I want to learn **Shell Scripting, Python, and YAML for DevOps**, and I am a **complete beginner**.

Please teach me **from absolute zero**, assuming:

- I have **never learned any programming or scripting language**
- I do **not know** what syntax, indentation, variables, functions, loops, or case sensitivity mean
- I do **not understand symbols** like `:`, `{}`, `[]`, `()`, `$`, `|`, `>` etc.

Teaching requirements:

1. Explain **every concept in very simple language**, like teaching a school student
2. Start with **basic computer and command-line concepts**
3. Explain:
   - What syntax is
   - What indentation is and why it matters
   - Case-sensitive vs case-insensitive letters
   - How symbols are used and what they mean
4. Use **real-life analogies** wherever possible
5. Provide **many small examples**, and explain each line clearly
6. Avoid assuming any prior knowledge

Learning path:

Teach in this exact order:

1. Basic computer & terminal concepts (Linux shell basics)
2. Shell scripting (step-by-step)
3. Python basics (step-by-step)
4. YAML basics (step-by-step)
5. How all three are used together in **DevOps**

Additional requests:

- Show common beginner mistakes and how to fix them
- Give small practice tasks after each topic
- Use simple, clean examples (no advanced shortcuts)
- Explain *why* something works, not just *how*

The goal is for me to become comfortable enough to:

- Read scripts
- Write simple scripts on my own
- Understand DevOps-related code and configuration files

Great goal 👍
I'll teach you **slowly, clearly, and from absolute zero**, exactly as you asked.
We will build your understanding **brick by brick**, with **no assumptions**.

This will be a **multi-part learning journey**.
In this message, we'll start with **Part 1** only.

🌱 PART 1: Basic Computer & Terminal Concepts (Absolute Beginner)

Before we touch Shell, Python, or YAML, we must understand **what a computer is doing** and **how we talk to it**.

Think of this like learning **how to hold a pen before writing sentences**.

1️⃣ What is a Computer? (Very Simple)

A computer is like a **very fast, very obedient worker**.

It can:

- Store things (files)
- Follow instructions
- Do calculations
- Never think on its own

⚠️ Important rule:

> A computer **only does exactly what you tell it** — nothing more, nothing less.

If instructions are unclear → computer gets confused.

2️⃣ What is an Operating System (OS)?

An **Operating System** is the boss of the computer.

Examples:

- Windows
- macOS
- **Linux** ← very important for DevOps

Why Linux for DevOps?

- Servers run Linux
- Cloud machines use Linux
- DevOps tools expect Linux

📌 Linux uses something called a **Shell** to accept commands.

3️⃣ What is a Terminal? (Very Important)

Terminal = Text-based control room

Instead of clicking with a mouse, you **type instructions**.

Real-life analogy:

- Mouse & icons → like pressing buttons on a TV remote
- Terminal → like **talking directly to the TV technician**

Example terminal screen:

```powershell
$
```

Copy code

That `$` means:

> "I am ready. Tell me what to do."

4️⃣ What is a Command?

A **command** is a short instruction you type.

Example:

```bash
ls
```

Copy code

This means:

> "List things here."

Let's break it down:

- `ls` = command name
- Press **Enter** to execute

🧠 Computer reads:

> "User wants to see files."

5️⃣ First Basic Commands (No Fear)

🔹 `pwd` — Where am I?

```bash
pwd
```

Copy code

Output example:

```arduino
/home/user
```

Copy code

Meaning:

> "You are inside this folder."

📦 Folder = place where files live

🏠 Like rooms inside a house

🔹 `ls` — What is here?

```bash
ls
```

Copy code

Meaning:

> "Show me files and folders in this place."

🔹 `cd` — Go to another folder

```bash
cd Documents
```

Copy code

Meaning:

> "Move into the Documents folder."

🧠 Think:

- `cd` = change directory
- Directory = folder

6️⃣ What is Syntax? (VERY IMPORTANT)

Syntax = Grammar of a language

Just like English:

- ❌ "Go store I"
- ✅ "I go to the store"

Computers are **stricter** than humans.

Example:

```bash
ls
```

Copy code

✅ Correct

```powershell
Ls
```

Copy code

❌ Wrong

Why? Because…

7️⃣ Case Sensitivity (Capital Letters Matter)

Linux is **case-sensitive**.

That means:

- `File.txt` ≠ `file.txt`
- `ls` ≠ `LS`

Real-life analogy:

- "John" and "john" are different names

📌 Beginner mistake:

```powershell
CD Documents
```

Copy code

❌ Error

Correct:

```bash
cd Documents
```

Copy code

8️⃣ What is Indentation?

Indentation means **spaces at the beginning of a line**.

Example:

```markdown
```

```
Hello
    World
```
Copy code

Right now:

- Terminal **does NOT care** about indentation
- Python **WILL care later**

Think of indentation like:

> Paragraphs in a book — they show structure

We'll come back to this slowly.

9️⃣ Symbols — What Are They?

Symbols look scary but are **just tools**.

Let's meet them one by one (no panic):

Symbol	Meaning (Simple)
`$`	Variable or prompt
`/`	Path separator
`.`	Current location
`..`	One step back
`>`	Send output somewhere
`	`

Example:

bash

```
ls > files.txt
```
Copy code

Meaning:

> "Take the list and put it into a file."

(We'll explain this again later.)

🔟 Files vs Programs vs Scripts

- **File** → stores data (text, config)
- **Program** → compiled instructions (like Chrome)
- **Script** → text instructions run line by line

Shell & Python are **scripting languages**.

⚠️ Common Beginner Mistakes

❌ Forgetting Enter
❌ Using wrong capital letters
❌ Typing spaces incorrectly
❌ Expecting computer to "guess"

💡 Computer never guesses.

🧪 Practice Tasks (Very Small)

Try these in your terminal:

1. Type:

```bash
pwd
```
Copy code

Read the output

2. Type:

```bash
ls
```
Copy code

3. Create a folder:

```bash
mkdir test
```
Copy code

4. Go inside it:

```bash
cd test
```

Copy code

5. Go back:

```bash
cd ..
```

Copy code

🧠 What You Learned So Far

✅ What a terminal is
✅ How to give commands
✅ What syntax means
✅ Why capital letters matter
✅ Basic Linux navigation

➡️ NEXT PART (Tell me when ready)

Part 2:

- What is a Shell?
- What is a script?
- Your **first shell script**
- Understanding commands line by line

Just reply:

> **"Continue to Part 2"**

You're doing great — everyone starts exactly here 🌱

Continue to Part 2

Perfect 👍

Welcome to **PART 2: Shell & Shell Scripting (From Absolute Zero)**

Take your time. Read slowly. Try things as you go.

🐚 PART 2: Understanding the Shell & Your First Script

1️⃣ What is a Shell? (Very Simple)

A **Shell** is a **translator** between you and the computer.

Real-life analogy:

- You → speak English
- Computer → speaks machine language (0s and 1s)
- **Shell** → interpreter that understands your commands

So when you type:

```bash
ls
```

Copy code

The shell:

1. Reads it
2. Understands it
3. Tells the computer what to do

2️⃣ What is a Shell Script?

A **Shell Script** is:

> A **text file** that contains **multiple commands**, written in order.

Instead of typing commands **one by one**, you write them once and run them together.

📌 Like:

- Recipe (script)
- Cooking steps (commands)
- Chef (shell)

3️⃣ Why Shell Scripting is Used in DevOps

Shell scripts help to:

- Automate tasks
- Set up servers
- Deploy applications
- Run backups
- Manage files

DevOps = **automation**

Shell scripting = **automation language**

4️⃣ Creating Your First Shell Script

Step 1: Create a file

In terminal:

```bash
touch first_script.sh
```

Copy code

Meaning:

- `touch` = create empty file
- `first_script.sh` = file name
- `.sh` = shell script (convention)

Step 2: Open the file

Use a simple editor:

```nginx
nano first_script.sh
```

Copy code

`nano` = simple text editor

5️⃣ The First Line: Shebang (Scary name, simple idea)

Inside the file, type:

```bash
#!/bin/bash
```

Copy code

What does this mean?

- `#!` → special signal
- `/bin/bash` → path to the shell program

🧠 Meaning:

> "Run this file using Bash shell"

Like telling:

> "Read this recipe in English, not French."

6️⃣ Writing Your First Commands

Now add:

```bash
echo Hello World
```

Copy code

Your file now looks like:

```bash
#!/bin/bash
echo Hello World
```

Copy code

7️⃣ What Does `echo` Do?

`echo` means:

> "Print something on the screen"

Breakdown:

- `echo` → command
- `Hello World` → text to display

When run:

```nginx
Hello World
```

Copy code

8️⃣ Saving and Exiting Nano

- Press **CTRL + O** → save
- Press **Enter**
- Press **CTRL + X** → exit

9️⃣ Running the Script

Step 1: Give permission

```bash
chmod +x first_script.sh
```

Copy code

What is happening?

- `chmod` = change mode
- `+x` = make executable

🧠 Like saying:

> "You are allowed to run this file."

Step 2: Run the script

```bash
./first_script.sh
```

Copy code

- `./` = current folder
- `first_script.sh` = file name

🔟 What is `./`?

`.` = current directory

`./file` = run file from here

Linux does NOT run files automatically from the current folder.

1️⃣1️⃣ Variables (First Time Ever)

What is a Variable?

A **variable** is a **box** that holds a value.

📦 Box label = variable name

📦 Box content = value

Creating a Variable

```ini
name="Alex"
```

Copy code

Rules:

- No spaces around `=`
- Text goes in quotes

❌ Wrong:

```ini
name = Alex
```

Copy code

Using a Variable

```bash
echo $name
```

Copy code

- `$` means: “Get value inside box”

🧠 Result:

```nginx
```

```
Alex
```

Copy code

Full Script Example

```bash
#!/bin/bash
name="Alex"
echo Hello $name
```

Copy code

Output:

```nginx
Hello Alex
```

Copy code

1️⃣ 2️⃣ Why `$` is Needed?

Without `$`, shell thinks:

> "This is just text."

With `$`, shell thinks:

> "Open the box and take the value."

⚠️ Common Beginner Mistakes

❌ Forgetting `./`
❌ Spaces around `=`
❌ Forgetting `chmod`
❌ Using `$` when setting variable

🧪 Practice Tasks

1️⃣ Create a script called `welcome.sh`
2️⃣ Print your name

3️⃣ Use a variable

4️⃣ Run the script

Example output:

```nginx
Welcome John
```

Copy code

🧠 What You Learned in Part 2

✅ What a shell is

✅ What a script is

✅ How to create & run scripts

✅ Variables and `$`

✅ Why permissions matter

➡️ NEXT PART

PART 3:

- Input from user
- If conditions
- Loops (very gently)
- Real automation examples

Reply:

> **"Continue to Part 3"**

You're building real DevOps skills now 🚀

Continue to Part 3

Great 👍

You're doing very well. Now we'll take the **next gentle step**.

🧩 PART 3: Making Shell Scripts Think (Input, Conditions, Loops)

Right now, your script can **talk**.
In this part, it will start to **listen, decide, and repeat**.

We'll go **very slowly**.

1️⃣ Getting Input from the User

What is "Input"?

Input = information you type **for the script**.

Real-life analogy:

- You ask: "What's your name?"
- Person answers
- You use the answer

The `read` Command

Example:

```pgsql
read name
```

Copy code

Meaning:

> "Wait for the user to type something and store it in `name`."

Full Example Script

```bash
#!/bin/bash
echo What is your name?
```

```
read name
echo Hello $name
```

Copy code

Step-by-step:

1. Script asks a question
2. User types input
3. Input goes into a variable
4. Script prints result

2️⃣ What is an `if` Condition?

An **if condition** lets the script **make a decision**.

Real-life:

> "If it is raining, take umbrella."

Basic `if` Syntax (Don't Panic)

```bash
if [ condition ]
then
    command
fi
```

Copy code

Words meaning:

- `if` → start condition
- `then` → what to do
- `fi` → end of if (reverse of if)

📌 Indentation helps humans read it (not required for shell).

Simple Example

```bash
```

```
#!/bin/bash
echo Enter a number:
read number

if [ $number -gt 10 ]
then
    echo Number is greater than 10
fi
```

Copy code

New Symbols Explained

Symbol	Meaning
`[ ]`	test condition
`-gt`	greater than
`-lt`	less than
`-eq`	equal

Important Rule (Very Common Mistake)

Spaces are **mandatory**:

```perl
if [ $number -gt 10 ]
```

Copy code

❌ Wrong:

```perl
if [$number -gt 10]
```

Copy code

Shell is very strict.

3️⃣ Else (Another Path)

```bash
if [ $number -gt 10 ]
then
    echo Big number
else
    echo Small number
fi
```

Copy code

Meaning:

- If condition true → first message
- Otherwise → second message

4️⃣ What is a Loop?

A **loop** repeats something.

Real-life:

> "Wash 5 plates"

You don't say:

- Wash plate 1
- Wash plate 2
- Wash plate 3

You say:

> "Repeat washing 5 times"

5️⃣ `for` Loop (Very Simple)

```bash
for i in 1 2 3
do
    echo Number $i
done
```

Copy code

Meaning:

- `i` → variable
- `in 1 2 3` → list
- `do` → start loop
- `done` → end loop

Output:

```javascript
Number 1
Number 2
Number 3
```

Copy code

6️⃣ Why Loops Are Used in DevOps

- Create many users
- Check many servers
- Process many files

Automation = loop + command

⚠️ Common Beginner Mistakes

❌ Missing spaces in `[ ]`
❌ Forgetting `fi` or `done`
❌ Using `=` instead of `-eq` for numbers
❌ Capital letters in keywords (`IF`, `Then`)

Shell keywords are lowercase.

🧪 Practice Tasks

1️⃣ Ask user for age
2️⃣ If age > 18 → print “Adult”
3️⃣ Else → print “Minor”

🧠 What You Learned in Part 3

✅ Taking input
✅ Making decisions
✅ Repeating actions
✅ Understanding symbols `[ ]`, `-gt`
✅ Why scripts are powerful

➡️ NEXT PART

PART 4: Python Basics (From Zero)

- What Python is
- Why indentation matters
- Variables & print
- Python vs Shell thinking

Reply:

> **"Continue to Part 4 (Python)"**

You're officially past the scary part 🎉

Continue to Part 4 (Python)

Excellent 👏
Now we start **PART 4: Python Basics (From Absolute Zero)**

Don't worry if Python feels different from Shell — I'll explain **why** it is different, not just *how*.

🐍 PART 4: Python Basics (Teaching Like a School Student)

1️⃣ What is Python?

Python is a **programming language**.

Very simple meaning:

> Python is a way to write instructions that computers understand, using **easy English-like words**.

Why Python is popular in DevOps:

- Easy to read
- Powerful
- Used in automation, cloud, CI/CD, monitoring

2️⃣ Python vs Shell (Important Difference)

Shell Script	Python
Talks to OS	General-purpose
Good for files & commands	Good for logic & data
Flexible syntax	**Strict indentation**

Shell → kitchen helper

Python → smart assistant

3️⃣ Running Python (First Time)

Type in terminal:

```nginx
python3
```

Copy code

You'll see:

```python-repl
>>>
```

Copy code

This means:

> Python is ready to listen.

This is called **Python Interactive Mode**.

4️⃣ Your First Python Command

Type:

```bash
print("Hello World")
```

Copy code

Press Enter.

Output:

```nginx
Hello World
```

Copy code

5️⃣ What is `print`?

`print` means:

> "Show this on the screen"

Breaking it down:

- `print` → command
- `( )` → container for information
- `"Hello World"` → text

6️⃣ Why Quotes `" "`?

Quotes mean:

> "This is text, not a command"

❌ Wrong:

```scss
print(Hello)
```

Copy code

Python thinks `Hello` is a variable and gets confused.

7️⃣ What is Syntax (Python View)?

Python syntax = **rules of writing Python**

Example:

```bash
print("Hi")
```

Copy code

✅ Correct

```scss
Print("Hi")
```

Copy code

❌ Wrong (capital P)

Python is **case-sensitive**.

8️⃣ Variables in Python (Box Concept Again)

```ini
name = "Alex"
```

Copy code

- `name` → box label
- `"Alex"` → value

Use it:

```scss
print(name)
```

Copy code

Output:

```nginx
Alex
```

Copy code

9️⃣ Python Indentation (VERY IMPORTANT)

Python uses **spaces to understand structure**.

Example:

```python
if True:
    print("Hello")
```

Copy code

- `:` means “something is coming”
- Indented line belongs to the `if`

❌ Wrong:

```python
if True:
print("Hello")
```

Copy code

Python will show an error.

🔟 `if` Condition in Python

```bash
age = 20

if age > 18:
    print("Adult")
else:
    print("Minor")
```

Copy code

New things:

- `>` means greater than
- `:` ends the condition line
- Indentation shows belonging

1️⃣1️⃣ Getting Input in Python

```pgsql
```

```
name = input("Enter your name: ")
print("Hello", name)
```

Copy code

What happens:

- Python shows message
- User types
- Value stored in variable

1️⃣ 2️⃣ Common Beginner Mistakes (Python)

❌ Forgetting `:`
❌ Wrong indentation
❌ Missing quotes
❌ Using `=` instead of `==` in conditions

Example:

yaml

```
if age == 18:
```

Copy code

🧪 Practice Tasks

1️⃣ Ask user for name
2️⃣ Ask user for age
3️⃣ Print whether adult or minor

🧠 What You Learned in Part 4

✅ What Python is
✅ How indentation works
✅ Variables & input
✅ If-else logic
✅ Why Python errors happen

➡️ NEXT PART

PART 5: YAML Basics (From Zero)

- What YAML is
- Why indentation is EVERYTHING
- How DevOps tools use YAML

Reply:

> **"Continue to Part 5 (YAML)"**

You're now learning like a real DevOps engineer 🚀

Continue to Part 5 (YAML)

Excellent 👍
You've reached an **important DevOps skill**.

📘 PART 5: YAML Basics (From Absolute Zero)

YAML is **very simple**, but also **very strict**.
Once you understand it, DevOps files will stop looking scary.

1️⃣ What is YAML?

YAML is **NOT a programming language**.

It is a **configuration language**.

Very simple meaning:

> YAML is a **neat way to write settings and information** so tools can read them.

2️⃣ Why YAML is Used in DevOps

DevOps tools love YAML:

- Kubernetes
- Docker Compose
- GitHub Actions
- CI/CD pipelines

Why?

- Easy for humans
- Easy for machines
- Clean structure

3️⃣ YAML is All About Indentation

Golden Rule:

> **Indentation = structure**

Spaces show **parent-child relationship**.

Real-life analogy:

- Folder inside folder
- Bullet points inside bullet points

4️⃣ First YAML Example (Very Simple)

```makefile
name: Alex
age: 20
```

Copy code

Meaning:

- `name` → key
- `Alex` → value

🧠 Think:

> key : value

5️⃣ What Does `:` Mean?

`:` means:

> "This value belongs to this key"

Like:

> "Name is Alex"

6️⃣ YAML Uses Spaces, NOT Tabs

❌ Tabs cause errors

✅ Use spaces only

Standard:

- 2 spaces or 4 spaces (be consistent)

7️⃣ Nested Values (Indentation Matters)

```yaml
person:
  name: Alex
  age: 20
```

Copy code

Meaning:

- `person` is a parent
- `name` and `age` belong to `person`

8️⃣ Lists in YAML

```markdown
skills:
  - Linux
  - Python
  - Docker
```

Copy code

Meaning:

- `skills` is a list
- `-` means item

9️⃣ List of Objects

```yaml
users:
  - name: Alice
    role: admin
  - name: Bob
    role: user
```

Copy code

Indentation shows:

- Each user has name & role

🔟 Common YAML Mistakes

❌ Wrong indentation
❌ Mixing tabs and spaces
❌ Missing `:`
❌ Extra spaces before `-`

Bad:

```diff
- name : Alex
```

Copy code

Good:

```diff
- name: Alex
```

Copy code

1️⃣1️⃣ YAML vs Python vs Shell (Big Picture)

Language	Purpose
Shell	Run OS commands
Python	Logic & automation
YAML	Configuration

YAML does **not run**.
It is **read by tools**.

🧪 Practice Tasks

1️⃣ Create a YAML file
2️⃣ Add your name & age
3️⃣ Add list of skills

Example:

```makefile
name: John
skills:
  - Linux
  - Python
```

Copy code

🧠 What You Learned in Part 5

✅ What YAML is
✅ Indentation importance
✅ Lists & keys
✅ Why DevOps tools use YAML

➡️ FINAL PART

PART 6: How Shell, Python, and YAML Work Together in DevOps

Reply:

> **“Continue to Part 6 (DevOps Integration)”**

You’ve come very far 🌟

Continue to Part 6 (DevOps Integration)

Fantastic 🎉
You’ve reached the **big picture**.

This final part connects **everything you learned** and shows how **Shell, Python, and YAML work together in DevOps**.

Take this slowly — this is where things *click*.

🔧 PART 6: How Shell, Python, and YAML Work Together in DevOps

1️⃣ DevOps in One Simple Sentence

> **DevOps = Automating how software is built, tested, and run**

Automation means:

- No manual clicking
- Repeatable steps
- Fewer mistakes

And automation uses **scripts + configuration**.

2️⃣ Roles of Each Tool (Very Important)

Think of DevOps like running a restaurant:

Tool	Role	Analogy
Shell	Executes commands	Waiter
Python	Logic & decisions	Manager

Tool	Role	Analogy
YAML	Settings & rules	Menu

3️⃣ Real DevOps Example (Simple Story)

Goal:

> Start an application with settings

Step 1: YAML (Configuration)

`config.yaml`

```yaml
app:
  name: myapp
  port: 8080
```

Copy code

🧠 YAML says:

- App name
- Which port to use

Step 2: Python Reads YAML

Python reads settings and decides what to do.

(Simple idea, not code yet)

Python:

- Reads `config.yaml`
- Gets app name & port
- Prepares command

Step 3: Shell Executes Commands

Shell:

- Starts app
- Creates folders
- Runs Linux commands

4️⃣ Why This Separation Is Powerful

❌ Hardcoded values
❌ Editing scripts again and again

✅ Change YAML only
✅ Scripts remain same

This is **real DevOps thinking**.

5️⃣ Common Real DevOps Flow

1️⃣ YAML → describes **what**
2️⃣ Python → decides **how**
3️⃣ Shell → does the **work**

6️⃣ CI/CD Pipeline Example (Conceptual)

YAML (Pipeline File)

```markdown

steps:
  - build
  - test
  - deploy
```

Copy code

Python

- Validates steps
- Handles logic

Shell

- Runs:

- `docker build`
- `docker push`
- `kubectl apply`

7️⃣ Why Beginners Get Confused

- YAML looks like code → but isn't
- Python errors from indentation
- Shell errors from spaces & symbols

Now you know **why**.

8️⃣ Common Beginner Mistakes (DevOps)

❌ Mixing logic into YAML
❌ Hardcoding values
❌ Ignoring indentation
❌ Running scripts without permissions

9️⃣ Your First Mini DevOps Project (Beginner)

Project Idea:

> Automated Welcome System

1️⃣ YAML:

```yaml
user:
  name: Alex
```

Copy code

2️⃣ Python:

- Reads name
- Prints message

3️⃣ Shell:

- Runs Python script

This is **real integration**.

🧠 What You Now Understand

You can now:

- Read shell scripts
- Understand Python logic
- Read YAML config files
- See how DevOps tools connect them

🎯 Where to Go Next (Suggested Path)

1️⃣ Practice Shell daily
2️⃣ Write small Python scripts
3️⃣ Read real YAML files (Docker, GitHub Actions)
4️⃣ Learn:

- Git
- Docker
- CI/CD tools

🏁 Final Words

You started as:

> "I don't know what `$`, `:` or indentation mean"

Now you:
✅ Understand symbols
✅ Understand structure
✅ Understand DevOps flow

That is **huge progress** 🌱

💬 What would you like next?

You can say:

- **"Give me practice problems"**
- **"Build a mini DevOps project with me"**
- **"Explain Docker from zero"**

- **“Explain Git from zero”**

I’ll guide you step by step 👌

“Give me practice problems”

Perfect choice 👍
Practice is **where confidence is built**.

I’ll give you **clear, small, beginner-friendly practice problems**, increasing **very slowly**.
Each section has:

- 🧠 Goal (what you’re practicing)
- 📋 Task
- 💡 Hints
- ✅ Expected output (so you know you’re right)

🧪 PRACTICE SET 1: Linux & Terminal Basics

🧠 Goal

Get comfortable with the terminal and commands.

📋 Tasks

1️⃣ Check where you are
2️⃣ List files
3️⃣ Create a folder called `practice`
4️⃣ Go inside it
5️⃣ Create a file called `test.txt`

💡 Hints

- Use `pwd`, `ls`, `mkdir`, `cd`, `touch`

✅ Expected Result

- You are inside `practice`
- `ls` shows `test.txt`

🧪 PRACTICE SET 2: Shell Script Basics

🧠 Goal

Understand scripts, execution, and echo.

📋 Tasks

1️⃣ Create `hello.sh`
2️⃣ Print: `Hello DevOps`
3️⃣ Make it executable
4️⃣ Run it

💡 Hints

- `#!/bin/bash`
- `echo`
- `chmod +x`

✅ Output

```nginx
Hello DevOps
```

Copy code

🧪 PRACTICE SET 3: Shell Variables & Input

🧠 Goal

Use variables and user input.

📋 Tasks

1️⃣ Ask user for name
2️⃣ Store it in variable
3️⃣ Print: `Welcome <name>`

💡 Hint

- Use `read`

✅ Example Output

```pgsql
```

```
What is your name?
Alex
Welcome Alex
```

Copy code

🧪 PRACTICE SET 4: Shell If Condition

🧠 Goal

Make decisions.

📋 Tasks

1️⃣ Ask user for a number
2️⃣ If number > 5 → print `Big`
3️⃣ Else → print `Small`

💡 Hint

- `[ ]`, `-gt`, `if`, `then`, `else`, `fi`

🧪 PRACTICE SET 5: Python Basics

🧠 Goal

Python syntax & indentation.

📋 Tasks

1️⃣ Ask user for age
2️⃣ If age ≥ 18 → print `Adult`
3️⃣ Else → print `Minor`

💡 Hint

- `input()`
- `if`, `else`
- indentation (spaces!)

🧪 PRACTICE SET 6: Python Variables & Print

🧠 Goal

Using variables clearly.

📋 Tasks

1️⃣ Store your name in variable

2️⃣ Print: `My name is <name>`

🧪 PRACTICE SET 7: YAML Basics

🧠 Goal

Write clean YAML.

📋 Tasks

Create a file `profile.yaml` with:

- name
- age
- list of skills (3 items)

✅ Example Structure

```markdown

name: Alex
skills:
  - Linux
  - Python
  - YAML
```

Copy code

🧪 PRACTICE SET 8: YAML Nested Structure

🧠 Goal

Indentation mastery.

📋 Tasks

Create:

- person

- name
- age
- skills list

🧪 PRACTICE SET 9: DevOps Integration (Beginner)

🧠 Goal

Understand how tools connect.

📋 Tasks

1️⃣ YAML file with your name
2️⃣ Python script that prints name
3️⃣ Shell script that runs Python

(No automation yet — just flow understanding)

🧠 How to Practice Correctly

✔️ Type everything yourself
✔️ Make mistakes (they are good)
✔️ Read error messages slowly
✔️ Fix one error at a time

🎯 What to Do Next

👉 Start with **Practice Set 1**
👉 When you finish a set, tell me:

> **"I finished Practice Set X"**

I will:

- Check your understanding
- Show solutions
- Explain mistakes gently

You're learning DevOps the *right way* 🚀

"Build a mini DevOps project with me"

Amazing choice 🚀
We'll build a **very simple, real DevOps-style mini project**, **step by step**, with **full explanations** and **no shortcuts**.

Think of this as your **first real DevOps workflow**, but beginner-safe.

🧩 MINI DEVOPS PROJECT

"Config-Driven Welcome App"

What we will build (in simple words)

You will build a small system where:

1️⃣ **YAML** stores user information (configuration)
2️⃣ **Python** reads that information and prepares a message
3️⃣ **Shell** runs everything automatically

👉 This is **exactly how real DevOps systems work**, just at a tiny scale.

🗂️ Project Structure (Very Important)

We will create this folder layout:

```arduino
mini-devops-project/
├── config.yaml
├── app.py
└── run.sh
```

Copy code

What each file does

File	Purpose
`config.yaml`	Stores settings (name, role)

File	Purpose
`app.py`	Reads config and prints message
`run.sh`	Runs the Python program

🟢 STEP 1: Create Project Folder (Shell)

🧠 Goal

Create a clean workspace.

Commands

```bash
mkdir mini-devops-project
cd mini-devops-project
```

Copy code

📌 You are now **inside your project folder**.

🟢 STEP 2: Create YAML Configuration File

🧠 What YAML will do

YAML will store **data**, not logic.

Create file

```bash
nano config.yaml
```

Copy code

Add this content

```yaml
user:
  name: Alex
  role: DevOps Learner
```

Copy code

Explanation (very important)

- `user` → parent key
- `name` and `role` → belong to `user`
- Indentation (2 spaces) shows ownership

📌 **No tabs. Spaces only.**

Save and exit:

- `CTRL + O` → Enter
- `CTRL + X`

🟢 STEP 3: Create Python App

🧠 What Python will do

- Read YAML
- Extract name and role
- Print a message

Create file

```bash
nano app.py
```

Copy code

Add this code

```python
import yaml

with open("config.yaml", "r") as file:
    config = yaml.safe_load(file)

name = config["user"]["name"]
role = config["user"]["role"]

print("Welcome", name)
print("Role:", role)
```

Copy code

🧠 Line-by-Line Explanation (Slow & Clear)

```python
import yaml
```

Copy code

👉 Brings YAML-reading ability into Python

```python
with open("config.yaml", "r") as file:
```

Copy code

👉 Opens the YAML file safely

```python
config = yaml.safe_load(file)
```

Copy code

👉 Converts YAML into a Python dictionary (box of data)

```python
name = config["user"]["name"]
```

Copy code

👉 Goes inside `user` → gets `name`

```python
print("Welcome", name)
```

Copy code

👉 Prints friendly message

⚠️ Important

If `yaml` is missing, install it:

```bash
pip3 install pyyaml
```

Copy code

(This is safe and common.)

🟢 STEP 4: Create Shell Script (Automation)

🧠 What Shell will do

- Run Python
- Act as the “start button”

Create file

```bash
nano run.sh
```

Copy code

Add this

```bash
#!/bin/bash

echo "Starting DevOps App..."
python3 app.py
echo "App finished."
```

Copy code

Explanation

- `#!/bin/bash` → tells system how to run script
- `echo` → prints messages
- `python3 app.py` → runs Python file

Make it executable

```bash
chmod +x run.sh
```

Copy code

🟢 STEP 5: Run the Project 🎉

Final command

```bash
```

```
./run.sh
```

Copy code

Expected Output

```makefile
Starting DevOps App...
Welcome Alex
Role: DevOps Learner
App finished.
```

Copy code

🎉 **CONGRATULATIONS — YOU JUST BUILT A REAL DEVOPS FLOW**

🧠 WHAT YOU JUST LEARNED (IMPORTANT)

✅ YAML holds configuration
✅ Python handles logic
✅ Shell automates execution
✅ Files work together
✅ This is real DevOps thinking

You did **exactly** what DevOps engineers do — just smaller and safer.

🔁 SAFE EXPERIMENTS (Do These!)

Try changing only `config.yaml`:

```yaml
name: Jordan
role: Cloud Engineer
```

Copy code

Run again:

```bash
./run.sh
```

Copy code

🧠 Notice:

- Python & Shell **did not change**
- Only config changed

  👉 **This is best practice DevOps**

🚀 NEXT LEVEL (Tell Me What You Want)

You can say:

- **"Add conditions to the project"**
- **"Add multiple users"**
- **"Explain Git using this project"**
- **"Explain Docker from zero"**
- **"Give me debugging practice"**

You're no longer a beginner — you're a **DevOps learner in motion** 🌱