PS1

Modifying the PS1 variable in the .bashrc file affects your command prompt in the bash shell. Let's break this down:

1. PS1 stands for "Prompt String 1". It's the primary prompt string that bash uses.
2. .bashrc is a script file that's executed whenever a new interactive shell session starts.

Modifying PS1 allows you to customize how your command prompt looks and what information it displays.

Real-world applications:

1. User Experience: Customizing the prompt can make the terminal more user-friendly and informative.
2. System Information: You can include system info like current directory, username, hostname, or time in the prompt.
3. Git Integration: Many developers modify PS1 to show git branch information in their prompt.
4. Server Identification: In multi-server environments, admins often customize prompts to quickly identify which server they're on.
5. Error Reporting: You can include the exit status of the last command in your prompt.
6. Performance Monitoring: Some users include system load or other performance metrics in their prompt.
7. Security: You might include a visual indicator if you're running with elevated privileges.

For example, a common modification might be:

PS1='\u@\h:\w\$ '

This would show username, hostname, and current directory.

Remember, while customizing PS1 can be very useful, overly complex prompts can slow down your terminal, especially in large directories or with slow network connections. It's about finding the right balance for your needs!

sudo apt update

1. `sudo`: Run the command with superuser privileges
2. `apt`: Advanced Package Tool, the package management system
3. `update`: Refresh the local package index

Real-world applications:

1) System Maintenance: Regular updates ensure your system has the latest information about available packages. It's often the first step before installing or upgrading software.

2) Security: Keeping your package lists updated is crucial for identifying and installing security updates.It helps protect against vulnerabilities in outdated software.
3) Automation:
4) This command is often included in automated scripts for system maintenance.
5) Useful in DevOps practices for keeping systems up-to-date.
6) Troubleshooting:
7) When facing issues with package installation, running this command can often help resolve problems.
8) Pre-installation Checks:
9) Before installing new software, it's good practice to run this command to ensure you're working with the latest package information.
10) System Administration:
11) System admins often run this command as part of their regular maintenance routines.
12) Software Development:
13) Developers working in Linux environments use this to ensure their development tools are up-to-date.

Remember, `sudo apt update` only updates the package lists. To actually upgrade the packages, you'd need to follow it with `sudo apt upgrade`. In production environments, it's important to test updates in a staging environment before applying them to production systems to avoid potential compatibility issues.

sudo apt search vim- This command will search for the vim package and related packages in the repository. It will list the package names along with short descriptions of each.

apt show vim- If you specifically want to see more details about the vim package (e.g., version, dependencies, etc.)

sudo apt install nginx To install NGINX, use the correct package name in lowercase:

NGINX (pronounced “engine x”) is an open-source web server software that can be used as a reverse proxy, load balancer, mail proxy, and HTTP cache. It was created by Russian developer Igor Sysoev and publicly released in 2004.

An alias in Linux (and other Unix-like systems) is a shortcut or alternative name for a command or a set of commands. It allows you to create custom, shorter commands that

stand in for longer or more complex ones. Aliases help you save time and effort by typing less, especially for commands you use frequently.

mkfifo [OPTIONS] NAME

The mkfifo command in Unix/Linux is used to create **named pipes**, also known as **FIFOs (First In, First Out)**. A FIFO is a special type of file that acts as a pipe, allowing data to be written by one process and read by another in a synchronized manner.

How FIFOs Work

- **Named pipes (FIFOs)** allow two processes to communicate with each other through the file system.
- One process writes data to the FIFO, while another process reads from it.
- The data is read in the same order it was written (First In, First Out).
- A named pipe exists as a file in the file system, unlike regular pipes (|), which are temporary.

The -n option in the nice command specifies the **nice value** to be assigned to the process being started. The nice value determines the **priority** of the process in relation to other processes.

Understanding nice Values:

- **Range**: The nice value can range from -20 (highest priority) to 19 (lowest priority).
  - **Lower values** mean **higher priority**, which gives the process more CPU time.
  - **Higher values** mean **lower priority**, which gives the process less CPU time compared to other processes.
- By default, if no -n value is specified, the nice value is 0.

-n in Action:

- **nice -n 10**: This assigns a **nice value of 10** to the process, meaning it's running with lower priority than most processes (since 0 is the default).

  Example:

  bash

  Copy code

  nice -n 10 sleep 300 &

  This command starts the sleep 300 process with a nice value of 10, running in the background. Since the nice value is positive, this process will receive less CPU time compared to higher-priority processes.

The renice command in Unix/Linux is used to **change the priority (nice value) of a running process**. This allows you to adjust the scheduling priority of processes that are already running, which helps manage CPU usage on a system.

sudo renice 10 -p 1234

The cat /proc/meminfo command is used to display information about the system's memory usage on Linux. It reads the content of the /proc/meminfo file, which contains details about the memory usage of the system such as total memory, available memory, buffers, cache size, and more.

The free -h command in Linux displays information about the system's memory usage in a human-readable format (with units like MB, GB, etc.). It shows the total amount of memory, the amount used, free, shared, buffer/cache, and available memory, along with swap usage.

vmstat

When you run vmstat without any arguments, it gives a single snapshot of the system's resource usage since the last reboot. This output is a static view of the system's current state

vmstat 2

When you run vmstat with a number (like 2), it continuously reports system performance statistics at regular intervals (in seconds). In this case, 2 means that the command will report performance metrics every 2 seconds.

What is Swap Memory in Linux?

Swap memory in Linux is a space on the disk that is used when the system's **RAM** (Random Access Memory) is fully utilized. When the system runs out of physical memory (RAM), inactive pages in memory can be moved to swap space, freeing up RAM for more active processes. This helps the system continue running efficiently even when memory demands exceed the available RAM.

However, since accessing data from disk is much slower than accessing data from RAM, excessive use of swap can lead to performance degradation. Ideally, swap is used as a backup when RAM is insufficient, not as a primary resource.

Key Points About Swap Memory:

1. **Location**: Swap can be a **swap partition** or a **swap file** on the disk.
2. **Usage**: Swap is used to hold idle data that isn't actively being used by the system, freeing up faster RAM for tasks that are more critical.
3. **Speed**: Accessing data in swap is significantly slower than accessing data in RAM.

4. **Kernel's Role**: The Linux kernel decides what gets swapped out to disk, based on how active the data is.

Checking Swap Memory Usage

You can check the swap usage in Linux using several commands:

1. **Using free -h**:

free -h

	total	used	free	shared	buff/cache	available
Mem:	15Gi	3.2Gi	8.5Gi	250Mi	3.4Gi	11Gi
Swap:	2.0Gi	500Mi	1.5Gi			

This shows that there is 2 GB of total swap space, with 500 MB currently in use and 1.5 GB available.

2. **Using swapon --show**: This command lists active swap areas.

swapon --show

Copy code

NAME TYPE SIZE USED PRIO

/swapfile file 2GiB 500M -2

It shows the location, type, size, and usage of the swap file.

3. **Using vmstat**: The vmstat command provides detailed memory information, including swap activity.

vmstat

procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----

r b swpd free buff cache si so bi bo in cs us sy id wa st

1 0 1000 120000 30000 80000 0 0 0 0 10 50 5 1 92 2 0

   - si (swap in): Amount of memory swapped in from disk to RAM (in kilobytes per second).
   - so (swap out): Amount of memory swapped out from RAM to disk.

Managing Swap Space

1. **Enabling a Swap File**: If you don't have a swap partition, you can create a swap file. Here's how:

sudo fallocate -l 1G /swapfile

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

This creates a 1 GB swap file. To make this permanent, add the following line to /etc/fstab:

/swapfile swap swap defaults 0 0

2. **Turning Off Swap**: You can temporarily disable swap with the following command:

   sudo swapoff -a

To turn it back on:

sudo swapon -a

3. **Adjusting Swap Usage: swappiness**: The **swappiness** parameter controls how aggressively the Linux kernel swaps data from RAM to swap. The value ranges from 0 to 100:
   - A lower value (closer to 0) means the system avoids swapping as much as possible, keeping more data in RAM.
   - A higher value (closer to 100) means the system will swap more aggressively.

Check the current swappiness value:

cat /proc/sys/vm/swappiness

To temporarily set swappiness to 10 (less aggressive swapping):

sudo sysctl vm.swappiness=10

To make this permanent, edit /etc/sysctl.conf and add:

vm.swappiness=10

When Swap Is Used:

- **Low RAM availability**: When physical memory is exhausted, swap helps maintain system stability by providing extra virtual memory.
- **Hibernation**: When a system hibernates, the entire contents of RAM are written to swap space so that it can be restored upon waking up.

vm and swap memory is same in linux?

No, **VM (Virtual Memory)** and **Swap Memory** are not the same in Linux, though they are closely related and often used together. Let me explain the difference and the relationship between them.

1. Virtual Memory (VM):

- **Virtual Memory** is a **logical memory management** technique that allows processes to use more memory than what is physically available on the system.
- It combines **physical RAM** with a portion of the hard disk (swap space) to create an illusion of a larger contiguous memory space.
- Every process sees a large, continuous block of memory, regardless of whether it is actually loaded in the RAM or on disk (swap). The operating system handles moving data between RAM and disk transparently.

How Virtual Memory Works:

- When a process needs memory, the operating system allocates a portion of virtual memory to it.
- The **MMU (Memory Management Unit)** translates the virtual memory addresses to physical memory addresses.
- If RAM becomes full, **less active/inactive pages** of memory are moved to swap (disk) to free up RAM for more active processes. This process is called **paging**.

2. Swap Memory:

- **Swap memory** (or swap space) is a specific portion of the hard disk that acts as **virtual memory's backup** for RAM. It holds memory pages that are not actively being used by processes.
- It is slower than physical RAM because accessing data from a disk is much slower than accessing data from RAM.
- Swap is used when the system runs out of physical RAM. It moves **inactive pages** of memory from RAM to swap space on the disk to free up RAM for processes that need it.

Differences Between Virtual Memory and Swap Memory:

Feature	Virtual Memory (VM)	Swap Memory
Definition	A technique that allows processes to use more memory than physically available by abstracting memory addresses.	A portion of disk space used to temporarily store inactive data from RAM when it's full.
Storage Medium	Uses both **physical RAM** and **swap** (disk) for managing memory.	Uses only **disk** space for storing memory pages swapped out from RAM.
Speed	Depends on whether memory pages are in RAM (fast) or swapped to disk (slow).	Much **slower** than RAM because it resides on a disk.

Feature	Virtual Memory (VM)	Swap Memory
Purpose	To provide a large logical memory space for processes and optimize memory usage.	To extend the available memory when RAM is fully utilized.
Control Mechanism	Managed by the operating system, specifically by the **MMU** and paging system.	Directly used by the kernel when there is **memory pressure** (low RAM).

Relationship:

- **Swap** is a part of the **virtual memory** system. It extends virtual memory by allowing data to be temporarily moved from RAM to disk when RAM is full. Virtual memory uses both **physical RAM** and **swap** space as storage for processes, making it seem like the system has more memory than it physically does.

Example:

- If your system has 8 GB of RAM and 2 GB of swap, the total **virtual memory** available to the system is **10 GB** (RAM + swap).
- When the system is actively using only 6 GB of memory, all processes fit in RAM, and swap isn't used.
- If more memory is required (say 9 GB is needed), 8 GB will be in RAM, and 1 GB of less-used data will be moved to **swap** to free up RAM for active processes.

Downsides of Relying on Swap:

- When the system starts using swap heavily (meaning it has to move data from RAM to disk frequently), the system performance can suffer greatly. This is often referred to as **"swapping"** or **"thrashing"**, where the system spends more time moving data between RAM and swap than executing tasks.

Conclusion:

- **Virtual Memory** is a combination of **physical RAM** and **swap space** that provides a larger, unified memory space for processes.
- **Swap memory** is the portion of the disk used by the virtual memory system when physical RAM is full.

While both are part of Linux's memory management, **swap** is just one element of the broader **virtual memory** system.

Logical Volume Manager

Logical Volume Manager (LVM) is a system for managing disk storage in a more flexible manner than traditional partitioning. It allows for dynamic resizing, merging, and snapshot creation of storage volumes, which makes it ideal for environments where storage needs change frequently.

Key Concepts in LVM:

1. **Physical Volume (PV)**: These are the actual storage devices (like hard drives or partitions) that are used in LVM. You can create a physical volume from a partition or an entire disk.

Example: /dev/sda1, /dev/sdb, etc.

2. **Volume Group (VG)**: A pool of storage that consists of one or more physical volumes. You can add or remove physical volumes to a volume group to expand or shrink storage dynamically.

Example: vg01, vg_data

3. **Logical Volume (LV)**: These are the partitions created from the storage pool in a volume group. Logical volumes can be resized (extended or reduced) as needed, making them much more flexible than traditional partitions.

Example: /dev/vg01/lv_root, /dev/vg_data/lv_home

4. **Physical Extents (PE) and Logical Extents (LE)**: LVM divides physical volumes into fixed-size chunks called physical extents (PE). Logical volumes are made up of these extents, called logical extents (LE).

Key Benefits of LVM:

- **Dynamic resizing**: LVM allows you to resize logical volumes without unmounting them (in most cases), allowing for seamless storage expansion or contraction.
- **Snapshots**: You can create snapshots of logical volumes, which can be used for backup purposes or to revert to a previous state.
- **Better disk space management**: By pooling physical volumes into a volume group, you can use available storage more efficiently.

Basic LVM Commands:

1. **Creating a Physical Volume:**

```
pvcreate /dev/sdb
```

2. **Creating a Volume Group:**

```
vgcreate vg_data /dev/sdb
```

3. **Creating a Logical Volume:**

lvcreate -L 100G -n lv_home vg_data

4. **Extending a Logical Volume:**

   lvextend -L +50G /dev/vg_data/lv_home

5. **Reducing a Logical Volume:** Be cautious when reducing volumes as it may cause data loss:

   lvreduce -L -20G /dev/vg_data/lv_home

6. **Creating a Snapshot:**

lvcreate --size 10G --snapshot --name snap_lv_home /dev/vg_data/lv_home

LVM Setup Example:

1. Create a physical volume from a disk:

   pvcreate /dev/sdb

2. Create a volume group from the physical volume:

   vgcreate vg_data /dev/sdb

3. Create a logical volume for storing data:

   lvcreate -L 50G -n lv_data vg_data

4. Format the logical volume:

   mkfs.ext4 /dev/vg_data/lv_data

5. Mount the logical volume:

   mount /dev/vg_data/lv_data /mnt/data

LVM provides much greater flexibility and control over disk storage, making it highly beneficial for large or dynamic environments.

$(...): This syntax tells the shell to execute the command inside the parentheses and substitute it with the output.

The 2> part of 2>/dev/null is used for **redirecting error messages** (specifically, the "standard error" stream) in shell scripting.

Explanation:

- **2>**: Redirects the **standard error** (stderr) stream, which has a file descriptor number of 2.
- **/dev/null**: A special file that discards any input sent to it. By redirecting errors to /dev/null, you effectively "hide" or ignore any error messages.

Why Use 2>/dev/null?

When you run a command like ls /home/user/test_temp_dir/*.tmp, if no .tmp files are found, it might output an error message such as No such file or directory. Using 2>/dev/null suppresses this message, keeping the output cleaner.

Example:

ls /nonexistent-directory 2>/dev/null

In this example, if the directory does not exist, no error message will be displayed because the error output is redirected to /dev/null.

In Bash, bc -l is used to invoke the **bc calculator** with the **-l option**, which loads the standard math library. This enables high-precision calculations and provides access to additional mathematical functions, such as s, c, and a for trigonometric functions (sin, cos, and arctan), along with constants like π (scale, which specifies the precision).

Example Usage of bc -l

1. **Basic Arithmetic with Precision**:

result=$(echo "3.14159 * 2" | bc -l)

echo "Result: $result"

This multiplies 3.14159 by 2 with high precision.

explain this line area=$(echo "$dim1 * $dim1" | bc -l)

area=$(echo "$dim1 * $dim1" | bc -l)

Here's a breakdown of each part:

1. **echo "$dim1 * $dim1"**:
    - The echo command outputs a string containing the mathematical expression for calculating the area of a square (side × side).
    - $dim1 represents the length of one side of the square. If dim1 is 5, for instance, this part would output 5 * 5.
2. **| (Pipe)**:
    - The pipe | sends the output of the echo command as input to the next command, which is bc.
3. **bc -l**:
    - bc is a calculator language in Unix/Linux that can handle mathematical expressions.
    - The -l option loads the standard math library in bc, allowing for operations with floating-point precision (decimal values).

- bc -l reads the 5 * 5 expression and evaluates it, returning the result as a floating-point number.

4. **area=$(...)**:
    - The $() syntax captures the output of the entire echo "$dim1 * $dim1" | bc -l command and assigns it to the variable area.

So, if dim1=5, this line calculates 5 * 5 = 25 and stores 25 in area. This method also allows for floating-point operations, making it suitable for shapes with dimensions that aren't whole numbers.

The tee command in Unix/Linux is used to read from standard input and write to both standard output and one or more files simultaneously. It's particularly useful for logging output while still displaying it on the screen.

Basic Syntax

bash

Copy code

command | tee [options] filename

Options

- -a : Append to the specified file(s) rather than overwriting them.
- -i : Ignore interrupts, which can be useful if you don't want to stop the tee command on accidental interruptions.

Examples

1. **Writing output to a file and displaying it:**

bash

Copy code

echo "Hello World" | tee output.txt

2. **Appending output to a file:**

bash

Copy code

echo "Hello Again" | tee -a output.txt

3. **Writing output to multiple files:**

bash

Copy code

echo "Hello Everyone" | tee file1.txt file2.txt

4. **Using with sudo to capture output with elevated permissions:**

bash

Copy code

sudo command | tee output.log

The tee command is especially handy for situations where you want to save logs or outputs for debugging or analysis later while still seeing it on the terminal

SSH shell

What is SSH?

SSH (Secure Shell) is a cryptographic network protocol used to securely access remote machines over an unsecured network, such as the internet. It allows users to log into remote systems, execute commands, transfer files securely, and perform administrative tasks.

---

1. How SSH Works

SSH provides a secure channel between a client and server by encrypting the data that passes between them, ensuring that sensitive information (like passwords) is not exposed to potential eavesdroppers.

Common Use Cases for SSH:

- **Remote login**: Log into another server or system from your local machine.
- **Secure file transfer**: Using tools like scp or sftp.
- **Port forwarding**: Allow secure communication with services via SSH tunnels.
- **Remote execution**: Execute commands on a server from a local machine.
- **Server administration**: Perform administrative tasks remotely.

---

2. What is root login in SSH?

The **root account** is the superuser account on Unix/Linux systems that has unrestricted administrative privileges. If attackers gain access to this account via SSH, they could potentially take full control of the server.

Why Disabling Root Login is Important

Disabling direct SSH login for the root account is a security best practice because:

- Attackers often try to brute-force passwords targeting the root user.

- If root login is restricted, attackers must guess another username and its corresponding password before attempting escalation.

---

3. How to Adjust SSH Configurations for Better Security

To secure your SSH setup, you can modify its configuration file /etc/ssh/sshd_config. Here's how you can adjust configurations:

Step 1: Edit the SSH Configuration

1. Open the SSH configuration file with a text editor:

bash

Copy code

sudo vi /etc/ssh/sshd_config

Replace vi with your preferred text editor (nano, vim, etc.).

2. Find the line:

shell

Copy code

#PermitRootLogin yes

Uncomment the line (remove #) and change it to:

perl

Copy code

PermitRootLogin no

This setting ensures that the root user cannot log in directly over SSH.

Step 2: Change the Default Port (Optional but Recommended)

By default, SSH uses port 22. Changing the default port to a custom port can obscure the service from automated attackers.

1. Look for the line:

bash

Copy code

#Port 22

2. Uncomment it and set it to a non-standard port, e.g., 2222.

yaml

Copy code

Port 2222

Step 3: Use Public Key Authentication

Instead of relying on password authentication, you can use **public/private key pairs** to authenticate securely.

How?

1. Generate a key pair on your client machine:

bash

Copy code

ssh-keygen -t rsa -b 4096

2. Copy the public key to the server:

bash

Copy code

ssh-copy-id username@yourserverIP

3. Disable password authentication in sshd_config:

perl

Copy code

PasswordAuthentication no

This ensures that only users with the correct private key can connect to the server.

Step 4: Use SSH Access Controls

You can restrict which users or groups can access the server over SSH.

Edit /etc/ssh/sshd_config and add:

Copy code

AllowUsers yourusername

or:

Copy code

AllowGroups sshusers

This will ensure only specified users or groups can log in via SSH.

Step 5: Restart SSH Service

After making changes to sshd_config, restart the SSH service for the changes to take effect:

bash

Copy code

sudo systemctl restart sshd

---

4. Best Practices for Secure SSH Access

1. **Disable root login**: Always use a normal user account with sudo privileges.
2. **Use SSH key-based authentication**: This is more secure than using passwords.
3. **Change default port**: Change from the default port 22 to something less common.
4. **Restrict IP addresses**: Use a firewall to restrict SSH access to specific IP addresses.
5. **Enable Two-Factor Authentication (2FA)**: Add another layer of security.
6. **Monitor login attempts**: Use tools like fail2ban to detect and block repeated failed login attempts.
7. **Update regularly**: Ensure your system and SSH server are up to date with patches.

---

5. Why These Measures Matter

Hackers or malicious actors commonly attempt to brute force SSH passwords or exploit misconfigurations to gain unauthorized access to servers. By implementing security measures like disabling root login, enabling key-based authentication, and setting appropriate access controls, you significantly reduce the attack surface.

1. Trap Command:

bash

Copy code

trap 'kill $(jobs -p); exit' SIGINT SIGTERM

- **Purpose:** This sets up a signal handler to cleanly terminate the script and its background processes when it receives SIGINT (Ctrl+C) or SIGTERM (termination signal).
- **How it works:**
    - trap executes the specified command when a signal is received.
    - kill $(jobs -p):
        - jobs -p lists the process IDs of all background jobs started by the script.
        - kill sends termination signals to those processes.
    - exit ensures the script itself exits cleanly after killing the background jobs.

the --line-buffered option in the grep command ensures that the script processes and handles data dynamically and continuously.

How --line-buffered Works:

1. **Buffered Output by Default**:
    - By default, grep and many other command-line utilities buffer their output, especially when writing to a pipe. This means they may wait until a certain amount of data is collected before processing or outputting it.
    - Without --line-buffered, grep might delay processing lines from tail -f until the buffer is full, causing significant delays in real-time scenarios.
2. **Immediate Line Processing**:
    - The --line-buffered option forces grep to process each line as soon as it is received.
    - In this script, as tail -f streams lines from the log file (LOG_FILE), grep immediately filters lines containing "error" and sends them to the next command (tee).

Why It Ensures Dynamic and Continuous Logging:

- Errors are detected and logged to ERROR_LOG in real time, without delays caused by buffering.
- This behavior is critical for scenarios requiring continuous monitoring, such as real-time error detection and dynamic report generation in the script.

Without --line-buffered:

- You might experience noticeable delays in detecting and logging errors, especially if the log file (LOG_FILE) is not being updated frequently.

In summary, --line-buffered is essential in this script to ensure dynamic, real-time error processing and continuous logging functionality.