Comprehensive Guide to the `cut` Command in Unix/Linux

The `cut` command is one of the classic Unix text-processing utilities — it's simple, fast, and incredibly useful for extracting specific parts of lines from files or piped input. Whether you're pulling columns from a CSV, grabbing usernames from `/etc/passwd`, or slicing fixed-width data, `cut` is often the quickest tool for the job.

This guide covers **every aspect** of `cut`: full syntax, all options, range selection, delimiters, advanced features, real-world patterns, pitfalls, best practices, and even differences across systems (GNU vs BSD/macOS). Everything here works on modern Linux (GNU coreutils) unless noted otherwise.

We'll focus on practical usage with plenty of examples.

---

1. Basic Purpose & Syntax

`cut` reads input (from files or stdin) line by line and outputs selected portions of each line.

General Syntax:

```
Bash
cut OPTION... [FILE]...
```

- If no FILE is given (or `-` as filename), reads from **stdin** (perfect for pipelines).
- Processes **one line at a time** — no context between lines.
- Output goes to stdout.

Three mutually exclusive modes (you pick one):

Mode	Option	What it cuts by	Use case
Bytes	`-b`	Byte positions	Binary or multibyte (UTF-8) data
Chars	`-c`	Character positions	Unicode-safe text (default in most)
Fields	`-f`	Delimiter-separated fields	CSV/TSV, logs, /etc/passwd

You **must** specify a list/ranges with the chosen mode (e.g., `-b 1-5`).

---

2. Selecting Positions / Ranges

The **list** argument (used with `-b`, `-c`, or `-f`) defines what to extract. Syntax is identical for all modes.

Range Formats:

Format	Meaning	Example (`-c`)	Output from "hello world"
`N`	Exact position N (1-based)	`cut -c 1`	"h"
`N-`	From N to end of line	`cut -c 6-`	"world"
`-N`	From start to N (inclusive)	`cut -c -5`	"hello"
`N-M`	From N to M (inclusive)	`cut -c 1-5`	"hello"
Multiple	Comma-separated	`cut -c 1,6-`	"hworld"

- Positions are **1-based** (not zero).
- Ranges are **inclusive**.
- Order doesn't matter — output preserves original order.
- No spaces allowed in list (use commas only).

Example input file `data.txt`:

```
text

1234567890
abcdefghij
```

```
Bash

cut -c 1-3 data.txt      # → 123 \n abc
cut -c 5- data.txt       # → 567890 \n efghij
cut -c -4,8 data.txt     # → 12348 \n abcdh
```

---

3. Character vs Byte Mode (`-c` vs `-b`)

Option	Counting Unit	Unicode/Multibyte Safe?	When to Use
`-c`	Characters	Yes	Text with UTF-8 (emojis, non-ASCII)
`-b`	Bytes	No (can split chars)	Fixed-width binary, ASCII-only, or legacy

Important: In UTF-8, one character can be multiple bytes.

Example with "café" (é is 2 bytes):

```
Bash

echo "café" | cut -c 4    # → é (correct)
echo "café" | cut -b 4    # →  (garbled — partial byte)
```

Rule: Use `-c` for text unless you specifically need byte-level (rare).

---

4. Field Extraction Mode (`-f` with `-d`)

This is the **most common** use case — cutting columns from delimited data.

Key options:

Option	Meaning	Example
`-f LIST`	Select fields (1-based)	`-f 1,3`
`-d DELIM`	Use DELIM instead of TAB as field separator	`-d ','` for CSV
`-s`	Suppress (skip) lines without delimiter	Useful for clean data
`--output-delimiter=STRING`	Change output field separator (GNU only)	`--output-delimiter=':'`
`--complement`	Output everything EXCEPT selected fields	Invert selection (GNU only)

Default: delimiter is TAB, fields are numbered starting at 1.

Classic example — usernames from `/etc/passwd` (colon-delimited):

```
Bash

cut -d ':' -f 1 /etc/passwd
# Output: root \n daemon \n bin \n ...
```

CSV example (`data.csv`):

```
text

name,age,city
Alice,30,New York
Bob,25,London
```

```
Bash

cut -d ',' -f 1,3 data.csv
# → name,city \n Alice,New York \n Bob,London
```

With `--output-delimiter` :

```Bash
cut -d ',' -f 1,3 --output-delimiter=' -> ' data.csv
# → name -> city \n Alice -> New York \n ...
```

Complement example (GNU only):

```Bash
cut -d ':' -f 1 --complement /etc/passwd   # everything except username
```

---

5. Real-World Examples

1. **Extract PID from** `ps` **output**:

```Bash
ps aux | cut -d ' ' -f 2     # messy because multiple spaces
# Better: use awk, but cut works with -s if needed
```

1. **Get IP from** `ifconfig` **/** `ip` **(older systems)**:

```Bash
ip addr show eth0 | grep 'inet ' | cut -d ' ' -f 6 | cut -d '/' -f 1
```

1. **Process log files** (space-delimited timestamp):

```Bash
tail -f access.log | cut -d ' ' -f 1,4-   # IP and rest of line
```

1. **Fixed-width data** (e.g., old reports):

```text
001John   Engineer  50000
002Alice  Manager   80000
```

```
Bash

cut -c 1-3 file.txt   # IDs: 001 \n 002
cut -c 4-10 file.txt  # Names (padded)
cut -c 11-20 file.txt # Job title
```

1. **Pipeline with other tools**:

```
Bash

ls -l | cut -c 11-   # filenames only (skip permissions/size)
```

---

6. Advanced / GNU-Specific Features

Option	Description	Example
`--complement`	Invert selection	`cut -f 2 --complement -d ':' /etc/passwd`
`--output-delimiter=STR`	Custom separator in output	As above
`--only-delimited` (same as `-s`)	Skip lines without delimiter	
`-z, --zero-terminated`	Lines ended by NUL instead of newline	Rare (binary data)

---

7. Common Pitfalls & Best Practices

- **Multiple spaces as delimiter**: `cut` treats consecutive delimiters as one empty field. Use `awk` for robust space handling.
- **No regex support**: `cut` is literal only. Use `awk` / `sed` for patterns.
- **Trailing newline**: `cut` preserves line endings.
- **Empty fields**: Consecutive delimiters → empty output field.
- **Performance**: Extremely fast — great for huge files.
- **Portability**: GNU cut (Linux) has `--complement` and `--output-delimiter`. BSD/macOS cut lacks them → scripts may break.
    - macOS workaround for complement: use `awk` instead.
- **Always quote delimiter** if special: `-d $'\t'` for tab.

Best practice: For simple column extraction → `cut`. For anything more complex (variable spaces, regex, math) → `awk`.

---

8. Quick Reference Table (All Options)

Option	Long Form	Meaning
`-b LIST`	`--bytes=LIST`	Select bytes
`-c LIST`	`--characters=LIST`	Select characters
`-f LIST`	`--fields=LIST`	Select fields
`-d DELIM`	`--delimiter=DELIM`	Field delimiter
`-s`	`--only-delimited`	Suppress lines without delimiter
`--complement`		Output non-selected parts (GNU only)
`--output-delimiter=STRING`		Change output separator (GNU only)
`-z`	`--zero-terminated`	NUL-terminated lines
`-n`	(with -b)	Don't split multibyte chars (obsolete)

You've now got **complete mastery** of `cut` — from basic column slicing to advanced GNU features.

What's your use case? Extracting from CSV? Processing logs? Fixed-width reports? Share a sample line/file snippet and I'll give you the perfect `cut` command (or suggest if awk/sed is better)! 😊