has been added to your `PATH`:

```
echo $PATH
```

5. Test the setup by running any script from the `.scripts` directory:

```
script_name
```

If the issue persists, ensure that the `.scripts` directory exists and contains the scripts:

```
ls -l ~/.scripts
```