TUTORIAL

How to Monitor Ubuntu Disk Usage with Command Line Tools

Learn how to monitor Ubuntu disk usage from the command line using df, du, ncdu, and scheduled checks. This tutorial shows what to measure, how to validate results, and how to turn output into a practical operational workflow.

Operating Systems - July 19, 2026

Why disk monitoring matters on Ubuntu

Disk exhaustion is one of the most common causes of avoidable service disruption on Ubuntu systems. When a filesystem reaches capacity, applications can fail to write logs, databases may stop accepting transactions, package updates can break, and even basic administrative actions can become unreliable. The operational problem is not just ?is the disk full?? but ?which filesystem is filling, what is consuming it, and how do I verify the system is still healthy before it crosses a critical threshold??

This tutorial shows how to monitor Ubuntu disk usage with command-line tools so you can identify pressure early, confirm where capacity is being consumed, and establish a repeatable check for production use. By the end, you will be able to inspect filesystem usage, find large directories, validate inode pressure, and build a simple workflow for ongoing monitoring.

Prerequisites and stop-here checks

Before you begin, make sure you can access the host through a shell with permission to read the filesystems you want to inspect. Most commands in this guide are safe for standard operational use, but some directories may require sudo to get an accurate picture.

Stop here if:

- The host is in active storage recovery, RAID rebuild, or filesystem repair mode.
- You are working on a mounted snapshot or backup target and do not intend to measure that data store.
- You do not know which mount points are expected to be local disk, network storage, or ephemeral volumes.

That last point matters because command output is only useful when you interpret the correct filesystem. For example, /var, /home, and / may be on different devices, and a network mount can fail or report usage differently than local disk.

Step 1: Identify which filesystems are actually at risk

Goal

Find the mount points and filesystems whose growth can affect service availability.

Action

Use df to review mounted filesystems and their available space:

The -h option makes sizes readable, and -T includes the filesystem type. For a more focused view, sort the output by usage and inspect the mount points that matter operationally, such as /, /var, /home, or application-specific data paths.

If you want to include inode pressure, which is separate from byte usage, run:

Expected output

You should see each mounted filesystem with total size, used space, available space, usage percentage, and mount point. For inode checks, you should also see inode totals and usage percentages.

Validation

Validate that you are looking at the right device and mount point before making decisions. A filesystem can have free bytes but still fail because it has run out of inodes.

Common failure

A common mistake is to inspect only the root filesystem and ignore separate mounts. Another is to confuse a bind mount or network mount with the underlying storage device.

Practical decision rule

If df -hT shows a filesystem at or above your alert threshold, move immediately to directory-level analysis. If df -ih shows inode usage near capacity, treat it as an incident even if byte usage looks normal.

For teams that also manage host access policies, disk monitoring often pairs well with operational hardening work such as How to Secure SSH on Ubuntu with Key-Based Authentication, because you want reliable access before a storage problem becomes a service outage.

Step 2: Locate the directories consuming space

Goal

Find which directories contribute most to filesystem growth.

Action

Use du to summarize usage for a target path. Start at a high-level directory that is on the affected filesystem:

This command does three useful things:

- -x stays on one filesystem, which prevents unrelated mounts from distorting the result.
- -h produces human-readable sizes.
- --max-depth=1 keeps the output manageable while showing top-level consumers.

If you need a different location, replace /var with the relevant mount path. Repeat the command in the largest child directory to narrow the search.

Expected output

You should get a short list of directories with size totals. The largest entries are the ones to inspect next.

Validation

Compare the du total for the top-level path with the df usage on the same filesystem. They will not always match exactly, but they should be directionally consistent. A very large gap can indicate deleted-but-open files, hidden mount points, or files that changed during the scan.

Common failure

The most common error is forgetting -x. Without it, du can cross into mounted filesystems and overstate the usage of the path you are actually trying to analyze.

Practical interpretation

Use du when you need a directory hierarchy view. Use df when you need filesystem-level capacity and free-space status. Those questions are related, but they are not interchangeable.

Step 3: Use an interactive view when you need faster triage

Goal

Inspect usage interactively when text output is too broad or when you need to drill down quickly on a live host.

Action

Install and run ncdu if it is available in your environment:

If your organization restricts package installation on production systems, use this step only on approved maintenance hosts or temporary troubleshooting shells.

Expected output

You should get a navigable terminal interface that lists directories by size, allowing you to expand and inspect large consumers without repeatedly running du.

Validation

Confirm that ncdu is restricted to the intended filesystem with -x. Verify that the largest entries correspond to the same directories identified by df and du.

Common failure

Do not rely on interactive inspection alone if you need a record for change management or incident notes. The terminal view is useful for triage, but you may still need to capture command output for auditability.

When to use it

ncdu is especially useful when a filesystem contains many nested application directories, rotated logs, or container data. It reduces the time spent hunting through large trees with repeated shell commands.

Step 4: Check for log growth, cache buildup, and old data

Goal

Identify common operational causes of disk consumption so you can distinguish normal growth from avoidable accumulation.

Action

Focus on directories that typically expand under load:

- /var/log for logs and rotated logs

- /var/lib for application state, package metadata, container data, and databases
- /tmp and /var/tmp for temporary files
- Application-specific data directories under /srv, /opt, or custom mounts

A simple sequence often works well:

If one application directory dominates, drill down further. For example, a log directory may be large because of one unbounded file, while a data directory may be large because retention is too aggressive.

Expected output

You should be able to tell whether the growth is due to logs, temporary data, package caches, application state, or a legitimate data increase.

Validation

Validate that the directory owner or service is expected to hold that data. A large /var/lib entry may be normal for a database or container runtime, but unusual for a host that should be mostly static.

Common failure

A frequent mistake is treating all growth as a cleanup problem. Some growth is legitimate and should be addressed with capacity planning, not deletion.

Step 5: Verify hidden disk pressure conditions

Goal

Detect situations where available space looks acceptable but the filesystem can still fail.

Action

Check for inode exhaustion and deleted-but-open files.

For inode exhaustion:

For open files that were deleted but still hold space, inspect the processes with open file handles when the filesystem usage does not match directory totals. A common first pass is:

That command lists open files with link count below 1, which can reveal deleted files still held open by running processes.

Expected output

For inode issues, you should see a filesystem with very high inode usage. For deleted-but-open files, you should see process names, file paths, and file descriptors that explain why space is not returning to the filesystem after deletion.

Validation

Match the suspicious process to a service you recognize before taking action. Restarting the wrong process can cause unnecessary disruption.

Common failure

The most common failure here is assuming a deletion immediately frees capacity. If a process still has the file open, the space is not reclaimed until the file descriptor closes.

Operational follow-up

If you find deleted-but-open files, validate the service?s restart behavior and maintenance window before restarting it. In environments with strict access controls, it is worth verifying that your administrative access workflow is already hardened so recovery work is not blocked during an incident.

Step 6: Turn a manual check into a repeatable command

Goal

Create a compact disk-usage check that can be run during maintenance, incident response, or scheduled monitoring.

Action

A practical shell snippet for a single filesystem might look like this:

Save this as a local operational script and run it against the mount point you care about most. If your environment has multiple critical filesystems, repeat the logic for each one separately rather than trying to report everything in one noisy output stream.

Expected output

You should see the filesystem-level status first, followed by the largest immediate subdirectories.

Validation

Confirm that the script uses -x or an equivalent one-filesystem constraint when you are measuring a single mount point. Confirm the output is readable enough to compare across runs.

Common failure

A frequent problem is making the script too generic too soon. Keep the first version simple and targeted so the output is easy to interpret during an incident.

Step 7: Set alert thresholds and operational checks

Goal

Define when disk usage becomes a meaningful operational risk.

Action

Decide on thresholds based on service sensitivity and free-space requirements. For example, a database host may need more headroom than a static web server, and log-heavy systems may need a more conservative warning threshold.

At minimum, define checks for:

- Byte usage on critical filesystems
- Inode usage on filesystems that hold many small files
- Sudden growth in /var/log or application data directories
- Deleted-but-open files that prevent space from being reclaimed

If you already use monitoring for host availability and remote access, align disk checks with the same maintenance workflow you use for other operational controls. For example, hosts that rely on SSH access for recovery should also have documented administrative access and validation steps before and after a storage incident.

Expected output

You should have a clear warning threshold, a critical threshold, and a known escalation path for each important filesystem.

Validation

Test the alert logic against a non-production host or a maintenance window where you can safely observe the thresholds without risking service impact.

Common failure

The most common mistake is setting thresholds without accounting for filesystem behavior, package updates, log bursts, or database maintenance windows. Alerts should trigger before exhaustion, not after service failure.

What a healthy finished state looks like

A well-monitored Ubuntu host does not just show ?free space exists.? It has a clear picture of which filesystems are in use, which directories are growing, whether inode pressure is present, and whether any open files are preventing space from being reclaimed. You should be able to answer these questions quickly from the command line:

- Which filesystem is closest to capacity?
- Is the pressure caused by byte usage or inode usage?
- Which directories are responsible for the growth?

- Is the growth expected, temporary, or a leak?
- Do I need cleanup, a service restart, or a capacity change?

That is the practical value of command-line disk monitoring: faster diagnosis, better validation, and fewer surprises when usage trends upward. If you can compare df, du, and inode output consistently, you can monitor Ubuntu disk usage in a way that is operationally useful rather than merely descriptive.

Use this guidance together with SSH key-based authentication to connect the workflow with related operational context already available on the site.

Use this guidance together with Windows Server 2022 security baseline and SELinux access denials to connect the workflow with related operational context already available on the site.