Part 3

Pipes & Redirection (The real Power of Linux)

Big Idea First :

Linux commands **don't work alone.**

They are designed to talk to each other.

Output of one command → input of another command.

This is what makes Linux fast, clean and automatable.

Section 1 : Standard Streams

Every command uses 3 data streams :

Stream	Meaning
STDIN	Input (keyboard)
STDOUT	Output (screen)
STDERR	Error messages

Section 2 : Output Redirection (`>` and `>>`)

1. `>` — **Redirect output to a file (overwrite)**

```
ls > files.txt
```

What happens :

- `ls` normally prints to screens.
- now it writes into `files.txt`
- Existing content is replaced.
- Think :
  `>` = "send output to"

2. `>>` **— Append output**

```
echo "new line" >> files.txt
```

Adds content without deleting existing data.

Symbol	Action
`>`	Overwrite
`>>`	Append

Section 3 : Input Redirection (`<`)

```
sort < names.txt
```

Meaning :

- Take input from `names.txt`
- Not from keyboard.
- Less common, but useful in scripts.

Section 4 : Pipes (`|`) — The Most Important Operator

What is Pipe ?

```
command1 | command2
```

Meaning :

Output of command1 becomes input of command2.

Visual :

```
Command A → Command B
```

Example :

```
ls | w -l
```

Breakdown :

- `ls` → list files
- `wc -l` → count lines
- **Result** : number of files

Example :

```
ps aux | grep python
```

Meaning :

1. `ps aux` → list all processes
2. `grep python` → filter only python ones

This pattern is used daily by professionals.

Sections 5 : Chaining Multiple Commands

```
cat logfile.txt | grep "error" | wc -l
```

Meaning :

1. Read log
2. Extract error files
3. Count them

One line replaces multiple programs.

Section 6 : Redirecting Errors (Important in Automation)

STDERR Redirection

```
command 2> error.txt
```

Symbol	Meaning
`2>`	Redirect errors

Symbol	Meaning
`1>`	Redirect output

Both together :

```
command > output.txt 2> error.txt
```

Or everything :

```
command > all.txt 2>&1
```

Used heavily in scripts and cron jobs.

Section 7 : `tree` — View AND Save Output

```
ls | tree files.txt
```

- shows output on screen
- save to file

Append mode :

```
ls | tree -a files.txt
```

Section 8 : Practical, Real World Use Cases

1. **Save running processes**

```
ps aux > processes.txt
```

2. **Find large file**

```
ls -lh | greo "M"
```

---

3. **Monitor disk usage**

```
df -h | grep "/dev"
```

4. **Log analysis**

```
grep "failed" auth.log | wc -l
```

Section 9 : Common Beginner Mistakes

Forgetting space:

```
ls|wc -l #works but bad style
```

Proper :

```
ls | wc -l
```

Overwriting accidentally :

```
> file.txt
```

Safer :

```
>> file.txt
```