Comprehensive Guide to Regular Expressions (Regex) in Unix/Linux

Hey there, newbie! Welcome to the world of **regular expressions** (regex) in Unix/Linux. If you've been writing Bash scripts and using commands like `grep`, you've probably already touched regex without realizing it. Regex is one of the most powerful tools in a Unix user's toolkit — it lets you **search, match, and manipulate text patterns** in a super flexible way.

This guide is written **specifically for beginners**, just like the ones we did for loops, if-else, and input handling. We'll go step-by-step: from "what the heck is regex?" to real-world examples in common Unix tools (`grep`, `sed`, `awk`, `find`, Bash itself, etc.). No prior knowledge assumed — we'll explain everything clearly.

---

1. What Is a Regular Expression?

A **regular expression** is a **pattern** that describes a set of strings/text you want to match.

Think of it like a "smart wildcard":

- `*.txt` → matches all files ending in `.txt` (simple glob)
- Regex → can match things like "all email addresses", "phone numbers starting with +91", "lines containing a word followed by a number", etc.

Example:

- Pattern: `cat`
    - Matches: "cat", "catch", "scatter", "concatenate"
- Pattern: `^cat$`
    - Matches **only**: "cat" (exact word on its own line)

Regex is used in almost every Unix text-processing tool.

---

2. Types of Regex in Unix

Unix tools support slightly different "flavors" of regex. Know these three:

Type	Used In	Key Features	How to Enable
BRE (Basic)	Default in `grep`, `sed`, `vi`	Basic metacharacters; `\( \)` for groups	Default
ERE (Extended)	`grep -E`, `awk`, `egrep`	More convenient: `( )` for groups, `+ ?		`
PCRE (Perl-Compatible)	`grep -P`, modern tools	Most powerful: lookarounds, non-greedy, etc.	`grep -P` (not always available)

Recommendation for beginners: Start with **ERE** (`grep -E`). It's easier to read than BRE.

---

3. Core Building Blocks (Metacharacters)

These are the "special characters" that make regex magic.

A. Literal Characters

Most characters match themselves:

- `hello` → matches "hello"

B. Anchors (Position Markers)

Symbol	Meaning	Example	Matches
`^`	Start of line	`^hello`	"hello world" (at beginning)
`$`	End of line	`world$`	"hello world" (at end)
`\b`	Word boundary (ERE/PCRE)	`\bcat\b`	"cat" but not "catch"
`\B`	Not word boundary		

C. Character Classes (Match One Character)

Symbol	Meaning	Example	Matches
`.`	Any single character (except newline)	`c.t`	"cat", "cot", "c t"
`[abc]`	Any one of a, b, c	`[ch]at`	"cat", "hat"
`[^abc]`	Anything EXCEPT a, b, c	`[^ch]at`	"bat", "rat"
`[a-z]`	Range: lowercase a to z	`[a-z]at`	"bat" to "zat"
`[0-9]`	Digits	`[0-9]+`	"123"
Predefined classes:			
`[:alnum:]`	Alphanumeric		
`[:alpha:]`	Letters		
`[:digit:]`	Digits (same as [0-9])		
`[:space:]`	Whitespace		

In BRE: use inside `[ ]` like `[[:digit:]]`

D. Quantifiers (How Many?)

Tell how many times the previous thing can appear.

Symbol	Meaning	BRE Version	Example	Matches
`*`	Zero or more	Same	`ca*t`	"ct", "cat", "caaaat"
`+`	One or more	`\+`	`ca+t`	"cat", "caaaat" (not "ct")
`?`	Zero or one	`\?`	`ca?t`	"ct", "cat"
`{n}`	Exactly n times	`\{n\}`	`ca{2}t`	"caat"
`{n,}`	n or more	`\{n,\}`	`ca{2,}` t	"caat", "caaaat"
`{n,m}`	Between n and m	`\{n,m\}`	`ca{1,3}t`	"cat" to "caaaat"

Note: In ERE (`grep -E`), use `+ ? {}` directly. In BRE, escape them.

E. Alternation and Grouping

Symbol	Meaning	Example	Matches
`|`	OR (alternation) – ERE/PCRE	`cat|dog`	
`( )`	Group (capture) – ERE/PCRE	`(cat|dog)s`	
`\( \)`	Group in BRE		

Captured groups can be referenced as `\1`, `\2`, etc. in replacements (sed).

---

4. Common Unix Tools That Use Regex

A. `grep` / `egrep` / `fgrep`

Search for patterns in files/text.

```
Bash
# Basic: BRE
grep "pattern" file.txt

# Extended regex (easier)
grep -E "pat|tern" file.txt    # or egrep

# Examples
grep -E "^[0-9]{3}-[0-9]{4}" phones.txt   # Indian mobile: 123-4567 format
grep -E "\berror\b" log.txt               # exact word "error"
grep -E -i "hello" file.txt               # -i = case insensitive
grep -E -v "skip this" file.txt           # -v = invert match
```

B. `sed` (Stream Editor) – Search & Replace

```bash
# Replace first occurrence per line
sed 's/old/new/' file.txt

# Global replace
sed 's/old/new/g' file.txt

# With ERE groups
sed -E 's/(cat|dog)s/\1 was here/g' file.txt

# Delete lines matching pattern
sed '/error/d' log.txt

# Print only matching lines
sed -n '/pattern/p' file.txt
```

C. `awk` – Powerful Text Processing

```bash
awk '/pattern/ { print $0 }' file.txt     # print matching lines

awk '$1 ~ /^[0-9]+$/ { print }' file.txt   # field 1 is number

awk '{ gsub(/old/, "new"); print }' file.txt  # replace all
```

D. Bash Built-in Regex Matching

```Bash
string="hello123"

if [[ $string =~ ^[a-z]+[0-9]+$ ]]; then
    echo "Matches alphanumeric pattern"
fi

# Capture groups
if [[ $string =~ ^([a-z]+)([0-9]+)$ ]]; then
    echo "Letters: ${BASH_REMATCH[1]}"
    echo "Numbers: ${BASH_REMATCH[2]}"
fi
```

E. Other Tools

- `find -regex` / `-iregex`
- `vim` / `vi` search: `/pattern`
- `rename` command for batch renaming

---

5. Real-World Beginner Examples

1. **Find email addresses** (simple):

```
Bash

grep -E "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" emails.txt
```

2. **Validate Indian phone number** (+91 followed by 10 digits):

```
Bash

if [[ "$$   phone" =~ ^\+91[0-9]{10}   $$ ]]; then
    echo "Valid"
fi
```

3. **Replace dates** from MM/DD/YYYY to DD-MM-YYYY:

```
Bash

sed -E 's|([0-9]{2})/([0-9]{2})/([0-9]{4})|\2-\1-\3|g' dates.txt
```

4. **Extract numbers from log**:

```
Bash

grep -E -o "[0-9]+" error.log
```

---

6. Best Practices & Common Pitfalls for Newbies

- **Start simple**: Test patterns on small text.
- **Use online testers**: regex101.com (choose "ECMAScript" or "PCRE" flavor).
- **Escape when needed**: In BRE, escape `+ ? ( ) { }` with `\.`
- **Greedy vs Lazy**: `*` is greedy (matches as much as possible). Use `*?` in PCRE for lazy.
- **Performance**: Very complex regex on huge files can be slow — sometimes combine with other tools.
- **Quote patterns**: `grep "pattern with |" file` or use single quotes.
- **Case sensitivity**: Default is sensitive — use `-i` for ignore case.

Quick Reference Cheat Sheet

Task	Command Example
Search exact word	`grep -w "word" file`
Search with OR	`` `grep -E "cat ``
Replace all occurrences	`sed 's/old/new/g' file`
Case-insensitive search	`grep -i "pattern" file`
Print line numbers	`grep -n "pattern" file`
Validate integer in Bash	`[[ "$var" =~ ^-?[0-9]+$ ]]`

You've now got a solid foundation in Unix regex! Practice by opening a terminal and trying these on some text files.

Got a specific pattern you're trying to write (like validating Aadhaar, extracting URLs, cleaning logs)? Share the use case or sample text — I'll help you build the perfect regex! 😊