Session 5

Agenda

- Part one: Text Processing
- Part two: Processes

Part One: Text Processing

Text Processing Agenda

1. Introduction to Text Processing
2. Searching and Pattern Matching : **grep**
3. Advanced Pattern Matching : **Regex**
4. Basic Text Manipulation commands
    - **cut**
    - **Sort**
    - **uniq**

Text Processing

- All Unix-like operating systems rely heavily on text files for data storage. So it makes sense that there are many tools for manipulating text.
- whether you’re managing servers, analyzing data, or maintaining complex configurations across multiple environments. The command-line approach offers unmatched speed, flexibility, and power, making it a key skill for anyone working in technical fields.

Why Text Processing is Crucial

- **Efficiency**: Efficient text processing helps in extracting valuable information quickly.
- **Data Analysis**: Processing logs and configuration files aids in monitoring troubleshooting, and performance tuning.
- **Customization**: Customizing outputs and generating reports tailored to specific needs

Searching and Pattern matching

grep : Searching in text

- "grep" stands for “global regular expression print,” so we can see that grep has something to do with regular expressions.
- command : `grep <option> <text> <file>`

Options

Option	Description
-l	Print name of each file that contain match text
-n	Print number of line with line that matches tex
-r	recursive searching in directory
-i	ignore case sensitive
-v	print all lines that doesn't match text
-c	Print the number of matches
-A	Print num lines of trailing context after matching lines.
-B	Print num lines of leading context before matching lines.

RegEx : Regular Expressions

- Regular Expressions are special characters which help search data and matching complex patterns. Regular expressions are shortened as ‘regexp’ or ‘regex’. They are used in many Linux programs like grep, bash, rename, sed, etc.

RegEx versions

1. BRE : Basic Regular Expression
2. ERE : Extend Regular Expression
3. PCRE : Perl-compatible Expression

Symbol Descriptions :

Symbol	Description
.	matches any single character
^	matches start of string
$	matches end of string

`\` : Use a backslash before a metacharacter to escape it.

- `grep "\." file` to match lines contain "."

`|` : OR, `grep "ab\|cd" file` all lines that have ab or cd

Special Characters : `.?*+{|()[\^$`

It's divided into two groups.

1. Characters lose their special meaning without using a backslash before them: `?+{}|()`
2. Characters are considered special by default: `.^*$\`

Repetition operators

Symbol	Description
`?`	matches zero or one for the preceding item
`*`	matches zero or more for the preceding item
`+`	matches one or more for the preceding item
`\{n\}`	matches exactly n times for the preceding item
`\{n,\}`	matches at least n times for the preceding item
`\{,m\}`	matches at most m times for the preceding item
`\{n,m\}`	matches from n to m times for the preceding item

? and +

```
hadeer@hadeer-ASUS:~/textProcessing$ grep "e\?" deep
dp dep deep
hadeer@hadeer-ASUS:~/textProcessing$ grep "e\+" deep
dp dep deep
hadeer@hadeer-ASUS:~/textProcessing$ grep -o  "e\+" deep
e
ee
hadeer@hadeer-ASUS:~/textProcessing$ grep -o  "e\?" deep
e
e
e
```

\{\}

```
hadeer@hadeer-ASUS:~/textProcessing$ cat fruits
Apple
Banana
Mango
Orange
Pineapple
Strawberry
hadeer@hadeer-ASUS:~/textProcessing$ grep "p\{1\}" fruits
Apple
Pineapple
hadeer@hadeer-ASUS:~/textProcessing$ grep -o "p\{1\}" fruits
p
p
p
p
hadeer@hadeer-ASUS:~/textProcessing$ grep -o "p\{2,\}" fruits
pp
pp
```

Character Classes and Bracket Expressions

`[range]` : `[a-b]`, `[0-1]`, `[123abc]`, `[a-z0-9A-Z]`

```
hadeer@hadeer-ASUS:~/textProcessing$ cat numbers
2
3
9
11
4
1
hadeer@hadeer-ASUS:~/textProcessing$ grep -o "[123]" numbers
2
3
1
1
1
hadeer@hadeer-ASUS:~/textProcessing$ grep "[1-5]" numbers
2
3
11
4
1
```

`[[:CLASSNAME:]]`

- **alnum**: alphanumeric characters (letters and numbers)
- **alpha**: alphabetic characters (letters)
- **digit**: digits (numbers)
- **lower**: lowercase letters
- **upper**: uppercase letters
- **punct**: punctuation characters
- **print**: Printable Characters , includes alnum, punt, space
- **space**: Space characters

Class name & invert match

```
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[[:digit:]]" info
John 30
Jane 25
Michael 50
Emily 20
David 11
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[^[:digit:]]" info
John 30
Jane 25
Michael 50
Emily 20
David 11
Sarah Johnson
```

Ranges & invert match

```
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[1-3]" info
John 30
Jane 25
Emily 20
David 11
hadeer@hadeer-ASUS:~/textProcessing/data$ grep "[^1-3]" info
John 30
Jane 25
Michael 50
Emily 20
David 11
Sarah Johnson
```

Special Backslash Expression

- `\b` : Match the empty string at the edge of a word.
- `\B` : Match the string provided it's not at the edge of a word.

```
hadeer@hadeer-ASUS:~/textProcessing$ grep "John" names
John    Doe
Sarah   Johnson
hadeer@hadeer-ASUS:~/textProcessing$ grep "\bJohn\b" names
John    Doe
hadeer@hadeer-ASUS:~/textProcessing$ grep "John\B" names
Sarah   Johnson
```

`( )` : are used for grouping.

- **Grouping**: `\(text\)`
    - example `grep "\(abc\)\{2\}" file` this will match abcabc
- **Capturing**: it capture a pattern, and in first match it save match in memory and specify number to refer to this match. So when you need to call or check for this exactly match again just call it by \num. numbers are arranged from 1 (1,2,3,..etc). In simple way you can check for duplicates words by using capture pattern.

```
hadeer@hadeer-ASUS:~/textProcessing$ cat capture
this is is text I wrote it it to check in in duplications. I it is amazing.
hadeer@hadeer-ASUS:~/textProcessing$ grep "\b\([[:alpha:]]\+\) \1" capture
this is is text I wrote it it to check in in duplications. I it is amazing.
hadeer@hadeer-ASUS:~/textProcessing$ grep -o "\b\([[:alpha:]]\+\) \1" capture
is is
it it
in in
```

Hands On #1  (10 Minutes)

- List information about files and directories in home that were last modified on "Aug 13"

Hands On #1 Solution

- `ls -al | grep "Aug 13"`

Basic Text Manipulation commands

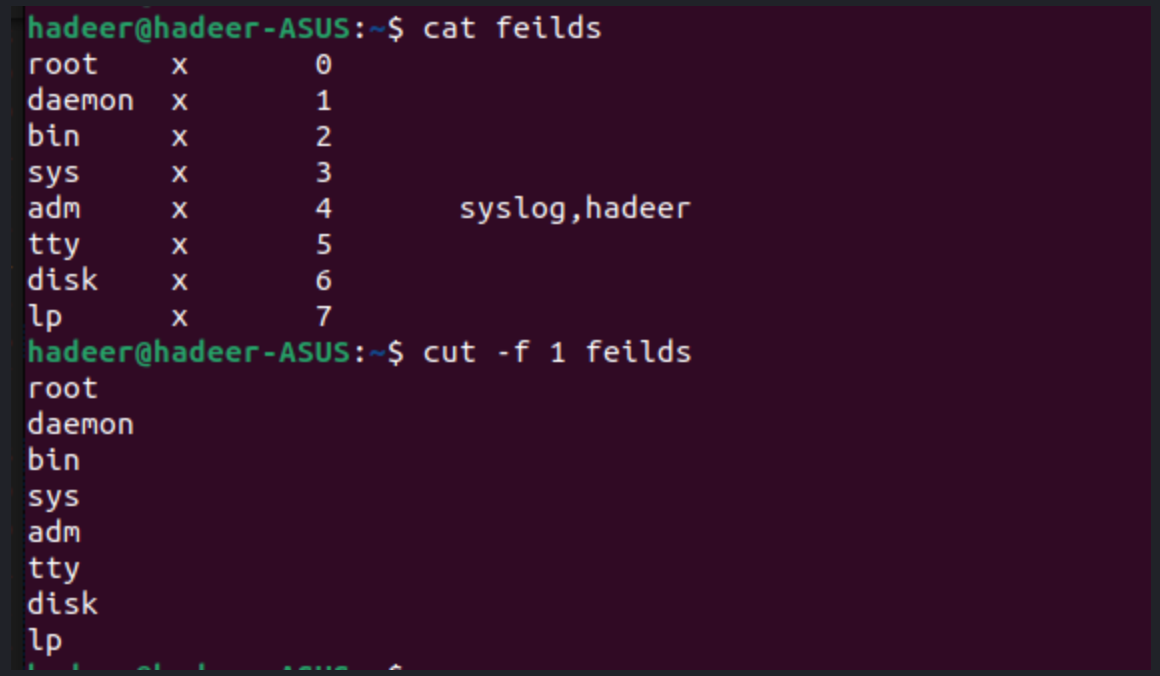
Cut

- The cut program is used to extract a section of text from a line and output the extracted section to standard output.
  command : `cut option <file>` (option is necessary)

cut Options

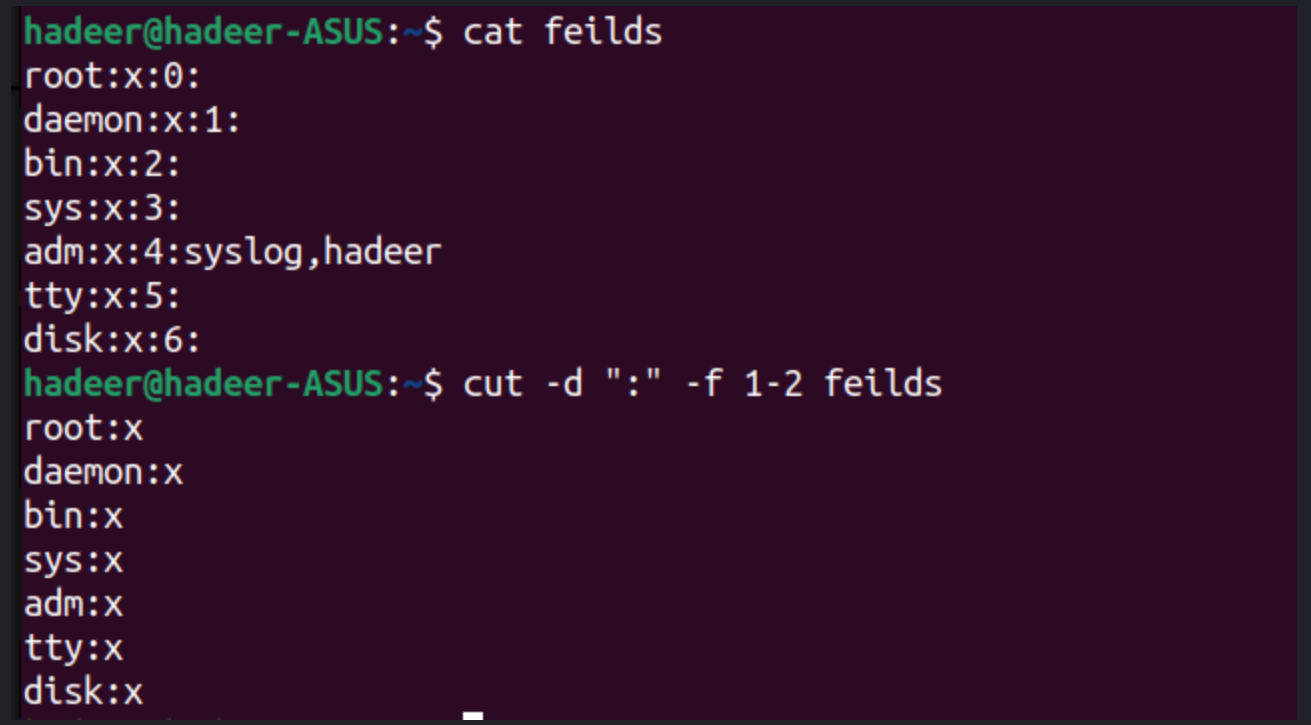
- `-b` : to extract specific bytes
  - `cut -b 1,2,3 file` or ranges `cut -b 1-3,5-6 file`
  - `cut -b 1- file` from 1 index line to end
  - `cut -b -5 file` from first to 5 index
- `-c` : to cut by character
  - `cut -c 2,5 names` print second and fifth character from each line
  - `cut -c 2-5 names` print form 2 to 5

- `-f` : To extract the useful information you need to cut by fields rather than columns
    - cut uses tab as a default field delimiter.
    - command : `cut -f (field number) file`



- `-d` : specifies the delimiter that separates fields, but it must be combined with the `-f` option to indicate which fields you want to extract based on that delimiter.

- `cut -d "delimiter" -f (field number) file`
  - example : `cut d ":" -f 1 file` cut first column
  - ranges : `cut -d "," -f 1-2 file` cut from first 1 index to 2



sort

- The sort program sorts the contents of standard input, or one or more files specified on the command line, and sends the results to standard output in a particular order.
- By default, it sorts file by ASCII code
  command : `sort <option> <filename>`

Sort options

option	Description
-c	to check if the file given is already sorted or not
-n	sort based on numeric values
-r	reverse sorting
-k	sorting a file according to any given field number (default delimiter : spaces)
-t	Define the field-separator character.

Examples in the sort command

```
hadeer@hadeer-ASUS:~/textProcessing$ sort -r fruits
Strawberry
Pineapple
Orange
Mango
Banana
Apple
hadeer@hadeer-ASUS:~/textProcessing$ sort -t "," -k 2 names2
Emily,Clark
John,Doe
Sarah,Johnson
Michael,Jordan
Jane,Smith
David,Wilson
```

uniq

- `uniq` : Report or omit Repeated Lines
- it removes any duplicate lines and sends the results to standard output. It is often used in conjunction with sort to clean the output of duplicates.
- **Only with sorted files**
  syntax : `uniq <option> <input file> <outFile>`
  OR : `uniq <input file> <output file>` by default , output will be all lines without duplication

uniq Options

Option	Description
-c	print each output line by the number of times it occur
-d	Display the repeated lines
-u	Display the lines that are not repeated

```
hadeer@hadeer-ASUS:~/textProcessing/uniq$ cat fruits
apple
banana
apple
orange
apple
grape
grape
grape
apple
hadeer@hadeer-ASUS:~/textProcessing/uniq$ sort fruits | uniq
apple
banana
grape
orange
```

Applications of Text

- Documents
- Web Pages
- Printer Output
- **And all software starts out as text. Source code, the part of the program the programmer actually writes, is always in text format.**

Break 🎉 (15 Minutes)

Part Two: Processes

Processes Agenda

- Listing Processes
- Process Relationships
- Process Types
- Killing Processes

Processes

- Linux is a multitasking operating system, which means that multiple programs can run at the same time.
- Processes are how Linux organizes the different programs waiting for their turn at the CPU (and other system resources).

Processes

- A **process** (or **task**) is an instance of a program. Each process has its own memory space and system resources allocated to it.

Processes

- Each process has the illusion that it is the only process on the computer, but in reality all processes share resources like the CPU and memory.
- Each process in Linux has a process ID (PID). and is associated with a specific user and group.

Listing Processes

- We can use `top` to show a continuously updating display of the system processes.

```
top - 17:14:49 up  2:05,  4 users,  load average: 1.44, 1.47, 1.65
Tasks: 420 total,   1 running, 419 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.0 us,  1.2 sy,  0.0 ni, 94.9 id,  0.3 wa,  0.4 hi,  0.2 si,  0.0 st
MiB Mem :   7683.7 total,    124.8 free,   7340.3 used,   2616.7 buff/cache
MiB Swap:   1277.0 total,      0.2 free,   1276.8 used.    343.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  12471 amna      20   0 1133.3g 215796  22736 S   7.9   2.7   2:18.24 codium
  12512 amna      20   0 1131.2g  61572  19580 S   7.6   0.8   1:56.19 codium
   9482 osc       -2   0 2179588 156168  97892 S   6.6   2.0   1:02.29 kwin_wayland
  13849 osc       20   0 2724464 162092  40340 S   6.3   2.1   0:03.14 spectacle
   2798 amna      20   0 1141.0g 266076  42792 S   3.6   3.4  13:13.29 Discord
   3390 amna      20   0   32.8g  62176   9692 S   2.6   0.8   3:04.32 codium
  12701 amna      20   0 2330552 203044  17324 S   1.3   2.6   0:21.62 language_server
   9635 osc       20   0 5935096 230112  44164 S   1.0   2.9   0:31.62 plasmashell
  13375 osc       20   0   12640   3144   1028 R   1.0   0.0   0:05.88 top
   1067 amna      20   0 5910168 233732  27416 S   0.7   3.0   5:14.08 plasmashell
  13178 amna      20   0 1935384  63772   9940 S   0.7   0.8   0:09.27 kscreenlocker_g
    451 root     -51   0       0      0      0 S   0.3   0.0   0:34.10 irq/154-ETD2205:00
```

- We can use the `ps` command to list processes.
- Running `ps` without any options lists processes associated with the current terminal session.

```
[osc@arch ~]$ ps
    PID TTY          TIME CMD
   6085 pts/2    00:00:00 bash
   6496 pts/2    00:00:00 ps
[osc@arch ~]$
```

- To print a more detailed output, run `ps -f`.

```
[osc@arch ~]$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
osc         6085    6082  0 19:52 pts/2    00:00:00 -bash
osc         6650    6085  0 19:58 pts/2    00:00:00 ps -f
[osc@arch ~]$
```

`ps` command

- To print all processes within the system, run `ps -e`.

```
$ ps -e
   PID TTY          TIME CMD
     1 ?        00:00:04 systemd
     2 ?        00:00:00 kthreadd
     3 ?        00:00:00 pool_workqueue_release
     4 ?        00:00:00 kworker/R-rcu_gp
     5 ?        00:00:00 kworker/R-sync_wq
     6 ?        00:00:00 kworker/R-slub_flushwq
     7 ?        00:00:00 kworker/R-netns
     9 ?        00:00:00 kworker/0:0H-events_highpri
    10 ?        00:00:00 kworker/0:1-events
    11 ?        00:00:00 kworker/u32:0-events_unbound
    12 ?        00:00:00 kworker/R-mm_percpu_wq
```

`ps` command

- To list a particular user's processes, run `ps -u [user name]` .

```
[osc@arch ~]$ ps -u osc
    PID TTY          TIME CMD
   3158 ?        00:00:00 systemd
   3163 ?        00:00:00 (sd-pam)
   3205 ?        00:00:00 kwalletd6
   3207 ?        00:00:00 startplasma-x11
   3217 ?        00:00:00 dbus-broker-lau
   3219 ?        00:00:00 dbus-broker
   3249 ?        00:00:00 kglobalacceld
   3252 ?        00:00:00 baloo_file
   3256 ?        00:00:00 xdg-desktop-por
   3262 ?        00:00:00 xdg-document-po
   3268 ?        00:00:00 xdg-permission-
   3285 ?        00:00:00 ksmserver
   3287 ?        00:00:00 kded6
   3288 ?        00:00:06 kwin_x11
```

`ps` command

- To search for a specific process by its name, we can pipe the output of `ps -e` to `grep`.

```
[osc@arch ~]$ ps -e | grep firefox
   1971 ?        00:03:03 firefox
  10398 ?        00:00:30 firefox
[osc@arch ~]$
```

- Alternatively, we can use the option `ps -C [process name]`.

```
[osc@arch ~]$ ps -C firefox
    PID TTY          TIME CMD
   1971 ?        00:03:04 firefox
  10398 ?        00:00:35 firefox
[osc@arch ~]$
```

- Another way to search for processes is by using `pgrep`.
- `pgrep` is a powerful command that searches for processes based on their name and other attributes, providing a more convenient alternative to combining `ps` and `grep`.
- By default, it returns only the PID(s) of matched processes, making it particularly useful in scripts for automating process management.

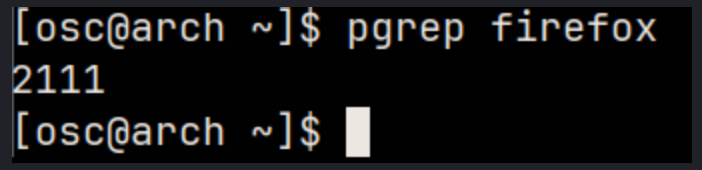
`pgrep` command

- The basic syntax for `pgrep` is:

```
pgrep [options] pattern
```

- To search for processes by name, simply provide the name as a pattern:

```
pgrep firefox
```



- This will return the process IDs (PIDs) of all processes matching the name "firefox".

- `-u [user]` : Search for processes owned by a specific user.
- `-l` : Lists the process name next to its PID.

```
pgrep -u osc -l
```

```
[osc@arch ~]$ pgrep -l -u amna
758 systemd
760 (sd-pam)
768 darkman
780 dbus-broker-lau
781 dbus-broker
784 kwalletd6
785 startplasma-way
791 pulseaudio
809 gsettings-helpe
859 kwin_wayland_wr
863 baloo_file
864 kwin_wayland
```

Process Relationships

1. Parent Process

- A **parent process** creates another process, either directly or indirectly.
- Every process has a parent process, except for **systemd** (PID 1). When a process is created, it inherits various attributes from its parent process, such as its working directory.

2. Child Process

- A **child process** is one created by another process (its parent process).
- Child processes inherit most of their attributes from their parent process but can also have their own unique attributes.

3. Orphan Process

- An **orphan process** is a process whose parent has terminated before them.
- Orphan processes are re-parented to **systemd**, which continues to manage them.

4. Zombie Process

- A **zombie process** is a process that has completed execution but still has an entry in the process table, as its parent process has not yet retrieved its exit status.
- Although zombies do not consume system resources like memory or CPU, they do occupy a slot in the process table.

Viewing process hierarchy

- To better understand process relationships, we can run `pstree` to view all system processes in a hierarch.

```
[osc@arch ~]$ pstree
systemd─┬─NetworkManager───3*[{NetworkManager}]
        ├─bluetoothd
        ├─containerd───13*[{containerd}]
        ├─dbus-broker-lau───dbus-broker
        ├─dhcpcd─┬─dhcpcd───dhcpcd
        │        └─2*[dhcpcd]
        ├─dnsmasq───dnsmasq
        ├─dockerd───16*[{dockerd}]
        ├─kwalletd6───5*[{kwalletd6}]
        ├─polkitd───3*[{polkitd}]
        ├─postgres───5*[postgres]
        ├─rtkit-daemon───2*[{rtkit-daemon}]
        ├─sddm─┬─Xorg───8*[{Xorg}]
        │      ├─sddm-helper───startplasma-way───{startplasma-way}
        │      └─{sddm}
```

`pstree` command

- `pstree $$` restricts the view to processes in the current shell.
- `pstree [username OR UID]` displays processes owned by a specific user.

Now that we understand what processes are, let's talk about a related but distinct concept: **jobs**.

Processes vs Jobs

Jobs

A **job** is a concept used by the shell. It is a unit of work performed by the user, which may consist of one or more processes.

For example, `ls | head` is a job consisting of two processes.

Jobs

A job is identified by a job ID (JID).

You can list the jobs of the current shell by running `jobs`.

```
[osc@arch ~]$ jobs
[1]-  Stopped                 xlogo
[2]   Running                 firefox &
[3]+  Stopped                 sleep 6
[osc@arch ~]$
```

Processes vs Jobs

We won't go deep into the differences between processes and jobs, but just note that they are distinct but related concepts, and that some commands take PID while others take JID.

Foreground and Background Processes

1. Foreground Processes

These are initialized and controlled through a terminal session. They need to be started by a user, as they don't start automatically as part of the system functions/services.

When a process is run in the foreground, no other process can be run on the same terminal until the process is finished or killed.

2. Background Processes

These are not connected to a terminal session and don’t expect any user input. They usually start automatically as part of the system functions/services.

However, user programs can be manually run as background processes. To run a process in the background, add an ampersand `&` at the end of the command:

```
[command] &
```

This launches the command in the background and returns control of the terminal to the user.

```
[osc@arch ~]$ firefox &
[6] 10083
[osc@arch ~]$
```

To bring a background or stopped (suspended) process to the foreground, use the `fg` command followed by the job ID.

```
fg %JID
```

```
[osc@arch ~]$ jobs
[1]+  Stopped                 xlogo
[4]   Running                 xlogo &
[5]   Running                 xlogo &
[6]   Running                 firefox &
[7]-  Done                    firefox
[osc@arch ~]$ fg %6
firefox
```

To resume a suspended process in the background, use the `bg` command:

```
bg %JID
```

```
[osc@arch ~]$ jobs
[4]+  Stopped                 xlogo
[5]-  Running                 xlogo &
[osc@arch ~]$ bg %4
[4]+ xlogo &
[osc@arch ~]$
```

Daemon Processes

A **daemon** is a type of background process that runs continuously, typically performing system-related tasks.

Daemon processes are often started during system boot-up and run in the background without any user interaction.

Daemons typically have no controlling terminal, which is indicated by a ? in the TTY field of the `ps` command's output.

```
[osc@arch ~]$ ps -e | grep ?
      1 ?        00:00:12 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 pool_workqueue_release
      4 ?        00:00:00 kworker/R-rcu_gp
      5 ?        00:00:00 kworker/R-sync_wq
      6 ?        00:00:00 kworker/R-slub_flushwq
      7 ?        00:00:00 kworker/R-netns
      9 ?        00:00:00 kworker/0:0H-events_highpri
     12 ?        00:00:00 kworker/R-mm_percpu_wq
     14 ?        00:00:00 rcu_tasks_kthread
     15 ?        00:00:00 rcu_tasks_rude_kthread
```

Killing Processes

Killing processes refers to ending or *terminating* them. The name is violent, just as killing processes can be, depending on the way it's done.

However, it might be useful if a certain process is not responding or causing the system to lag.

`kill` Command

To kill a process, we can find its PID and pass it to the `kill` command.

```
kill PID
```

`kill` Command

The `kill` command doesn't exactly “kill” processes; rather, it sends them **signals**. Signals are one of several ways that the operating system communicates with programs.

```
kill [signal] PID
```

Signals

There are many signals, but the most common ones are:

Number	Signal	Description
9	KILL	Terminate immediately, hard kill.
15	TERM	Terminate whenever, soft kill.
19	STOP	Pause the process. (CTRL+Z)
18	CONT	Resume the process.

`kill` Command

Signals can be specified in three ways:

1. By number:

```
kill -15 PID
```

2. By name:

```
kill -TERM PID
```

3. By name prefixed with `SIG`:

```
kill -SIGTERM PID
```

The default signal for `kill` is 15 or `TERM` (terminate).

```
[osc@arch ~]$ ps
    PID TTY          TIME CMD
   3915 pts/6    00:00:00 bash
   9260 pts/6    00:00:00 xlogo
   9264 pts/6    00:00:00 xlogo
  11132 pts/6    00:00:00 xlogo
  11139 pts/6    00:00:00 ps
[osc@arch ~]$ kill -TERM 11132
[6]+  Terminated              xlogo
[osc@arch ~]$
```

`pkill` Command

- Another option is `pkill`, which is the murderous version of `pgrep`.
- `pkill` is a command used to send signals to processes based on their name or other attributes.
- The basic syntax for `pkill` is:

```
pkill [options] pattern
```

`pkill` Command

- `-u [user]` : Only kill processes owned by a specific user.
- `-n` : Explicitly send the signal n (default is 15 or TERM).

Hands-On Task 2 

1. Open a terminal and run four `sleep` processes in the background with the following durations (800, 700, 600, and 400 seconds).
2. Bring the `sleep 600` process to the foreground, and then suspend it. *(Hint: Find its job ID)*.
3. The `sleep 800` process has been compromised and is a danger to your system! **Destroy** it **immediately**.
4. You are suspicious of all `sleep` processes now, but killing them is too aggressive. You should **suspend** all of them for now.
5. **Make sure** all `sleep` processes have been suspended to so that none of them start acting crazy.

Hands-On Task Solution

1. Open a terminal and run four `sleep` processes in the background with the following durations (800, 700, 600, and 400 seconds).

```
sleep 800 &
sleep 700 &
sleep 600 &
sleep 400 &
```

2. Bring the `sleep 600` process to the foreground, and then suspend it. *(Hint: Find its job ID).*

```
jobs
fg %JID
# To suspend, click CTRL+Z OR run
kill -STOP PID # from another terminal
```

3. The `sleep 800` process has been compromised and is a danger to your system! **Destroy** it **immediately**.

```
kill -KILL PID
```

4. You are suspicious of all `sleep` processes now, but killing them is too aggressive. You should **suspend** all of them for now.

```
pkill -STOP sleep
```

5. **Make sure** all `sleep` processes have been suspended to so that none of them start acting crazy.

```
jobs
```

Thank you!