Linux day-4 notes

Dealing with log file and we want to extract meaningful messages from it and also and all the error's from it.

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep "ERROR" logfile.txt
2024-02-01 07:32:18 ERROR [app-server-1] Failed to connect to payment gateway: Connection timed out
2024-02-01 07:32:19 ERROR [app-server-1] Transaction 392841 failed: PAYMENT_GATEWAY_ERROR
2024-02-01 07:48:12 ERROR [app-server-1] Invalid request: Missing required field 'authorization'
2024-02-01 08:10:14 ERROR [app-server-1] Database connection lost
2024-02-01 08:35:27 ERROR [app-server-1] Failed to process payment: INVALID_CARD_NUMBER
2024-02-01 08:55:27 ERROR [app-server-2] 500 Internal Server Error: Java heap space
2024-02-01 09:18:56 ERROR [app-server-1] Missing required configuration: SMTP_PASSWORD
2024-02-01 09:20:11 ERROR [notification-service] Failed to send email notification: Authentication failed
2024-02-01 09:45:30 ERROR [app-server-1] Database query failed: Deadlock detected
2024-02-01 10:15:22 ERROR [app-server-1] Failed to connect to external API: https://partner-api.example.com
2024-02-01 10:15:23 ERROR [app-server-1] External API error: Connection refused
2024-02-01 10:40:11 ERROR [app-server-1] 404 Not Found: Resource /api/v1/legacy_endpoint no longer exists
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$
```

Now if we want to search from the emails:

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -Eo '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' logfile.txt
admin@example.com
john.doe@company.org
sarah.jenkins@company.org
sarah.jenkins@company.org
michael.brown@example.net
lisa.wong@company.org
david.kim@example.com
emma.davis@company.org
carlos.rodriguez@example.org
admin@example.com
olivia.parker@company.org
james.wilson@example.net
sophia.nguyen@company.org
admin@example.com
ethan.miller@example.com
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$
```

```
Pattern Syntax
    -E, --extended-regexp
          Interpret PATTERNS as extended regular expressions (EREs, see below).
```

```
-o, --only-matching
      Print only the matched (non-empty) parts of a matching line, with each such part on a  separate  output
      line.
```

If we don’t use -o then it would search for and return extra details as well:

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -E '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9]+\.[a-zA-Z]{3,}' logfile.txt
2024-02-01 07:23:45 INFO [app-server-1] User 'admin@example.com' logged in successfully
2024-02-01 07:30:42 INFO [app-server-2] User 'john.doe@company.org' logged in successfully
2024-02-01 07:42:56 INFO [app-server-2] User 'sarah.jenkins@company.org' logged in successfully
2024-02-01 07:45:34 INFO [notification-service] Email notification sent to sarah.jenkins@company.org
2024-02-01 07:50:22 INFO [app-server-2] User 'michael.brown@example.net' logged in successfully
2024-02-01 08:07:56 INFO [app-server-2] User 'lisa.wong@company.org' logged in successfully
2024-02-01 08:30:42 INFO [app-server-2] User 'david.kim@example.com' logged in successfully
2024-02-01 08:45:30 INFO [app-server-2] User 'emma.davis@company.org' logged in successfully
2024-02-01 09:00:45 INFO [app-server-1] User 'carlos.rodriguez@example.org' logged in successfully
2024-02-01 09:20:45 INFO [app-server-1] Configuration updated by user 'admin@example.com'
2024-02-01 09:25:33 INFO [app-server-2] User 'olivia.parker@company.org' logged in successfully
2024-02-01 09:42:12 INFO [app-server-2] User 'james.wilson@example.net' logged in successfully
2024-02-01 10:00:45 INFO [app-server-2] User 'sophia.nguyen@company.org' logged in successfully
2024-02-01 10:22:33 INFO [app-server-1] User 'admin@example.com' logged in successfully
2024-02-01 10:42:56 INFO [app-server-2] User 'ethan.miller@example.com' logged in successfully
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -E '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9]+\.[a-zA-Z]{3,}' logfile.txt
```

and we don't mention E then it won't understand regex itself and won't find any pattern

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -E "completed in [1-9][0-9]{3,}ms" logfile.txt
2024-02-01 07:33:04 INFO [app-server-2] API request completed in 2781ms
2024-02-01 09:15:24 INFO [app-server-2] API request completed in 1878ms
2024-02-01 09:38:58 INFO [app-server-1] API request completed in 2156ms
2024-02-01 10:35:34 INFO [app-server-2] API request completed in 7245ms
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$
```

And in curly braces, it is not mentioning to length of output, but it would show the number of occurences of pattern from the previously mention regex

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ awk '{print $3}' logfile.txt  | sort | uniq -c
```

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ awk '{print $3}' logfile.txt  | sort | uniq -c
     13 DEBUG
     12 ERROR
     60 INFO
     12 WARN
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$
```

if we want to search for timing by removing m/s and other information

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ awk '/API request completed in/ {print $9}' logfile.txt | sed 's/ms//' | sort -n
312
1878
2156
2781
7245
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ awk '{print $3}' logfile.txt  | sort | uniq -c
```

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$  awk '/API request completed in/ {print $9}' logfile.txt  | sort -n
312ms
1878ms
2156ms
2781ms
7245ms
```

here we used sed command to replace with nthing we can even replace it with some name

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ awk '/API request completed in/ {print $9}' logfile.txt | sed 's/ms/jinesh/' | sort -n
312jinesh
1878jinesh
2156jinesh
2781jinesh
7245jinesh
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ awk '/API request completed in/ {print $9}' logfile.txt | sed 's/ms//' | sort -n
312
1878
2156
2781
7245
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$
```

now if we want to return cpu usage when greater then 70%

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -E "CPU usage: .*[7-9][0-9]%" logfile.txt
2024-02-01 08:03:11 INFO [monitoring] CPU usage: 45%, Memory: 79%, Disk: 52%
2024-02-01 08:51:14 INFO [monitoring] CPU usage: 52%, Memory: 81%, Disk: 52%
2024-02-01 09:31:14 INFO [monitoring] CPU usage: 48%, Memory: 76%, Disk: 53%
2024-02-01 10:19:14 INFO [monitoring] CPU usage: 62%, Memory: 83%, Disk: 53%
2024-02-01 10:51:14 INFO [monitoring] CPU usage: 58%, Memory: 85%, Disk: 54%
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -E "CPU usage: .*[7-9][0-9]%" logfile.txt
```

```
rootjinesh@DESKTOP-KN25QO6:/mnt/c/Users/srs33/Downloads$ grep -E "Memory: [7-9][0-9]" logfile.txt
2024-02-01 08:03:11 INFO [monitoring] CPU usage: 45%, Memory: 79%, Disk: 52%
2024-02-01 08:51:14 INFO [monitoring] CPU usage: 52%, Memory: 81%, Disk: 52%
2024-02-01 09:31:14 INFO [monitoring] CPU usage: 48%, Memory: 76%, Disk: 53%
2024-02-01 10:19:14 INFO [monitoring] CPU usage: 62%, Memory: 83%, Disk: 53%
2024-02-01 10:51:14 INFO [monitoring] CPU usage: 58%, Memory: 85%, Disk: 54%
```

Python:

to start python interpreter on wsl

```
rootjinesh@DESKTOP-KN25QO6:~/Codebase/bazel-python-project$ python3
Python 3.10.12 (main, Jan 17 2025, 14:35:34) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> age = 25
"John Doe">>> name = "John Doe";
>>> print(age)
25
>>> print(name)
John Doe
>>>
```

```
>>> import math;
>>> a = 5
>>> b = 2
>>>
>>> sum_ab = a + b
>>> product_ab = a * b
>>> difference = a - b
>>> quotient = a / b  # division in Python is float
rt(a)  # import >>> sqrt = math.sqrt(a)  # import math module
>>> print(sum_ab)
7
>>> print(quotient)
2.5
>>>
```

(by default input in python is in string)

```
>>> x = input("Enter number")
Enter number5
>>> if x.isdigit():
...     print(f"Age must be an integer.")
... else:
print("Please en...     print("Please enter an integer.")
...
Age must be an integer.
```

for loops

```
Please enter an integer.
>>> for i in range(3):
...     print(i + 1)
...
1
2
3
```

while loop

```
Enter your age: 33
>>> while age < 0:
...     print("Please enter a non-negative age.")
...     age = int(input("Enter your age again: "))
...
>>> age = int(input("Enter your age: "))
Enter your age: -6
>>> while age < 0:
...     print("Please enter a non-negative age.")
...     age = int(input("Enter your age again: "))
...
Please enter a non-negative age.
Enter your age again: -56
Please enter a non-negative age.
Enter your age again: 78
>>>
```

functions in python

```
>>> def greet(name):
...     return f"{name}'s name is {name}."
...
g = greet("Alice>>> greeting = greet("Alice")
g)>>> print(greeting)
Alice's name is Alice.
>>>
```

object creating class

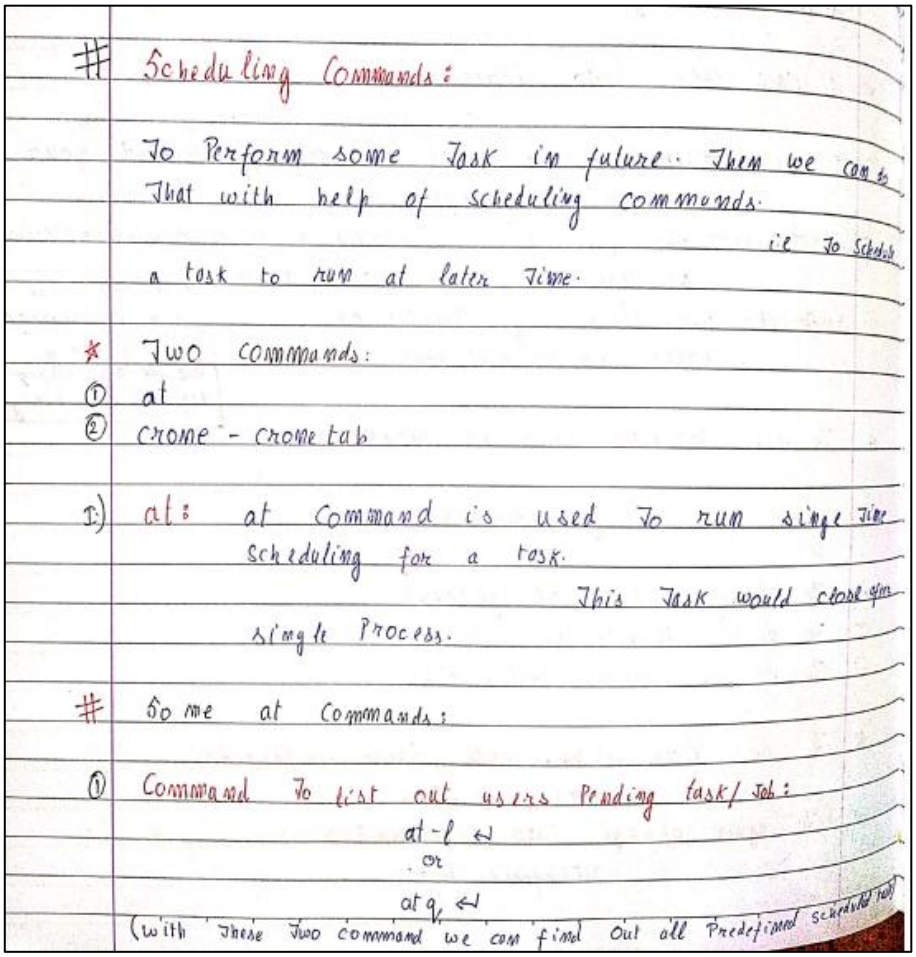
7. **Classes and Objects**

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def age(self):
        return 10 - self.age

dog = Dog("Buddy", 3)
print(dog.name)
print(dog.age())
```

Crontab:




```
rootjinesh@DESKTOP-KN25QO6:~/Codebase/bazel-python-project$ crontab -e
no crontab for rootjinesh - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.basic
  3. /usr/bin/vim.tiny
  4. /bin/ed

Choose 1-4 [1]: 1
No modification made
```

```
rootjinesh@DESKTOP-KN25Q
  GNU nano 6.2                                  /tmp/crontab.a5cPL2/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
30 2 * * * echo "hello world"
```

⑦ To schedule a task:

Syntax: $ at now ↵ # if we want to execute right now

$ at now + 5 min ↵ # if we want to execute after 5 min

now enter task say..

touch job12.txt # To create file

Then ctrl+D

ctrl+D (To save changes)

Output: Job 11 at Mon Jan 2 15:23:00 2023

no. of task

Various formats:

$ at 1:40 01 02 23 ↵

month date year

go to at mode give the task,

ctrl+D ↵

ctrl+D

⑧ To delete that task/job:

$ at -rm JobNo ↵

⑨ Crone - Crone tab:

(Crone Table)

it is a software utility, this automate a scheduled task at a Pre-defined time. This is daemon Process which runs as a background Process and Perform specified operations at a predefined time.

★ Chrome tab:

it is a list of command for executing the scheduled operations at a particular time.

So it permits → ① editing → $ Crontab -e ↵
② adding → $ Crontab -e ↵
③ removing → $ crontab -e ↵
④ To list scheduled job → $ Crontab -l ↵
⑤ To delete scheduled job → $ crontab -r ↵

To create cron job (or create a task using cron)

* * * * * Commandname ↵

↓	↓	↓	↓	↓
minutes	hours	date	months	days
(0-59)	(0-23)	(1-31)	(1-12)	(0-6)

minutes → (0-59)
hours → (0-23)
date → (1-31)
Month → (1-12)
day → (0-6) (sunday to saturday)

eg: After opening editor

36 15 2 11 Shut down ↵

Press esc To exit editor

:wq To save it.

P.T.O

Rules for Time/date format →

* → if in min we leave astric symbol, then it would perform task every other minute & lly for hour, date, month, day etc.

- → for range like (2-5) month, it would perform task in blw range 2 to 5 in month

, → for particular, like (3, 5, 6) ↵
i.e we want to assign some specific months. (or mins, hours, days etc.).

SHELL SCRYPTING:

A SHELL SCRYPT is a computer program designed to be run by UNIX/LINUX cell.
i.e it is a command line interpreter. (it executes line by line).

SOME Typical operations performed by shell scrypts:
1. file manipulation
2. Program execution
3. Printing Text

SE LINUX (Security Enhanced Linux):

it is a Linux Kernel. it provides high security developed by NSA. (National security)

SE Linux kernel security module, That provides mechanism for access control, security Policy is MAC (Mandatory Access control).

* it can run into Three Modes:

1. Disabled → here SE LINUX rules are not applied and your system is at higher risk.
2. Permissive → The file system is labelled & "access denial" entries are logged & no access is actualy denied. →

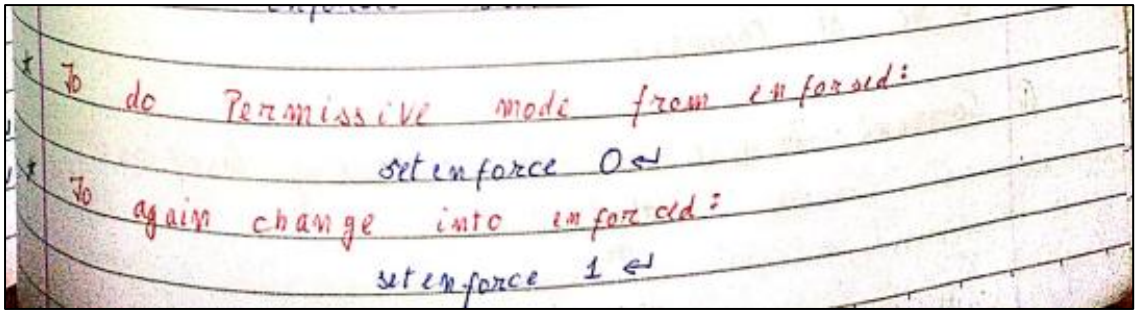
here we have given permission to every entry. But we are keeping log of each entry.

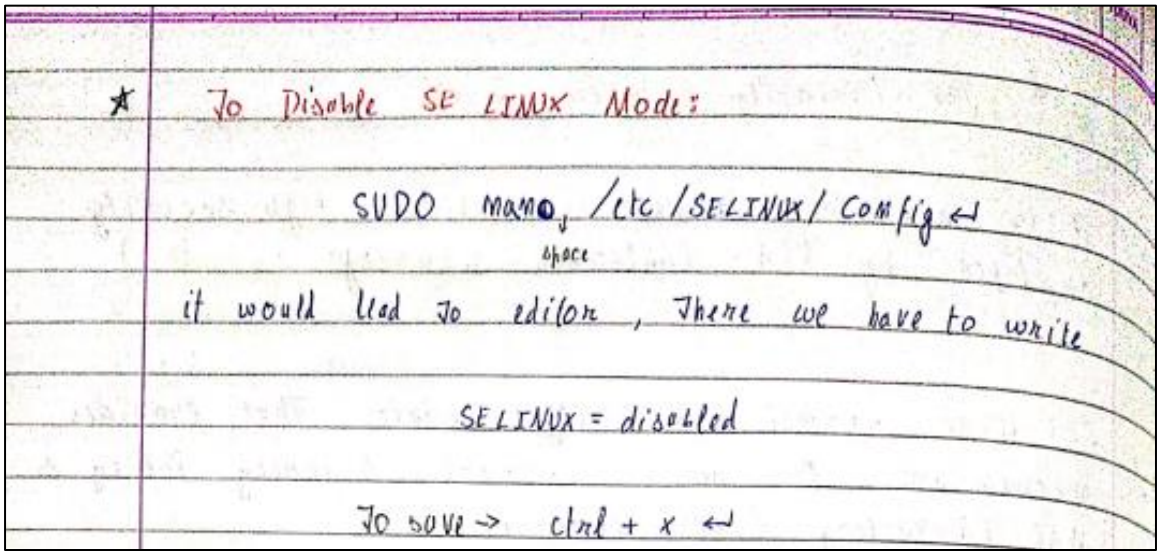
3. Enforced → here security Policies of SE LINUX is in Full Mode.

* To see working condition/status of SE LINUX:

getenforce ↵

* To change mode of system:
1. To do in Permissive mode → 0
2. To do in Enforced mode → 1


To do Permissive mode from enforced:
setenforce 0
To again change into enforced:
setenforce 1


To Disable SE LINUX Mode:
SUDO nano /etc/SELINUX/config
space
it would lead to editor, there we have to write
SELINUX = disabled
To save -> ctrl + x