aws re/start

Managing Processes

Linux Fundamentals

Welcome to Managing Processes.

What you will learn

At the core of the lesson

You will learn how to:

- Define a process in Linux
- Describe basic commands for process management
- Compare the `at` and `cron` commands for job scheduling

aws re/start

During this lesson, you will learn how to:

- Define a process in Linux
- Describe basic commands for process management
- Compare the **at** and **cron** commands for job scheduling

What is a process?

This section defines a process in Linux.

Programs

- A program is a **series of instructions** given to a computer that **tells which actions the computer should take.**
- Programs are usually one of two kinds:
  - System programs
  - Application programs

aws re/start

Prior to covering what a process is, you must understand what a program is. Programs are instructions given to the computer to indicate which actions the computer should take.

There are two kinds of programs: system and application.
A system program has the following attributes:

- Primal computer functions
- Operating system commands
- Usually do not interface with the computer user

Utility programs are examples of system programs.
An application program has the following attributes:

- Comprehensive program that performs a specific function
- Can be used by a user or another program

Word processors, database management, and games are examples of application programs.
The following are the two major differences between system and application programs:

- A system program is a native computer function.
- An application program is added to the computer.

How a program is found

```
]$ echo $PATH
```

When the command is run, the system:

- Looks in the specified path
- Searches the $PATH variable for the program

```
[ec2-user]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ec2-user/.local/bin:/home/
ec2-user/bin
[ec2-user]$
```


aws re/start

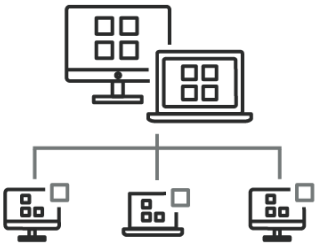
When a command is run, the system searches the $PATH variable for the executable file for the program corresponding to the command.

When the executable is found, the system loads the file into memory, schedules running time for it on the processor, and assigns it a process ID number (PID).

Most process administration is done by using this PID.

What is a process?

- Is a running program
- Is identified by process ID number (PID)
- Is viewed with multiple commands:
  - `ps`
  - `pstree`



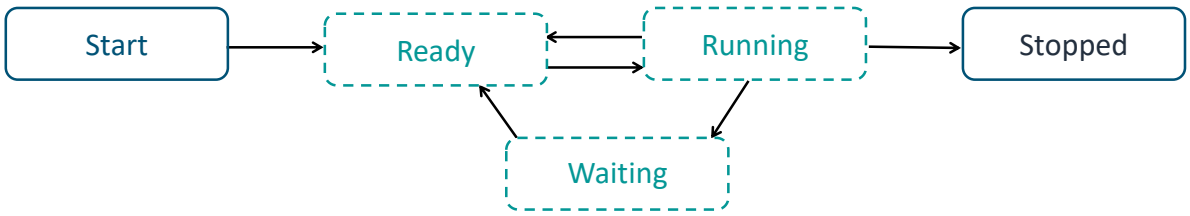
aws re/start

Processes found in a Linux system include services started by the operating system and programs started by individual users.

Process ID numbers are used to identify an active process that usually looks like a string of numbers, such as the following: 31532.

You can view these processes with a few commands, such as **ps** and **pstree**. The main difference between **ps** and **pstree** is that **ps** is shown in a list view, and **pstree** is shown in a tree view. Both commands show running processes in Linux.

States of a process



Start The process is created.

Ready The process is waiting to be assigned.

Running The process is in progress.

Stopped Everything has ended.

Waiting The process is waiting for an event.

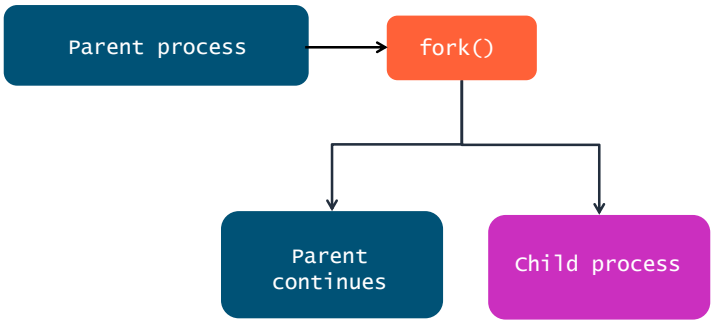
aws re/start

When a process runs, it can cycle through various states:

1. Start: The process is created.
2. Ready: The process is waiting to be assigned processor time.
3. Running: The process is in progress.
4. Waiting: The process is waiting for an event.
5. Stopped: The process is finished running.

What is a child process?

Some services and applications are complex and require more than one process to provide more functionality. These services spawn child process.



aws re/start

Some services and applications are complex and require more than one process to provide more functionality. These services spawn child process.

Child processes can also be known as sub processes.

Often, child processes inherit most of the attributes of the parent process.

Basic commands for process management

This section describes basic commands for process management.

The ps command

```
]$ ps [options]
```

- Displays current processes in your operating system

```
[ec2-user]$ ps
  PID TTY          TIME CMD
31532 pts/0    00:00:00 bash
31590 pts/0    00:00:00 ps
```


aws re/start

What is the **ps** command?

The **ps (process status)** command gives an overview of the current processes that will be running in your operating system (OS). Within this overview, it displays information on the active process that includes the following:

- Process ID (PID)
- Terminal type (TTY) that the user is using
- Time that the process has been running
- Command (CMD), which is the name of the command that launched the process

You can use the syntax **ps [options]** command to filter the information of the active processes.

Why would you want to use the **ps** command?

Linux is good at running multiple processes at one time. So as a user, you might need to understand what processes are running, how long a process has been running, and what PID a process is under. This information can be useful for troubleshooting purposes to save time and effort.

Example syntax:

- **ps –ef | grep sshd** -> Use the **grep** option to filter an exact process. In this

example, find the **sshd** process in the **ps** output. If you have multiple running processes, which can be pages long, you can use this command to filter to find an exact process.

- **ps-ef | less** -> You can use the **less** option to display all processes in a page-by-page format

How to display all processes:

- **ps –ef** -> Don’t use any option to display all processes.

Common ps command options

Option	Description
`-e`	List all current processes
`-b`	Use batch mode
`-fp <number>`	List processes by PID

aws re/start

The previous slide touched on the idea that you would use the **ps** command to identify current processes and troubleshoot. When troubleshooting using the **ps** command, there a few most commonly used commands and options.

Commands and options:

- **ps –e** -> Displays every current process. **–A** is an identical option.
- **ps –a** -> Displays all processes not associated with a terminal. The **–T** option displays all processes related to the terminal.
- **ps –r** -> Restricts to running processes only.
- **ps –ef** -> Views in full-format listing.
- **ps –fp <number>** -> Lists processes by PID.
- **ps L** -> Lists all format specifiers.

The pidof command

```
]$ pidof [options] programName
```

- This command displays the process ID (PID) of a running program.

```
]$ pidof sshd
```

- In this example, pidof sshd prints the PID of sshd as the following code shows.

```
[ec2-user]$ pidof sshd
31531 31499 2431
```


aws re/start

What is the **pidof** command?

The **pidof** command shows the PID of the current running program. It is used to print the PID name of a specific program. For example, **pidof sshd** will show the PID of **sshd**.

Why would you want to use the **pidof** command?

There are not a lot of options when using **pidof** because the use of **pidof** is almost like a set filter. Because it searches for a specific PID of a specific running program, you use it when you know exactly what you are looking for to save time.

The following is example syntax of how to use this command to run with a specific program: **pidof [program]** -> **pidof sshd** or **pidof bash**. The output will give the PID of either program when the command is run.

The following are options that you can use with **pidof [options]**:

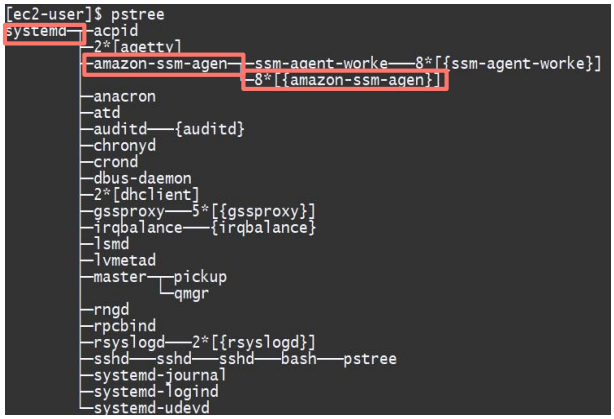
- **-s** -> Returns only one PID
- **-c** -> Returns only PIDs that are in the same root directory
- **-w** -> Shows processes that do not show a command line
- **-S** -> Serves as a separator that is used between PIDs

The pstree command

```
]$ pstree [options] [pid, user]
```

When a command is run:

- Displays the current running processes in a tree format
- Helps identify parent (denoted by [ ]) and child (denoted by { }) processes



aws re/start

What is the **pstree** command?

The **pstree** command displays the current running processes in a tree format. This command merges identical branches denoted by square brackets [ ] and child processes that are under the parent processes as denoted by curly brackets { }.

Why would you want to use the **pstree** command?

The root of the tree will be either the process, such as system (as used in this example), or init. Within the given process, you can see a process in a hierarchal way to understand what the output is. You can also see the parent and child processes.

The following is example syntax of how to use this command to run with a specific program: **pstree [options] [pid, user]**

This slide includes an example of how **pstree** merges identical branches. In the example on this slide, the process system is shown in tree view.

- systemd is the given root of the process.
- Amazon-ssm-agen is the process.
- The 8* in front of the 8*[{amazon-ssm-agen}] indicates that there are eight amazon-ssm-agen processes running.
- In [{amazon-ssm-agen}] the square brackets [ ] indicate that there is a parent

process. And the curly brackets { } indicate that there is a child process.

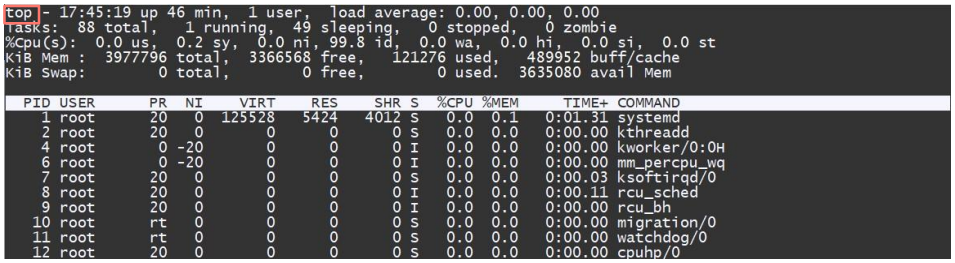
The following are common options that you can use with **pstree [options]**:

- **-a** -> Shows arguments in the command line within the output
- **-p** -> Shows the PID, which will show in decimal numbers inside parentheses after a process name
- **-c** -> Expands identical subtrees (and by default, compacts the tree)
- **-n** -> Sorts by the same parent PID instead of process name

The top command

```
]$ top [options]
```

The top command displays a real-time summary of system performance and utilization and lists the processes and threads active in the system.



aws re/start

What is the **top** command?

The **top** command displays a real-time system summary and information of a running system. It displays information and a list of processes or threads being managed.

Why would you want to use the **top** command?

- It displays system summary information and a list of processes or threads that the Linux kernel is currently managing.
- It provides a limited interactive interface for process manipulation and a more extensive interface for personal configuration. It encompasses every aspect of operation.

Common top options

Use these options to view and alter different aspects of a running system.

Option	Description
-h and -v	Displays usage and version information
-b	Starts top in Batch mode

```
[ec2-user]$ top -hv
  procps-ng version 3.3.10
Usage:
  top -hv | -bcHiOSs -d secs -n max -u|U user -p pid(s) -o field -w [cols]
```

```
[ec2-user]$ top -b
top - 17:46:31 up 47 min,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  88 total,   1 running,  49 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  3.2 sy,  0.0 ni, 96.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3977796 total,  3366940 free,   120900 used,   489956 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3635456 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0  125528   5424   4012 S   0.0  0.1   0:01.31 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
```


aws re/start

To view or alter different aspects of a running system, use **top [option]**:

- **-hv** -> Use this option for help and to show the version.
  - The **-h** option gives you the help section for the top command.
  - The **-v** option shows the library version.
- **-b** -> Use this option to start in batch mode.
  - This option starts **top** in batch mode, which is used primarily when sending output from **top** to other programs or other files.

Task status in top

A top task (process or thread) can be in one of the following states:

running	sleep	stopped	zombie
The process is running on the CPU or present in the run queue.	The process is waiting for an I/O operation to complete.	The process has been stopped by a job control signal or is being traced.	The process is a child process whose parent process has been ended.

Output

```
top - 20:12:28 up  3:13,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  90 total,   1 running,  49 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
KiB Mem :  3977796 total,  3359996 free,   122140 used,   495660 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3634520 avail Mem
```


aws re/start

The top command shows tasks (processes or threads) in one of the following states:

- **Running** is a process that is running on the CPU or present in the run queue.
- **Sleep** is a process that is waiting for an I/O operation to complete.
- **Stopped** is a process that has been stopped by a job control signal or that is being traced.
- **Zombie** is a child process whose parent process has been ended.

The output of the **top** command under **Tasks** contains this information.

CPU values in `top`

Value	Description
`us` value	Time spent running processes in user space
`sy` value	Time spent running kernel space processes
`id` value	Idle time
`wa` value	Time waiting for I/O to complete
`hi` value	Time handling hardware problems
`si` value	Time handling software problems
`st` value	Time lost waiting for other CPU processes to complete

Output

```
top - 20:12:28 up  3:13,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  90 total,   1 running,  49 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
KiB Mem :  3977796 total,  3359996 free,   122140 used,   495660 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3634520 avail Mem
```


CPU values in **top** are sections that show the various CPU usages throughout the system. These are used to analyze the effectiveness of the system. It is measured in percentages based on the latest interval from the most recent refresh.

- **us** value - Time spent running processes in user space
- **sy** value - Time spent running kernel space processes
- **id** value - Idle time
- **wa** value - Time waiting for I/O to complete
- **hi** value - Time handling hardware problems
- **si** value - Time handling software problems
- **st** value - Time lost waiting for other CPU processes to complete

Memory use and swap in `top`

Memory

This option shows the memory usage of the system.

Swap space

This option is used like random access memory (RAM) in a hard disk.

RAM usage

When RAM usage is close to full, regions that are used less frequently are written over to swap space, where they wait until retrieved.

Output

```
top - 20:12:28 up  3:13,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  90 total,   1 running,  49 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
KiB Mem :  3977796 total,  3359996 free,   122140 used,   495660 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3634520 avail Mem
```


aws re/start

There are three major memory uses and swaps in **top**:

- **Memory** shows the memory usage of the system.
- **Swap space** is used like random access memory (RAM) in a hard disk.
- **RAM usage** gets close to full, and regions that are used less frequently are written over to swap space, where they wait until retrieved.

Information is viewed at the top of the page after running **top**.

The value KiB stands for kibibytes, can be expresses as exbibytes (EiB) based on scaling, and can be enforced with the **E** option.

KiB Mem shows the physical memory and shows how much is free, used, and buff or cache.

KiB Swap shows virtual memory and shows how much is free, used, and available. (The available memory here is physical memory.)

The output of the **top** command under **KiB Mem** and **KiB Swap** contains this information.

The `kill` command

```
]$ kill [options] processID
```

When this command is run:

- It explicitly ends processes usually when the process fails to quit on its own.
  - If a parent process is ended, it usually also ends the child processes.
- A frequently used `kill` commands include the following:
  - `-9 SIGKILL` – Immediately stops the process with no graceful exit
  - `-15 SIGTERM` – Exits without immediately terminating
  - `-19 SIGSTOP` – Pauses the process and can use the command line

aws re/start

What is the **kill** command?

- It explicitly ends processes usually when the process fails to quit on its own.

Why would you want to use the **kill** command?

- If you've ever tried to close a program and it doesn't close but freezes instead, you might have used Task Manager to end the program. This command is very similar to that experience.

The following are **kill** command signals that you can use:

- **-9 SIGKILL** – Stops any process immediately
- **-15 SIGTERM** – Exits without immediately terminating
- **-19 SIGSTOP** – Pauses the process and can use the command line

Note that the name of the kill command will be changed in the future.

The `nice` and `renice` commands

```
]$ nice [options] [cmd]
```

- This command launches a new process at a given priority.
- The range is **-20** for **highest priority** and **19** for **lowest priority**.

```
]$ renice [options] [cmd]
```

- This command adjusts the priority of a process **that is already running**.

```
[ec2-user]$ nice -5 bash
[ec2-user]$ pidof bash
32271 32145
```


aws re/start

What is the **nice** command?

- It manages processes that are scheduled to be run at specific times on the CPU. The **nice** command manages the schedule priority.
- The highest priority for the nice command is -20, and the lowest priority is 19.

Why would you use the **nice** command?

It helps with running programs on a schedule with certain priorities. With **nice**, when a program is given a high priority, the kernel allows more CPU to process that request.

What is the **renice** command?

It adjusts the priority of a process.

Why would you use the **renice** command?

You would use this command when you need to adjust or modify the scheduling priority of a running process.

Demonstration: Using Various Tools to Manage Processes

Each instance of running code on the system is referred to as a *process*. Use `ps`, `pstree`, and `top` to view processes on the system.

aws re/start

1. Open a terminal window.
2. Run **ps** and examine the output.
3. Run **pstree** and compare the tree output to the previous **ps** command.
4. Run **top** and view the output. Press the **Q** key to exit.
5. Run **top –o cpu** to return the command with the output in order of CPU usage.

The jobs command

```
]$ jobs
```

- Jobs are processes that users start and manage that are identified by a job number.
  - The `jobs` command will list the job process number with the following syntax:
    - `bg [job process number or name]` to run in the background
    - `fg [job process number or name]` to run in the foreground
  - You use the `jobs` command to manage these processes.

aws re/start

Jobs are processes that the users start and manage that are identified by a job number. You use the **jobs** command to manage these processes.

Jobs that run in the foreground consume shell until they are completed. You can move jobs to the foreground with the following:

- **fg %jobnumber**

Jobs that run in the background continue to run, but the shell becomes available to use.

- You must suspend the job with **CTRL+Z** first.
- You use **bg %jobnumber** to move jobs to the background.

The **jobs** command lists the job process number with the following syntax:

- **bg [job process number or name]** to run in the background
- **fg [job process number or name]** to run in the foreground

The `at` and `cron` commands

```
]$ at [options]
```

This command runs a task once at a specified time.

- This command is used for one-time tasks, for example, running a backup script at 4 PM today only.
- To display a scheduled job, use the `at -l` option.
- To delete a scheduled job, use the `atrm [number]` command.

```
]$ cron [option]
```

This command runs a task on a regular basis at a specified time.

- This command is used for repetitive tasks, for example, running backup scripts on a weekly basis at 4 PM.
- This command maintains the list of tasks to run in a `crontab` file.

```
[root@server00 ~]# at 1600
at> ./demo-script.sh
at> <EOT>
job 2 at Wed Mar  6 16:00:00 2019
[root@server00 ~]# at -l
2       Wed Mar  6 16:00:00 2019 a root
[root@server00 ~]#
[root@server00 ~]# at -l
```


aws re/start

What are the **at** and **cron** commands?

- The **at** command is useful when having to run a task only once at a specified time.
- The **cron** daemon is useful when running tasks on a regular basis at a specified time.

Why would you use the **at** and **cron** commands?

- You can schedule tasks with the **at** and **cron** commands. Administrators can configure services or scripts to run automatically at specified times without having to be manually submitted.

There are also cron directories. These are precreated and prescheduled directories that administrators can use to store scripts that will run at specified times:

- **/etc/cron.daily**
- **/etc/cron.hourly**
- **/etc/cron.monthly**
- **/etc/cron.weekly**

The crontab command

```
]$ crontab -a fileName
```

Creates the crontab file that holds the commands and steps that the cron daemon will run

Can also be used to list, modify, or delete the crontab file

- The crontab file format has six fields, which stand for the following:
  - MIN: Minutes – any value from 0 to 59
  - HOUR: Hour – any value from 0 to 23
  - DOM: Day of month – any value from 1-31
  - MON: Month – any value from 1-12
  - DOW: Day of week - any value from 0-6
  - CMD: Command – any command or path

aws re/start

What is **crontab**?

crontab stands for cron table, is made up of a list of commands, and is also used to manage this table. The following fields make up the table:

- Minute
- Hour
- Day of the month
- Month of the year
- Day of the week
- Command

- crontab specifies commands or scripts to be run at a specific time, which the user can modify.
- The **cron** daemon can check the file each minute for scheduled tasks, and **cron** will run these tasks.
- These commands and steps are stored in the **crontab** file that holds the commands and steps that the cron daemon will run.

Why would you use it?

- This command is useful for regularly scheduled backups, clean-up scripts, scripts to rotate log files, or reminder messages.

Editing and listing the crontab file

```
]$ crontab -e
```

This command edits the crontab file as the root user:

- Before entering –e, copy the original crontab text to a new file that you can revert to if needed.

```
]$ crontab -l
```

To see a list, use `crontab –l`

aws re/start

Edit the **crontab** file as the root user by using the **crontab -e** command. This automatically installs the new **crontab** file, putting all changes into place.

It uses the system's designated default text editor (usually Vim).

Best practices:

- Use **crontab – l** to list files.
- Before you enter **-e**, copy the original **crontab** text to a new file that you can revert to if needed.

There is a system **crontab** and user **crontab**:

- System-wide scheduled tasks are managed by root at **/etc/crontab**.
- Each user manages user-specific tasks at **/var/spool/cron/crontabs/$USERNAME**.
- You should always be editing with **crontab –e**.

Checkpoint questions

1. Why might you need to stop a process?
2. Why is ps -ef | grep [process name] useful?
3. What is the difference between the at and cron commands?

aws re/start

Answers:

1. If a process becomes unresponsive, you must manually shut it down. An example would be a web server that—although it is running 100 percent of CPU—is not serving webpages. If the normal processes for stopping the web server fail, you must manually stop the process.
2. Using this command, an administrator can verify whether a program, a process, or an application is running.
3. You use the **at** command to run a single task and use **cron** to run repeated tasks.

Key takeaways

- Linux systems run daemons, services, and programs.
- You can use the `ps` and `pstree` commands to display running processes.
- You can use the `top` command to examine the processes that run on a server and to examine resource utilization.
- You can use the `at` and `cron` commands to schedule when a command is run.

aws re/start

Some of the key takeaways include the following:

- Linux systems run daemons, services, and programs.
- You can use the **ps** and **pstree** commands to display running processes.
- You can use the **top** command to examine the processes that run on a server and to examine resource utilization.
- You can use the **at** and **cron** commands to schedule when a command is run.

Thank you

aws re/start

Thank you.