Tutorial: Automating Configuration File Backup with `fetch_config.sh`

System administrators often need to ensure that configuration files are regularly backed up, especially the ones that are actively in use. These files can include system configurations, application preferences, or user-specific settings. In this tutorial, we'll walk through a bash script that automates the process of identifying and backing up configuration files that have been accessed within the last 90 days.

This process ensures that the most relevant and active configuration files are saved in a convenient ZIP archive for safekeeping. This tutorial will explain each step in detail, showing you how this script works and why each part is essential.

Why Automate Configuration Backup?

Configuration files are crucial for system and software behavior. When systems are updated, reinstalled, or need troubleshooting, having a backup of the configurations is invaluable. Automating this task saves time, reduces the risk of human error, and ensures that backups are created regularly.

By focusing on files accessed within the last 90 days, we prioritize the most relevant settings, reducing clutter and unnecessary backups.

How the `fetch_config.sh` Script Works

This script traverses through user directories, checks which configuration files have been accessed in the last 90 days, copies them to a temporary directory, and archives them into a ZIP file. After the backup is completed, the temporary files are cleaned up.

Let's break down the script:

Step 1: Create a Temporary Directory

At the start, the script creates a temporary directory using `mktemp` to hold the configuration files before archiving.

```
# Temporary directory for storing used config files
temp_dir=$(mktemp -d)
archive_name="used_config_files.zip"
```

- **`mktemp -d`**: This creates a unique temporary directory where the files will be copied.
- **`archive_name`**: This defines the name of the final ZIP archive.

Using a temporary directory ensures that we can collect the files without affecting their original location or structure.

Step 2: Check If a Configuration File Is in Use

The `is_in_use` function checks whether a given file has been accessed in the last 90 days. This is done using the `find` command with the `-atime` flag, which checks the file's access time.

```
is_in_use() {
    local config_file=$1
    if [ -f "$config_file" ]; then
        if [ $(find "$config_file" -type f -atime -90 2>/dev/null) ];
        then
            return 0
        fi
    fi
    return 1
}
```

- **`find "$config_file" -atime -90`**: This searches for files accessed in the last 90 days.
- **`return 0`**: If a file is in use (i.e., accessed recently), the function returns `0`, which is standard in bash for success.
- **`return 1`**: If the file hasn't been accessed, it returns `1`, indicating it won't be archived.

This function ensures that only active files (files that have been used recently) are backed up.

Step 3: Traverse User Directories and Copy Files

Next, the script loops through each user's home directory to find configuration files. It checks hidden files and `.config` directories and uses the `is_in_use` function to decide whether to back them up.

```
for user_home in /home/*; do
    if [ -d "$user_home" ]; then
        # Process hidden files in the user's home directory
        find "$user_home" -maxdepth 1 -name ".*" -type f | while read
        config_file; do
            if is_in_use "$config_file"; then
                dest_dir="$temp_dir${config_file#/home}"
                mkdir -p "$(dirname "$dest_dir")"
                cp "$config_file" "$dest_dir"
            fi
        done

        if [ -d "$user_home/.config" ]; then
            find "$user_home/.config" -type f | while read
        config_file; do
                if is_in_use "$config_file"; then
                    dest_dir="$temp_dir${config_file#/home}"
```

```
                    mkdir -p "$(dirname "$dest_dir")"
                    cp "$config_file" "$dest_dir"
                fi
            done
        fi
    fi
done
```

- **`/home/*`**: The script checks each home directory on the system.
- **`find "$user_home" -name ".*"`**: This finds hidden files in the home directory (common for configuration files).
- **`find "$user_home/.config"`**: This searches the `.config` directory, where many user application configurations are stored.

For every configuration file found, it creates the corresponding directory structure inside the temporary folder and copies the file into it.

Step 4: Create a ZIP Archive of the Files

Once the files are copied to the temporary directory, the script uses the `zip` command to archive them into a single file.

```
# Create a zip archive from the copied files
zip -r "$archive_name" "$temp_dir"/*
```

- **`zip -r`**: This command recursively creates a ZIP file. The `-r` flag ensures that all files and directories inside the temporary folder are included in the archive.

This step is crucial as it compresses all the files into a single archive, making it easy to move or store.

Step 5: Clean Up Temporary Files

After creating the ZIP archive, the temporary directory is no longer needed. The script removes it to avoid leaving any unnecessary files behind.

```
# Clean up the temporary directory
rm -rf "$temp_dir"
```

- **`rm -rf "$temp_dir"`**: This command forcefully deletes the temporary directory and all its contents.

This ensures that the system doesn't accumulate temporary files over time, which could eventually take up valuable disk space.

Step 6: Notify the User

The script finishes by informing the user that the task is complete and the archive has been created.

```
echo "Used configuration files have been archived into $archive_name."
```

This feedback is helpful because it confirms that the backup was successful and provides the name of the generated archive.

Conclusion

This script provides a simple, automated way to back up active configuration files. By focusing on files that have been accessed recently, you reduce clutter and ensure that you're only backing up relevant configurations. It is especially useful for administrators who want to set up regular backups via cron jobs or those who need to quickly archive important user settings.

Key Benefits of Using This Script:

- **Automated Backups**: Save time by automating the backup process for configuration files.
- **Relevance**: Focuses only on files that have been used in the last 90 days, keeping backups lean and efficient.
- **Customization**: Easily modify the script to change the time window (90 days) or the directories to scan.

Feel free to customize this script for your specific needs, whether that involves modifying the time frame for file access or targeting specific configuration directories!

If you have any questions or need further clarification, feel free to ask!