Introduction to the Linux Operating System and Basics of Fundamental Operations

Shanghai Supercomputer Center　邹伟新

2026.1

Linux Basic

Learn the essential basics of Linux. Through terminal practice, you will master common commands, the directory structure, and file permissions, enabling you to efficiently complete daily tasks and rapidly get started with Linux—providing a solid foundation for future HPC practice.

Exploring Linux Distributions

1

Debian

Origin
Established in 1993 by Ian Murdock, emphasizing the principles of free software.

Distributions
Debian: Stable and community-driven, with slow but reliable updates (commonly used for servers).
Ubuntu (released 2004): Based on Debian, releasing a new version every 6 months, with an LTS (Long-Term Support) version every 2 years (e.g., 20.04, 22.04, 24.04).
Derivatives: Kubuntu, Xubuntu, Ubuntu Server, Ubuntu HPC.
Linux Mint: Targeted at general users, known for its strong usability.

2

Red hat

Origins
Red Hat Linux was created by Marc Ewing in 1994.

Representative Distributions
Red Hat Enterprise Linux (RHEL): Enterprise-grade, offering paid support with extremely high stability (widely used in finance, telecommunications, and HPC).
CentOS (2004–2021): A free rebuild of RHEL; it was once the standard for web servers.
CentOS Stream (2021–present): The upstream, rolling-release development branch for RHEL.
Rocky Linux / AlmaLinux: Community-driven, RHEL-compatible alternatives that emerged following the CentOS discontinuation.
Fedora: A cutting-edge community distribution sponsored by Red Hat, serving as a testing ground for new features.

3

SUSE

Origin
Originated in Germany in 1992, predating Red Hat.

Key Distributions
Open SUSE: The community edition, available as Tumbleweed (rolling release) and Leap (stable release).
SUSE Linux Enterprise Server (SLES): The enterprise edition, particularly popular in Europe and the HPC sector.

Basic commands

The following content compiles essential commands based on common usage scenarios. Master the skills of path navigation and directory structure creation using the cd and mkdir commands.

Permission

These permissions are divided among three categories of users:
Owner (u) : The user who created the file or has been designated as its owner.
Group (g) : A collection of users who share the same access rights to the file.
Others (o) : All other users on the system who are not the owner and do not belong to the group.
There are three types of permissions:
Read (r) : Allows viewing the contents of a file or listing the contents of a directory.
Write (w) : Allows modifying the contents of a file or creating/deleting files within a directory.
Execute (x) : Allows running a file as a program or script, or accessing a directory.

file permission 775 is as follows: The first digit 7: indicates that the owner of the file has read (4), write (2), and execute (1) permissions. Read permission (4) allows the owner to view the file's contents. Write permission (2) allows the owner to modify the file's contents. Execute permission (1) allows the owner to run the file as a program (if it is executable).

```
drwxr-xr-x   19 root root  3400 Oct 22 14:43 dev
drwxr-xr-x.  228 root root 20480 Oct 22 15:56 etc
drwxr-xr-x    3 root root  4096 Jan 12  2021 home
```

command

Basic commands

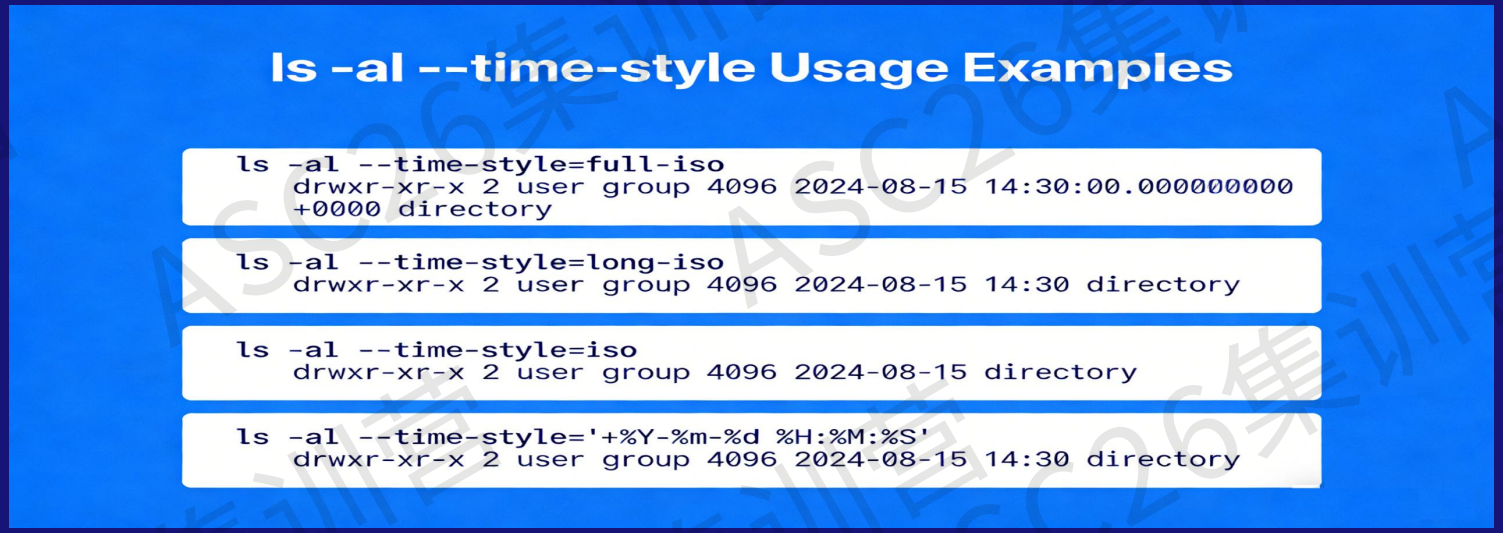
Linux commands serve as the core interface for interacting with Unix-like systems. Mastering common commands significantly enhances efficiency in system administration and troubleshooting. The following content compiles essential commands based on typical usage scenarios, covering fundamental functions such as file operations, directory management, and text processing.

Basic Command Syntax

Linux commands generally follow the format: command [options] [arguments].

For example : ls -l /home (displays the contents of the /home directory in long format).

advanced



Basic commands

Head & tail

Use the head and tail commands to extract data from the beginning and end of a file, with support for default behavior and custom line count control.

head

```
[root@[redacted] log]# head    boot.log-20251023
[  OK  ] Started Show Plymouth Boot Screen.
[  OK  ] Reached target Paths.
[  OK  ] Reached target Basic System.
[  OK  ] Found device Logical_Volume 3.
         Starting File System Check on /dev/...0-ba80-441c-a
[  OK  ] Started dracut initqueue hook.
[  OK  ] Reached target Remote File Systems (Pre).
[  OK  ] Reached target Remote File Systems.
[  OK  ] Started File System Check on /dev/d...510-ba80-441c-
         Mounting /sysroot...
```

tail

```
[root@[redacted]1 log]# tail  boot.log-20251023
         Starting Job spooling tools...
[  OK  ] Started Command Scheduler.
         Starting Command Scheduler...
[  OK  ] Started Login Service.
[  OK  ] Started Accounts Service.
[  OK  ] Started Dynamic System Tuning Daemon.
[  OK  ] Started Crash recovery kernel arming.
[  OK  ] Started /etc/rc.d/rc.local Compatibility.
         Starting GNOME Display Manager...
[  OK  ] Started GNOME Display Manager.
```

Basic commands

Use tail -f to monitor log files in real time, meeting the need for continuous tracking during system operations and maintenance.

updates in real time

Keep running and monitor file changes, immediately outputting any new lines to the terminal as they are written. 。

View the latest records.

`tail -f /var/log/syslog`

view real-time writes to system logs, facilitating quick issue identification

Application Management

rpm vs yum

RPM: The Foundation

Package Manager

A low-level tool for installing, uninstalling, and querying individual `rpm` packages.

Pros

Direct control, offline installation, lightweight.

Cons

No dependency resolution, no repo support.

Key Cmd

`rpm -i`, `rpm -e`, `rpm -qa`, `rpm -U`

YUM: The Convenience Layer

Yellowdog Updater Modified

A high-level tool built on RPM that simplifies management with automatic dependency resolution.

Pros

Auto dependency resolution, repo support, user-friendly.

Cons

Relies on RPM, typically internet-dependent.

Key Cmd

`yum install`, `yum remove`, `yum update`

Key Differences

Dependency Handling

RPM:Manual

YUM:Automatic

Repository Support

RPM:No built-in

YUM:Yes

User Friendliness

RPM:Complex

YUM:Simple

Level

RPM:Low-level

Backup command

tar

Commands

Create a compressed backup

```
tar -czvf backup.tar.gz /path/to/directory
```

-c: Create, -z: Gzip, -v: Verbose, -f: Filename

Extract a backup

```
tar -xzvf backup.tar.gz -C /path/to/restore
```

-x: Extract, -C: Specify target directory

List archive contents

```
tar -tzvf backup.tar.gz
```

Key Advantages

- **Preserves File Attributes**

  Retains permissions, ownership, and timestamps.

- **Flexible Compression**

  Works with gzip, bzip2, and xz for optimal size/speed.

- **Highly Versatile**

  Supports incremental backups and excluding files.

- **Cross-Platform Compatible**

  Standard tool across Unix, Linux, and macOS.

Linux User, Group, and Permission Management

Users

Every interaction with the Linux system is associated with a user account, which has unique permissions and access levels.

Key Commands:

- `useradd [options] username`
- `usermod [options] username`
- `userdel [options] username`
- `passwd [username]`

Groups

Groups are logical collections of users, designed to simplify permission management for multiple users.

Key Commands:

- `groupadd [options] groupname`
- `groupmod [options] groupname`
- `groupdel groupname`
- `gpasswd [options] groupname`

Permissions

Linux uses a granular permission system to control access to files and directories.

Permission Types:

- **Read (r):** View content
- **Write (w):** Modify content
- **Execute (x):** Run program

Key Commands:

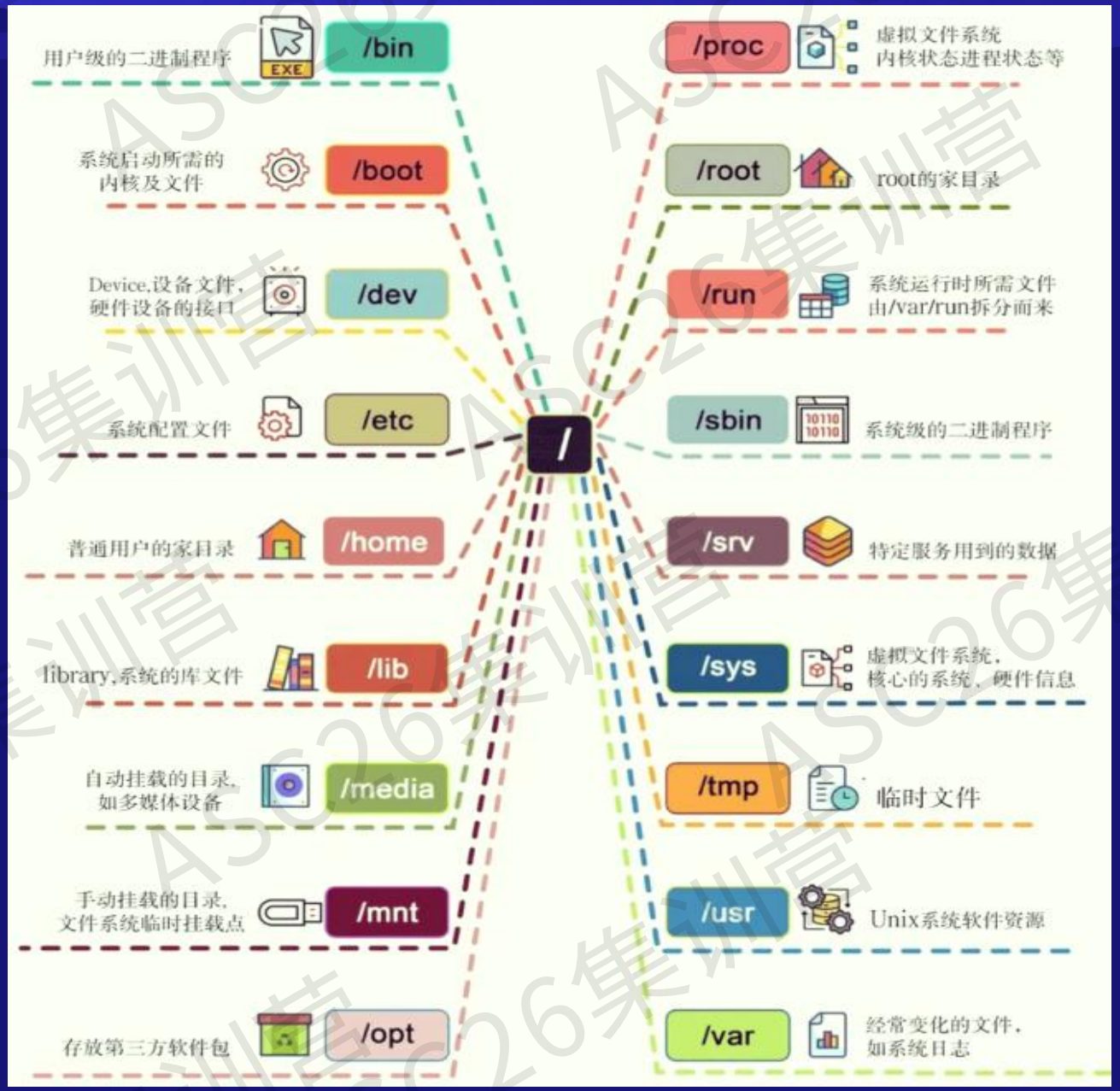
- `chmod [mode] filename`
- `chown [user:group] filename`

Filesystems and Directories

Tree-like directory structure

Linux employs a hierarchical directory structure where all directories and files originate from the root directory /. This unified architecture ensures clear resource management and precise path location.

Linux's directory structure may appear complex at first glance, but its logic is actually very clear—each directory has a distinct purpose and role. Mastering these fundamentals is like unblocking the vital energy channels in the system world; it empowers you to configure services, troubleshoot issues, and learn DevOps and security with far greater ease.



Filesystems and Directories

/proc

The filesystem is a pseudo-filesystem (also known as a virtual filesystem) that does not occupy actual disk space, but rather resides in memory. It provides an interface for user space to access kernel data structures and system information.

The /proc directory contains a vast collection of special files and subdirectories that provide various system information:

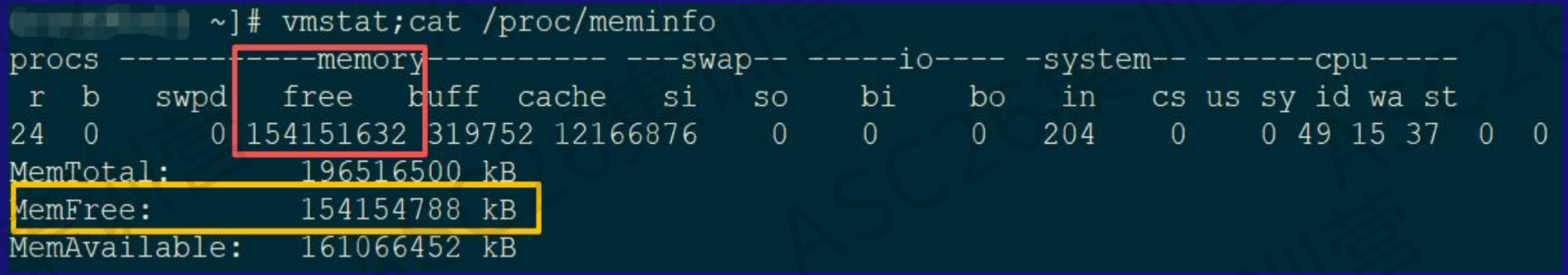
System Information Files:

/proc/cpuinfo: Contains detailed information about the CPU, such as model, frequency, and cache size.
/proc/meminfo: Displays memory usage statistics, including free memory, used physical memory, and swap space.
/proc/loadavg: Shows the system's average load.
/proc/version: Displays the kernel version, build time, and GCC version.
/proc/uptime: Shows the duration for which the system has been running.
/proc/cmdline: Displays the kernel command-line parameters used at boot.
/proc/filesystems: Lists the filesystems currently loaded in the system.
/proc/swaps: Shows the usage statistics of swap space.
/proc/partitions: Displays disk partition information.
/proc/bus/*: Contains information about bus devices.
/proc/sys: Allows users to dynamically modify kernel parameters.
/proc/[PID]: A directory named after a Process ID (PID) containing detailed information about that specific process.

Filesystems and Directories

/proc

In Linux systems, commands such as free, iostat, vmstat, and sar are essential tools for monitoring system performance. Although they focus on different resource layers, together they form a comprehensive performance analysis framework. These commands are not located directly within the /proc directory; rather, they are standalone executable utilities. However, some of their data sources are linked to the /proc filesystem (for instance, vmstat and mpstat read from /proc/stat).。



Filesystems and Directories

/proc Clever Use Cases——— Recovering Deleted Files

If a file is deleted using rm but is still held open by a process (such as a log process), the disk space will not be immediately released. In fact, the file data still resides on the disk.

```
[root@ ~]# lsof | grep deleted

pulseaudi   2948                    6u      REG              0,4   67108864        38481 /memfd:pulseaudio (deleted)
null-sink   2948   2951             6u      REG              0,4   67108864        38481 /memfd:pulseaudio (deleted)
top        54338                    0u      CHR            136,8        0t0           11 /dev/pts/8 (deleted)
top        54338                    1u      CHR            136,8        0t0           11 /dev/pts/8 (deleted)
top        54338                    3u      CHR            136,8        0t0           11 /dev/pts/8 (deleted)
python3.1  63909                    0u      CHR           136,11        0t0           14 /dev/pts/11 (deleted)
python3.1  63909                    1u      CHR           136,11        0t0           14 /dev/pts/11 (deleted)
python3.1  63909                    2u      CHR           136,11        0t0           14 /dev/pts/11 (deleted)
squid      75384                root    3u      REG              8,3      1785      5505260 /var/log/squid/cache.log-20251116 (deleted)
squid      75387               squid    3u      REG              8,3      1785      5505260 /var/log/squid/cache.log-20251116 (deleted)
[root@ ~]#
[root@ ~]# find /proc/75384/fd -ls | grep dele
2304801080    0 lrwx------   1 root     squid          64 Jan 20 09:10 /proc/75384/fd/3 -> /var/log/squid/cache.log-20251116\ (deleted)
You have new mail in /var/spool/mail/root
[root@ ~]# cp /proc/75384/fd/3 /tmp/20250127
[root@ ~]# ls -al /tmp/20250127
-rw-r----- 1 root root 1785 Jan 20 09:12 /tmp/20250127
You have new mail in /var/spool/mail/root
[root@ ~]# more  /tmp/20250127
2025/11/11 14:00:09 kid1| Set Current Directory to /var/spool/squid
2025/11/11 14:00:09 kid1| Starting Squid Cache version 3.5.20 for x86_64-redhat-linux-gnu...
2025/11/11 14:00:09 kid1| Service Name: squid
2025/11/11 14:00:09 kid1| Process ID 75387
2025/11/11 14:00:09 kid1| Process Roles: worker
```

Filesystems and Directories

/etc/logrotate

logrotate is a log file management utility in Linux systems. It is used to automatically rotate log files to control their size and retention duration. Additionally, it can compress and delete old log files.
The main functions of logrotate include the following aspects:
1,Control Log File Size: Log files continuously grow over time. Without management, they may consume excessive disk space. logrotate periodically rotates log files to limit the size of individual files.

2,Retain Log History: Besides controlling file size, logrotate can be configured to keep a specific number of log files or for a set duration. When these limits are reached, the oldest log files are deleted or compressed, thus maintaining a history of logs.

3,Prevent Performance Issues: Excessively large log files can slow down read and write operations, potentially impacting system performance. Regular rotation prevents files from becoming too large, ensuring the system runs smoothly.

4,Compress Old Log Files: logrotate can compress archived log files to save disk space. Compressed files can be further archived or transferred for subsequent analysis.

5,Automated Management: logrotate is configured via configuration files, making log management fully automated. You can customize rotation conditions and behaviors to suit specific requirements.

```
[root@ etc]# cat logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
monthly

# keep 4 weeks worth of backlogs
rotate 6

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
dateext

# uncomment this if you want your log files compressed
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp and btmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    create 0664 root utmp
        minsize 1M
    rotate 6
}
```

```
[root@ logrotate.d]# cat /etc/logrotate.d/tomcat
/var/log/tomcat/catalina.out {
    copytruncate
    weekly
    rotate 52
    compress
    missingok
    create 0644 tomcat tomcat
}
```

Linux Shell

The Linux shell is a command-line interface that acts as an intermediary between the user and the operating system's kernel. It interprets commands entered by the user, executes them, and returns the results—enabling direct control over the system's resources, files, processes, and services.
Unlike graphical user interfaces (GUIs), the shell operates through text-based input, making it powerful, scriptable, and highly efficient—especially for automation, system administration, and remote server management.

Common Types of Shells

Bash (Bourne Again Shell): The default shell on most Linux distributions; feature-rich and widely used.
Zsh (Z Shell): Offers advanced customization, auto-completion, and theming (popularized by frameworks like Oh My Zsh).
Fish (Friendly Interactive Shell): Designed for usability and interactive features with syntax highlighting.
Dash, Ksh, Tcsh: Other shells used in specific environments or for compatibility.

Key Features

Command execution: Run programs, scripts, and utilities.
Scripting: Automate tasks using shell scripts (.sh files).
Pipelines & redirection: Chain commands (|) and control input/output (>, >>, <).
Environment control: Manage variables, paths, and session settings. Job control: Run, pause, and manage background/foreground processes

Shell - grep

grep Command: From Basic to Advanced

Basic Usage

Search for a String

Searches for the exact string "error" in file.txt.

```
grep "error" file.txt
```

Case-Insensitive Search

Ignores case, matching "Error", "ERROR", etc.

```
grep -i "error" file.txt
```

Show Line Numbers

Displays the line numbers of matching lines.

```
grep -n "warning" file.txt
```

Advanced Usage

Regular Expressions

Lines starting with "Error:" (using ^ anchor).

```
grep "^Error:" error.log
```

Recursive Search

Search in current directory and subdirectories.

```
grep -r "TODO" .
```

Pipe with Other Commands

Combine with ps to find nginx processes.

```
ps aux | grep nginx
```

grep is essential for developers & sysadmins. Master it to save hours of manual searching.

Shell - awk

awk Command: From Basic to Advanced

Basic Commands

Print Specific Columns

```
awk '{print $1, $4}' data.txt
```

Prints the 1st and 4th fields of each line.

Custom Delimiter

```
awk -F, '{print $2, $3}' data.log
```

Uses `-F,` to specify comma as separator.

Pattern Matching

```
awk '/error/ {print $0}' log.txt
```

Prints lines containing the pattern 'error'.

Advanced Usage

Associative Arrays

```
awk '{count[$1]++} END {...}' access.log
```

Counts occurrences (e.g., IP addresses).

Built-in Functions

```
awk '{print toupper($0)}' text.txt
```

String manipulation (e.g., uppercase).

Conditionals & Loops

```
awk '{if ($3 > 90) print ...}' grades.txt
```

Filter and process based on conditions.

Shell - sed

awk Command: From Basic to Advanced

Basic Usage

Substitute Text

Replace first occurrence of "old" with "new".

```
sed 's/old/new/' file.txt
```

Global Substitution

Replace all occurrences of "old" with "new".

```
sed 's/old/new/g' file.txt
```

Delete Lines

Delete lines that match the pattern.

```
sed '/pattern/d' file.txt
```

Advanced Usage

Address Ranges

Apply command to lines 5 through 10.

```
sed '5,10s/old/new/g' file.txt
```

Append/Insert Text

Append (a) or Insert (i) text around matches.

```
sed '/pattern/a\text' file.txt
```

Regular Expression Groups

Reference groups with \1, \2, etc.

```
sed 's/\(.*\) \(.*\)/\2, \1/'
```

Mastering sed significantly boosts productivity in text processing and shell scripting.

Shell - programming

Linux Shell Programming: if for while

if

Used for conditional execution, allowing the script to make decisions based on test outcomes.

```
if [ condition ]; then
# Code if true
elif [ cond ]; then
# Code if false
else
# Fallback
fi
```

Example:

```
read -p "Num: " n
if [ $n -gt 10 ]; then echo "Greater";
fi
```

for

Iterates over a sequence of items (words, numbers, files) and executes code for each.

```
for var in item1 item2 ...; do
# Code for each item
done
```

Example:

```
for file in *.txt; do
echo "Processing $file"
done
```

while

Executes code as long as a specified condition remains true, useful for reading input.

```
while [ condition ]; do
# Code while true
done
```

Example:

```
count=1
while [ $count -le 5 ]; do
echo "Count: $count"; ((count++))
done
```

Linux Shell

System monitoring script: Monitor real-time data and generate status reports

01 02 03 04

By collecting basic resource data, such as top and iostat, and using commands like awk , sed to extract key values, Generate system basic report. the monitored data is written into the log for backup purposes.

05 06 07 08

Integrate notification functions, send reports regularly, allowing system administrators to be informed promptly. Issue warnings when thresholds are exceeded. Optimize scripts for long-term monitoring to ensure service quality

Crontab Command in Linux: The Ultimate Guide

Syntax

```
* * * * *  command_to_execute
```

- **Minute (0-59):**The minute when the job runs.
- **Hour (0-23):**The hour when the job runs.
- **Day of Month (1-31):**The day of the month.
- **Month (1-12):**The month of the year.
- **Day of Week (0-6):**Day of week (0 or 7 =Sunday).

Special Characters

- *: Wildcard, means "every".
- ,: Specifies a list of values (e.g., 1,3,5).
- - /: Specifies range or step (e.g., 1-5, */15).

Commands

- **crontab -e**: Edit crontab file.
- **crontab -l**: List crontab entries.
- **crontab -r**: Remove crontab file.
- **crontab -u [user] -e**: Edit another user's crontab.

Examples

- **5 * * * ***: Run hourly at minute 5.
- **0 0 * * ***: Run daily at midnight.
- **0 2 * * 0**: Run every Sunday at 2 AM.

Pro Tip: Use absolute paths and redirect output to a log file for debugging.

Linux security

Linux PAM Security: Hardening Your Authentication

What is PAM?

PAM (Pluggable Authentication Modules) is a flexible framework that decouples authentication from applications, allowing centralized management of how users are authenticated for services like `sshd`, `login`, and `sudo`.

Core Concepts

Modules

Independent `.so` files performing specific tasks (e.g., `pam_unix.so`).

Configuration Files

Located in `/etc/pam.d/` for each service's auth stack.

Example: **system-auth-ac**

```
password requisite pam_pwquality.so retry=3 minlen=12 dcredit=-1 ucredit=-1 lcredit=-1 ocredit=-1
```

Key Hardening Strategies

Enforce Strong Passwords

Use `pam_pwquality.so` for complexity and `remember=5` to prevent reuse.

Limit Failed Logins

Lock accounts with `pam_faillock.so` after 5 attempts for 15 minutes.

Restrict User Access

Control login sources with `pam_access.so` and `/etc/security/access.conf`.

Linux security

Policies & Configuration

User Authentication

- SSH Key Authentication
- Strong Password Policies
- Multi-Factor Authentication

Network Security

- Firewall (UFW/Iptables)
- Port Hardening & Service Minimization
- Network Segmentation

System Hardening

- Kernel Parameter Tuning
- Secure Filesystems
- System vulnerability patch(update)

Data Protection

- Secure Backup Strategies (rsync,tar)
- File-level Deletion (shred)

Monitoring & Auditing

- Centralized Log Management
- Intrusion Detection
- Regular System Audits

Continuous Security

- Automated Updates & Patching
- Security Awareness Training
- Incident Response Planning

Advanced： Linux & HPC

Linux, with its open-source, stable, and customizable features, perfectly meets the stringent requirements of HPC for the underlying operating system. It not only supports global supercomputers but also deeply integrates into the operation, maintenance, and management ecosystem of HPC. The two complement each other, forming the cornerstone of modern high performance computing.

Why choose Linux

Four core advantages of Linux making it the preferred choice for HPC

01

Freedom and open

- Linux is a free and open-source Unix-like operating system, allowing users to freely access, modify, and distribute its source code, providing infinite possibilities for customization needs in the HPC field 。

02

Stability and security

- Linux is renowned for its exceptional stability and robust security. Its kernel design is robust and updated promptly, ensuring an uninterrupted and stable operating environment for HPC clusters.

03

free

- Compared to proprietary systems, Linux is free of charge, significantly reducing the cost of building and operating large-scale HPC systems. At the same time, the abundant ecosystem of free and open-source software further saves software investment 。

04

powerful group

- With a vast community of developers worldwide, who continuously contribute code, fix bugs, and develop new features, the continuous progress and vitality of Linux technology are ensured 。

HPC& Linux

The best choice

1 Integration with open-source systems

HPC requires continuously optimized software. The open-source nature of Linux allows researchers and developers to directly delve into the operating system and software kernel for customization, optimization, and integration.

2 Excellent parallel support

The Linux kernel itself supports multitasking, multithreading, and symmetric multiprocessing (SMP), providing a solid foundation for parallel computing frameworks such as MPI and OpenMPI 。

3 hardware support

Linux supports various hardware architectures ranging from x86 to ARM, enabling flexible adaptation to diverse computing nodes, storage, and network devices in the HPC field

4 Mature cluster management

The mainstream cluster management systems (such as LSF, SLURM, PBS) and resource schedulers are all developed based on Linux, achieving efficient management and scheduling of computing resources.

Linux applications in HPC

The specific applications of Linux in high-performance computing, including core components and technologies such as resource scheduling, parallel computing frameworks, and system monitoring.

HPC Job Scheduling Systems

Core Functions, Mainstream Software & Scheduling Policies

Core Functions

Job Lifecycle Management
Submission, queuing, execution, termination.

Resource Management
Allocate CPUs, GPUs, memory, I/O.

Queue Management
Maintain queues and prioritize jobs.

Monitoring & Reporting
Track progress, utilization, health.

Policy Enforcement
Fair share, access control, accounting.

Mainstream Software

Slurm
Open-source standard, highly scalable.

LSF (IBM)
Enterprise-grade, high reliability.

PBS
Flexible policy engine, workflows.

Scheduling Policies

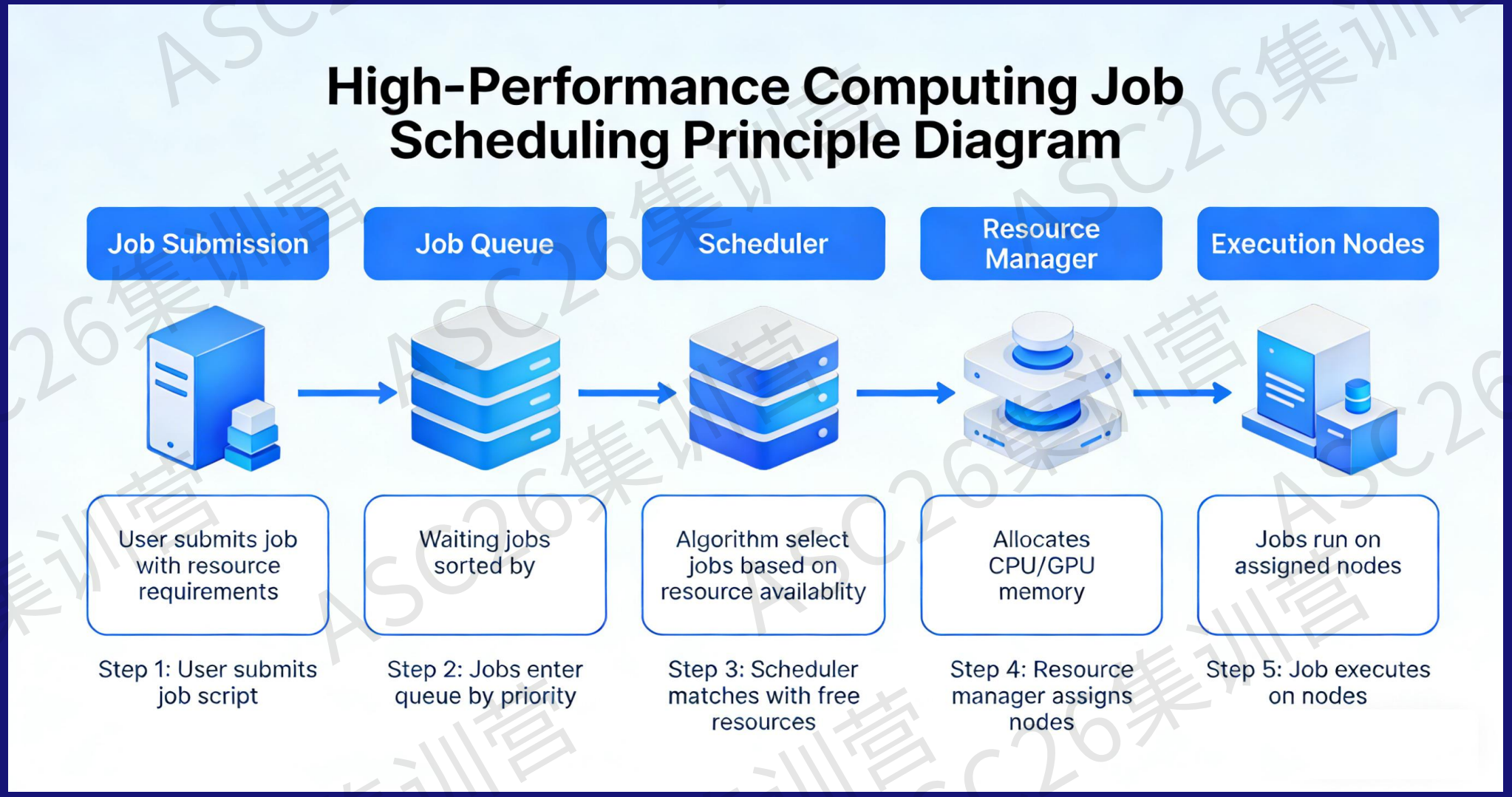
FCFS
Simple, fair, but may be inefficient.

Preempt
High-priority tasks preempt low-priority job.

Fair-Share
Ensures equitable access over time.

Backfilling
Maximizes cluster utilization.

HPC Job Scheduling Systems



HPC System Administration

System management is the cornerstone of High-Performance Computing (HPC).

It ensures the efficient operation of HPC clusters, which consist of computing nodes, high-speed networks, file systems, and scheduling systems.

maintenance team is responsible for managing job scheduling, machine health, fault maintenance, and optimizing hardware performance. Without their support, HPC systems will face issues such as low resource utilization and research delays

Hardware Management: The Foundation of HPC

Computing Nodes

Multi-core CPUs & accelerators (GPUs) for parallel processing tasks.

High-Speed Interconnect

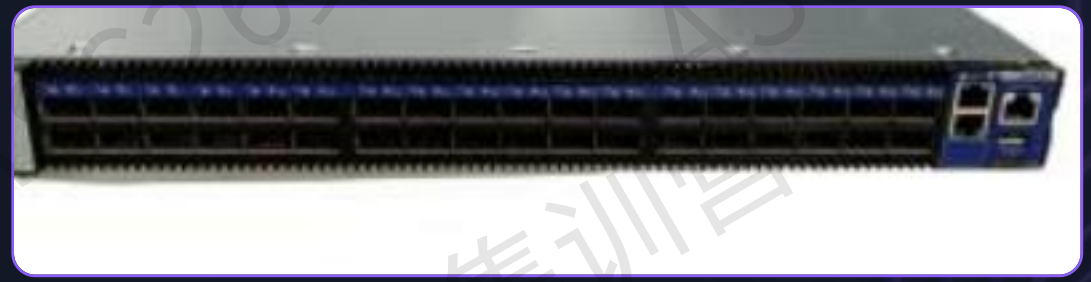
Low-latency networks (InfiniBand, OPA) for efficient node communication.

Storage Systems

Scalable solutions (Lustre, GPFS) to handle massive I/O requirements.

Power & Cooling

Compute Nodes



High-Speed InfiniBand Interconnect

Software Management

Operating System

A lightweight, optimized Linux distro (e.g., CentOS, RHEL) providing a stable foundation.

Device Drivers

Low-level software enabling OS-hardware communication, critical for GPUs and high-speed networks.

Middleware & Libraries

MPI, compilers (GCC, Intel), and math libraries (BLAS, LAPACK) form the application backbone.

Application Management

Tools for deploying, updating, and maintaining user apps across the cluster.

Resource Management: Maximizing Utilization & Efficiency

Cluster Provisioning

Automated tools for efficient deployment and configuration of OS and software across nodes.

Monitoring & Telemetry

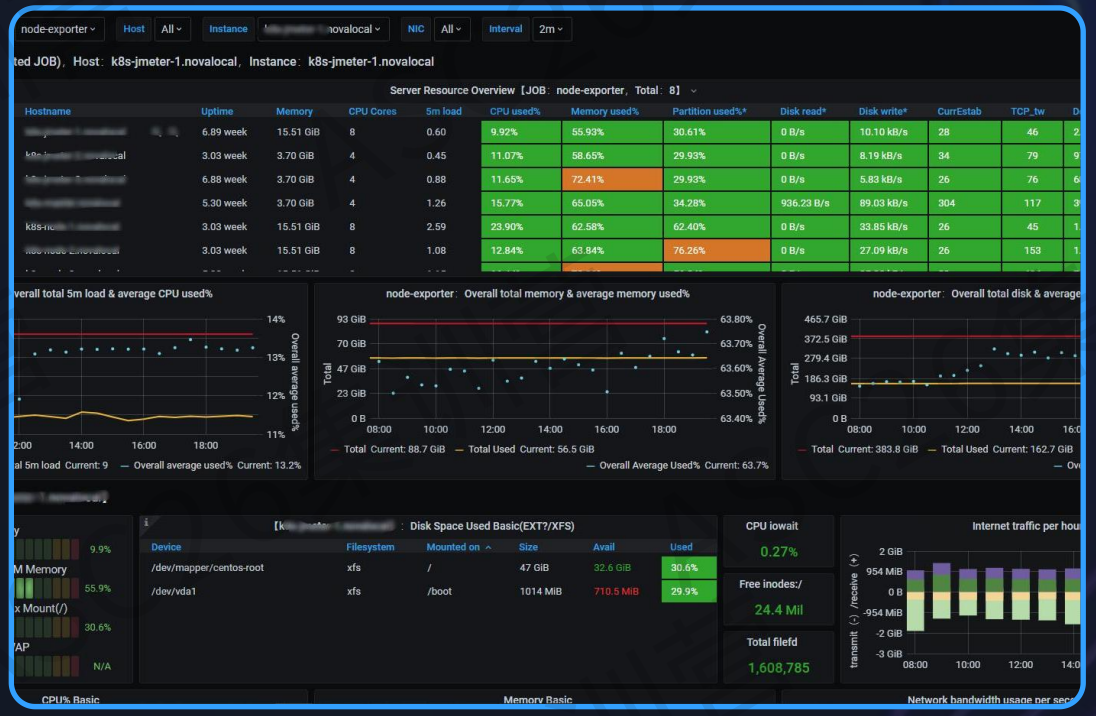
Real-time tracking of system health, resource utilization, and application performance.

Performance Optimization

Identify bottlenecks and optimize resource allocation using profiling tools.

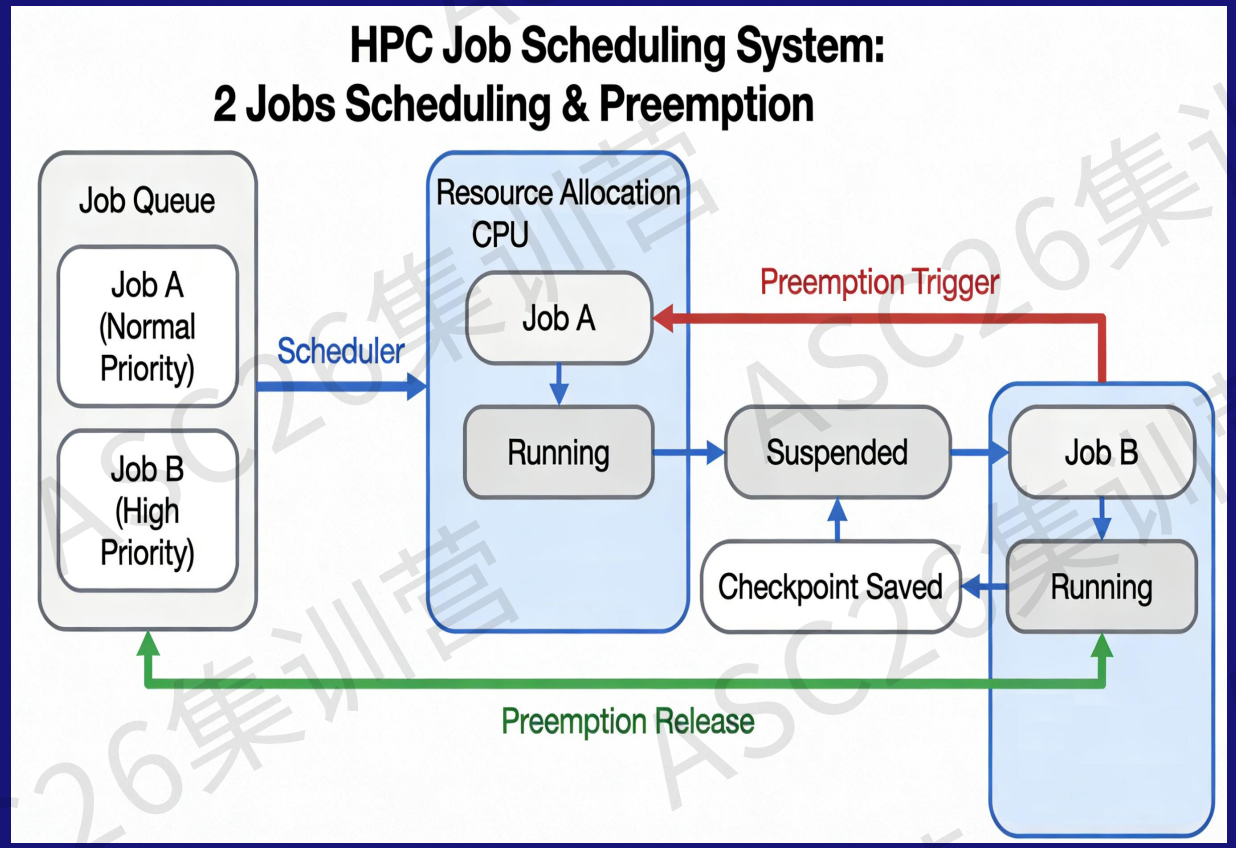
Dynamic Resource Allocation

Flexible scaling based on demand to ensure critical workloads get resources.



Real-time Resource Monitoring Dashboard

Job Management: Scheduling the Workload



Job Schedulers

Core software (Slurm, PBS) managing job submission, queuing, and execution.

Queue Management

Organizing jobs by priority or resources for efficient utilization.

Workload Orchestration

Managing complex interdependent workflows to run in the correct order.

Fairshare & Policies

Allocating resources fairly to prevent monopolization of the cluster.

Supercomputer Management

Core Roles

System Monitoring & Maintenance
7x24 oversight to ensure stability and maximize uptime.

Performance Optimization
Tuning hardware/software for peak computational efficiency.

Security & Compliance
Protecting sensitive data and critical infrastructure.

User Support & Enablement
Assisting users to effectively leverage supercomputer capabilities.

Key Benefits

Maximized ROI
Extends hardware lifespan and increases system utilization.

Accelerated Innovation
Shortens the path to scientific discovery via faster simulations.

Reduced Operational Costs
Prevents costly downtime and emergency repairs.

Enhanced Reliability & Reputation
Attracts top talent and fosters collaboration.

thanks