Here’s a clear and practical guide to **loops** and **if-else** statements in **Unix/Linux shell scripting** (primarily **Bash**, the most common shell).

1. If-else / If-elif-else (Conditional Statements)

Basic syntax:

```bash
if [ condition ]; then
    # commands when condition is true
elif [ another_condition ]; then
    # commands when first condition false but this is true
else
    # commands when all above conditions are false
fi
```

Important notes:

- Spaces around `[` and `]` are **mandatory**
- Use double quotes around variables: `"$var"`
- Common test operators:
    - `-eq` → equal (numbers)
    - `-ne` → not equal
    - `-lt` → less than
    - `-gt` → greater than
    - `-le` → less or equal
    - `-ge` → greater or equal
    - `=` / `==` → string equal
    - `!=` → string not equal
    - `-z "$str"` → string is empty
    - `-n "$str"` → string is non-empty
    - `-f file` → file exists and is regular file
    - `-d dir` → directory exists

Examples:

```bash
#!/bin/bash

age=25

if [ "$age" -ge 18 ]; then
    echo "You are an adult"
else
    echo "You are a minor"
fi
```

More realistic example with `elif`:

```bash
#!/bin/bash

marks=85

if [ "$marks" -ge 90 ]; then
    echo "Grade: A+"
elif [ "$marks" -ge 80 ]; then
    echo "Grade: A"
elif [ "$marks" -ge 70 ]; then
    echo "Grade: B"
else
    echo "Grade: C or below"
fi
```

File check example:

```bash
#!/bin/bash

file="notes.txt"

if [ -f "$file" ]; then
    echo "$file exists → opening it..."
    cat "$file"
else
    echo "$file not found! Creating empty file."
    touch "$file"
fi
```

2. Loops in Bash

There are three main loop types:

A. for loop (most common)

Best when you know the list/range in advance.

Style 1 – list style (classic)

```bash
for var in apple banana orange grapes
do
    echo "I like $var"
done
```

Style 2 – C-like style (very popular)

```bash
for (( i=1; i<=10; i++ ))
do
    echo "Number: $i"
done
```

Style 3 – range with `{start..end}` (Bash 3+)

```bash
for i in {1..8}
do
    echo -n "$i "
done
echo   # new line
```

Loop over files:

```bash
for file in *.txt
do
    echo "Processing: $file"
    wc -l "$file"   # count lines
done
```

B. while loop (runs while condition is true)

```bash
#!/bin/bash

count=1

while [ $count -le 5 ]
do
    echo "Count = $count"
    ((count++))        # or: count=$((count+1))
done
```

Interactive example (keep asking until correct):

```bash
while true
do
    read -p "Enter password: " pass
    if [ "$pass" = "secret123" ]; then
        echo "Access granted!"
        break
    else
        echo "Wrong password. Try again..."
    fi
done
```

C. until loop (runs until condition becomes true)

Opposite of while.

```bash
num=1

until [ $num -gt 5 ]
do
    echo "Number: $num"
    ((num++))
done
```

Combining if-else + loops (very common pattern)

Example: Print even numbers 1–20

```bash
for i in {1..20}
do
    if [ $((i % 2)) -eq 0 ]; then
        echo "$i is even"
    else
        echo "$i is odd"
    fi
done
```

Example: Find .log files bigger than 10MB and delete old ones

```bash
for file in *.log
do
    if [ -f "$file" ]; then
        size=$(stat -c %s "$file")          # size in bytes
        if [ $size -gt 10485760 ]; then     # 10MB = 10*1024*1024
            echo "Big file: $file (${size} bytes) → deleting"
            rm -v "$file"
        fi
    fi
done
```

Quick Reference Table

Loop Type	When to use	Syntax style example
`for`	Known list or range	`for i in {1..10}` or `for ((i=1;i<=10;i++))`
`while`	Condition-based (unknown iterations)	`while [ $count -lt 10 ]`
`until`	Run until something becomes true	`until [ $count -gt 10 ]`
`if-elif`	Decision making	`if [ "$a" -gt "$b" ]; then ... elif ...`

Save any script with `.sh` extension, make it executable:

```bash
chmod +x myscript.sh
./myscript.sh
```

Which loop or condition are you trying to write right now?

Feel free to share your task/use-case — I'll help you write the exact code! 😄