**LINUX PROJECT: BACKUP & RESTORE**

**Objective**

Practice file backup and restoration techniques.

This project demonstrates a complete backup and restoration workflow using Linux utilities like tar, gzip, rsync, and find. It covers real-world scenarios such as selective archiving, disaster recovery, and audit-ready verification. All operations are performed on a sample “project\_data” directory containing configuration files, documentation, and image assets.

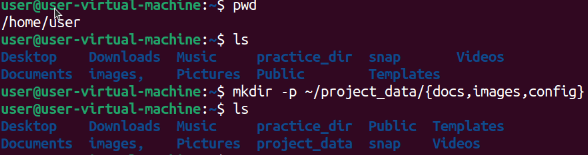
1. **Create sample directory structure**

**Command**

mkdir -p ~/project\_data/{docs,images,config}

**Explanation**

This command creates multiple directories “*docs, images, and config*” under the “*project\_data*“ folder in the user's home directory. The “*-p*” flag ensures that the parent directory “*project\_data*” is created if it doesn't exist and suppresses errors if any target directories already exist.





**Post-Creation Verification**

Given “*ls project\_data/”* command to list the contents of the newly created directory. The output showed config, docs, and images, confirming that all intended subdirectories were successfully created.

**Command**

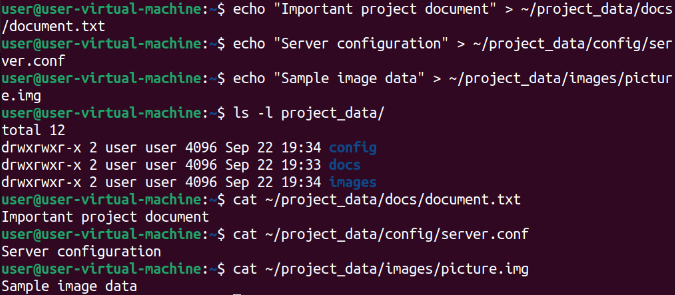
echo "Important project document" > ~/project\_data/docs/document1.txt

echo "Server configuration" > ~/project\_data/config/server.conf

echo "Sample image data" > ~/project\_data/images/picture1.img

**Explanation**

The “*echo”* command is used three times to create files with predefined content: *document.txt* in the “docs directory”, *server.conf* in the “config directory”, and *picture.img* in the “images directory”—all under the *project\_data* folder in the user's home directory. Each command uses output redirection (>) to write a string directly into the respective file.



**Post-Creation Verification**

Given “*ls -l project\_data/”* command to list the contents of the *project\_data* directory in long format, which confirms the presence of the three subdirectories.

To verify the contents of each file, the cat command is used on each one individually. This displays the exact text given to each file.

1. **Create backups using different methods**

**Command**

tar -czvf project\_backup\_full.tar.gz ~/project\_data/

**Explanation**

This command is used to compress the entire “*project\_data”* directory into “.tar.gz” archive.

Here,

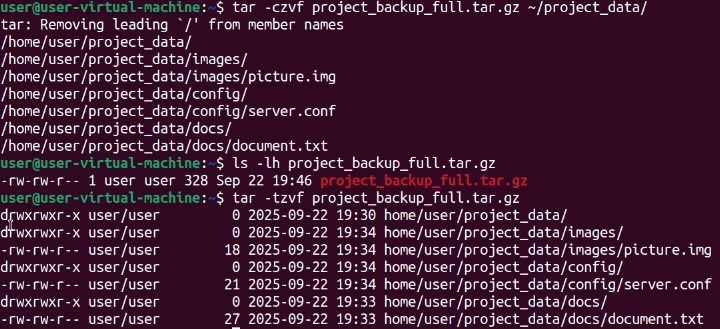
-c creates the archive,

-z enables gzip compression,

-v provides verbose output showing each file being added, and

-f specifies the archive file name.

The output confirms that all subdirectories “*docs, images, config*” and their respective files “*document.txt, picture.img, server.conf*” were successfully included in the archive.



**Post-Creation Verification**

To verify this, given “*ls -lh project\_backup\_full.tar.gz*” which displays detailed metadata including file size (328K), permissions, ownership, and timestamp - confirming the archive exists and is properly formatted.

Further verification is done “*tar -tzvf*” which is used to inspect the contents of the archive without extracting it. Where,

-t: Lists the files stored in the archive.

-z: Indicates that the archive is compressed using gzip.

-v: Enables verbose output, showing detailed file paths and metadata.

-f: Specifies the archive file to be read.

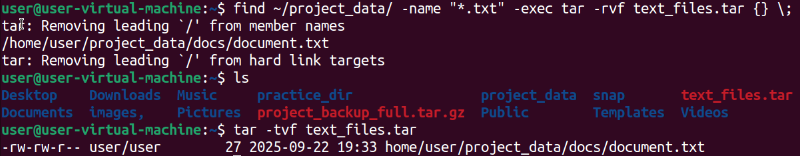
It confirms that the archive contains the complete directory structure and all expected files. This is a crucial verification step in backup workflows, ensuring that the archive is not only present but also correctly populated before relying on it for restoration or transfer.

**Command**

find ~/project\_data/ -name "\*.txt" -exec tar -rvf text\_files.tar {} \;

**Explanation**

This command searches recursively within the “*project\_data”* directory for all files ending with the “.*txt*” extension. For each matching file, it appends the file to an archive named “*text\_files.tar*” using the tar command with the “-r (append) and -v (verbose)” options. This approach is particularly useful for dynamically archiving specific file types without manually listing each one. This creates a **non-compressed tar archive** containing only .txt files.



**Post-Creation Verification**

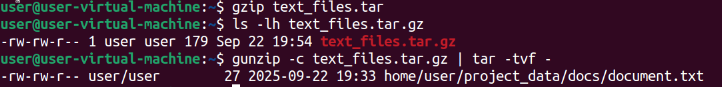
To verify this the ls command confirms its presence in the current directory. To verify the contents of the archive, “*tar -tvf text\_files.tar*” is executed, listing the archived file “*document.txt*” along with its metadata—permissions, ownership, size, and timestamp. This sequence validates that the correct file was located, archived, and stored with accurate attributes

**Command**

gzip text\_files.tar

**Explanation**

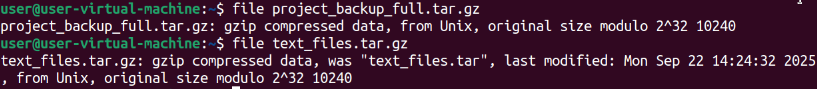
This command compresses the existing archive “*text\_files.tar*” into a “.*gz*” format, resulting in “*text\_files.tar.gz”*. This is a common method for reducing file size and preparing archives for storage or transfer.



**Post-Creation Verification**

To verify compression, “*ls -lh text\_files.tar.gz*” is executed, displaying metadata such as file permissions, ownership, size (179 bytes), and timestamp—confirming successful creation of the compressed archive. Instead of fully extracting the archive, “*gunzip -c text\_files.tar.gz | tar -tvf –“,* is given which decompresses the archive in-memory and pipes its contents directly into the tar command for listing. This technique is efficient for inspecting the contents without altering the file system. Decompress text\_files.tar.gz and immediately list the contents of the resulting .tar archive, without saving anything to disk.

The output confirms that the archive contains document.txt located in home/user/project\_data/docs/, along with its size, permissions, and modification date. This approach is ideal for validating compressed backups, ensuring that critical files are included and intact before extraction or deployment.



The file command was given to verifying the type and integrity of the compressed archive files without opening or extracting them. It ensures the archive wasn’t corrupted or misnamed. The output confirms that,

* The file is gzip-compressed.
* It originated from a .tar archive.
* The last modified date and original size are intact.
* It's compatible with Unix systems.

1. **Create incremental backup with rsync**

**Command**

mkdir ~/backup\_location

**Explanation**

This command used to create a new directory named “*backup\_location*” inside the user's home directory (~). This directory is intended to serve as a dedicated location for storing backup archives or related files.



**Post-Creation Verification**

To verify “*ls -ld*” is used to confirm that the directory was successfully created.

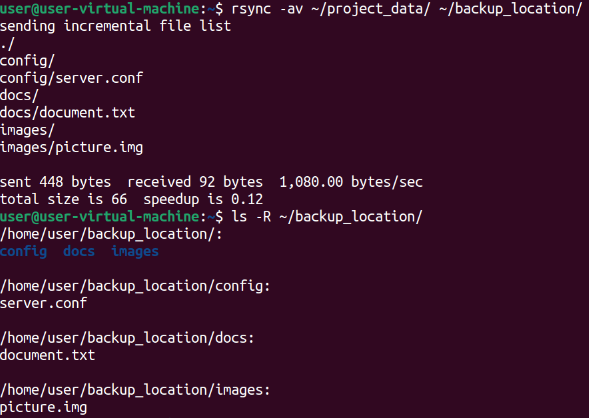
**Command**

rsync -av ~/project\_data/ ~/backup\_location/

**Explanation**

This command rsync synchronizes the contents of the “*project\_data”* directory into the

“*backup\_location*” directory. The “-*a*” (archive) option preserves file permissions, timestamps, symbolic links, and directory structure, while “-*v*” (verbose) provides detailed output during the transfer. This ensures a reliable and consistent backup of all files and subdirectories. The transfer summary confirms that all key files “*document.txt, server.conf, and picture.img*” were successfully copied along with their respective folders.



**Post-Creation Verification**

To verify the backup, “*ls -R ~/backup\_location/”* is executed, which recursively lists the contents of the “*backup\_location*” directory. The output confirms that all subdirectories (docs, config, images) and their respective files are present and correctly placed. This step validates the integrity of the backup and confirms that the directory structure and data have been accurately replicated.

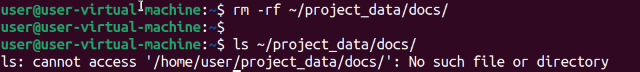
1. **Simulate data loss and restore**

**Command**

rm -rf ~/project\_data/docs/

**Explanation**

Here, first “rm -*rf*” command forcefully removes the docs directory located inside “*project\_data”* with the “-*r*” (recursive) and “-*f*” (force) options. This deletes the directory and all its contents without prompting for confirmation.



**Post-Creation Verification**

To verify “*ls ~/project\_data/docs/”* is given to list the contents of the now-deleted docs directory and the output confirms that the docs directory has been successfully removed. The error message serves as a verification step, indicating that the target directory no longer exists in the file system.

**Command**

tar -xzvf project\_backup\_full.tar.gz -C ~/

**Explanation**

Here, the “*tar*” command extracts the contents of the compressed archive “*project\_backup\_full.tar.gz*” into the user's home directory (~).

The options used are,

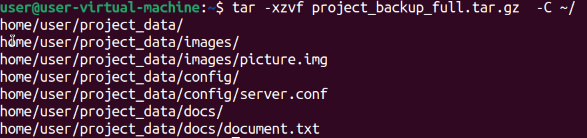
-x for extract,

-z for gzip decompression,

-v for verbose output, and

-f to specify the archive file.

The “-*C ~/”* option ensures that extraction occurs directly into the home directory. The output confirms that all expected files and subdirectories were unpacked into “*project\_data”*.



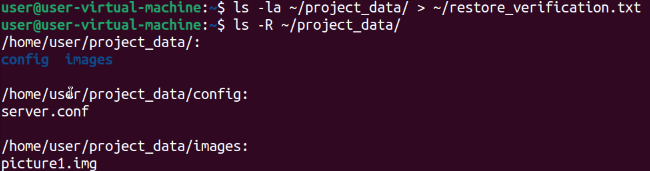
1. **Verify restoration**

**Command**

ls -la ~/project\_data/ > ~/restore\_verification.txt

**Explanation**

Here, the “*ls*” command lists all files and directories within “*project\_data”* in long format, including hidden files, and redirects the output to a file named restore\_verification.txt in the user's home directory where no output was shown.



**Error**

The second command recursively lists all contents of “*project\_data”*, displaying the full hierarchy of subdirectories and files. But here the listing confirms that the “*project\_data”* directory contains only two subdirectories: config and images and inside these, the files “*server.conf*” and “*picture1.img*” are present respectively. The absence of the “*docs directory*” and its “*document.txt*” file indicates that the restoration was partial or that the docs folder was not included in the extracted archive.

When this command was used “*tar -czvf project\_backup\_full.tar.gz ~/project\_data/*”, here I’m “*tar*” to archive the entire path starting from your home directory. So, inside the tarball, the structure looks like:

home/harika/project\_data/docs/document1.txt

home/harika/project\_data/images/picture1.img

home/harika/project\_data/config/server.conf

when extracting it with the command “*tar -xzvf project\_backup\_full.tar.gz -C ~/”* it says “Unpack this tarball into my home directory”. But the tarball contains “*home/harika/project\_data/”,* so it again creates another home directory “~/*home/harika/project\_data*/” which is not same as the original directory. So, your restored files end up nested inside “~/home/harika/project\_data/” which is second home directory, not where it expected to be.

**Resolution**

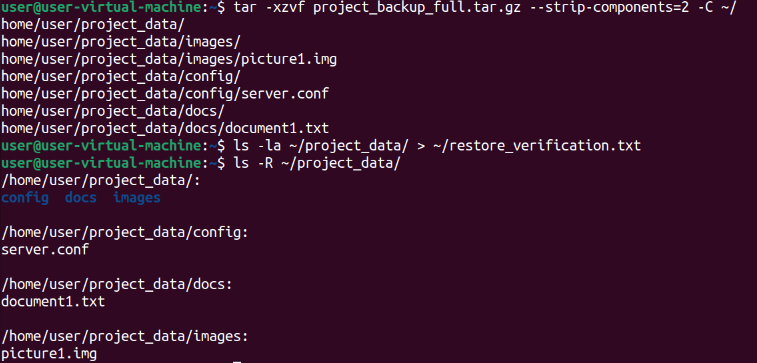
**Command**

tar -xzvf project\_backup\_full.tar.gz --strip-components=2 -C ~/

**Explanation**

To resolve the earlier directory issue, used the above command to extracts the contents of the “*project\_backup\_full.tar.gz*” archive into the user's home directory while removing the first two leading path components from each file inside the archive.

The *“--strip-components=2*” option ensures that deeply nested paths like “*home/user/project\_data/docs/document.txt*” are flattened to just “document.txt”, and “-*C ~/”* placing files directly under “~/” home directory. This is especially useful when the archive contains full absolute paths that would otherwise recreate the entire directory tree.



**Post-Creation Verification**

To output confirms that all expected files “*document.txt, server.conf, and picture.img*” were

extracted. By stripping the first two path components, the issue encountered earlier “*ls: cannot*

*access '/home/user/project\_data/docs/*'” is resolved, as the files are no longer nested.

This approach simplifies restoration and avoids path conflicts, ensuring that the

extracted files are immediately accessible in the target directory.

**Real-world scenarios**

**Disaster Recovery Simulation in a Linux Environment**

**Context**: A system administrator simulates accidental deletion of a directory and tests restoration from backup.

**Why It Matters**: Validates the reliability of backup archives and the ability to recover from real-world data loss scenarios.

**Daily Backup of Project Artifacts in a DevOps Pipeline**

**Context**: A software team maintains a project data directory containing configuration files, documentation, and image assets. These need to be backed up daily before CI/CD jobs run.

**Why It Matters**: Ensures that every deployment starts with a clean, restorable snapshot. Prevents data loss and supports rollback strategies.

**Audit-Ready Backup Verification for Compliance Teams**

**Context**: A compliance team needs proof that backups are complete and restorable.

**Why It Matters**: Provides a transparent, verifiable trail of backup and restoration activities—essential for audits and regulatory checks.

**Training New Engineers on Linux File Management & Backup**

**Context**: As part of onboarding, junior engineers are taught how to manage files, create backups, and verify restoration using Linux commands.

**Why It Matters**: Builds foundational skills in shell scripting, file system navigation, and backup hygiene. Also introduces audit logging via ls -la > restore\_verification.txt.