grep Command Cheat Sheet for Linux

From Ubuntu Free — Get more Linux Cheat Sheets, free!

Search files, match patterns, filter output, and script with confidence.

1. Basics

grep searches input for lines that match a pattern. By default it prints matching lines.

```
# Basic search (case sensitive)
grep "error" /var/log/syslog

# Case insensitive
grep -i "error" /var/log/syslog

# Search multiple files
grep "timeout" *.log

# Read from stdin
dmesg | grep -i "usb"

# Show line numbers
grep -n "main" src/*.c

# Only show filenames with a match
grep -l "TODO" -R .

# Invert match (show non matching lines)
grep -v "^#" config.ini
```

Tip: If your pattern begins with a dash, use `-e` to avoid confusion with options: `grep -e "--start"`.

grep stdin case sensitivity line numbers

2. Common flags

- `-i` case insensitive
- `-v` invert match
- `-r/-R` recursive (`-R` follows symlinks)
- `-n` show line numbers
- `-H/-h` force show/hide filenames
- `-c` count matches per file
- `-l/-L` list files with/without matches
- `-o` print only the matching part
- `-w` match whole words
- `-x` match whole lines
- `-E` extended regex, `-F` fixed strings, `-P` PCRE (Perl style)
- `-A/-B/-C` context after/before/both
- `--color=auto` colorize matches
- `--exclude/--include/--exclude-dir` filter files and dirs
- `-Z/--null` NUL separator for filenames
- `-z/--null-data` input lines are NUL terminated
- `-a/-I` treat binary as text / ignore binary files

3. Patterns and regex

3.1 Anchors and boundaries

```
# Start and end of line
grep "^ERROR" app.log
grep "timeout$" app.log

# Word boundaries (portable)
grep -E "[[:<:]]fail[[:>:]]" app.log

# Word boundaries (PCRE)
grep -P "\bfail\b" app.log
```

3.2 Character classes

```
# POSIX classes
grep -E "[[:digit:]]{3}" data.txt     # any 3 digits
grep -E "[[:alpha:]_]+" names.txt
grep -E "^[[:space:]]$" file.txt      # all whitespace lines
```

3.3 Quantifiers and groups

```
# Extended regex with -E
grep -E "colou?r" file.txt     # color or colour
grep -E "warning|hint" file.txt
grep -E "(foo|bar|baz)" file.txt
grep -E "ab{2,4}c" file.txt    # abb c to abbbc
```

3.4 Backreferences and lookarounds

```
# Backreference (ERE extension in ERE, widely works)
grep -E "^(.)\1$" duplicates.txt     # repeated substring lines

# Lookahead or lookbehind (requires -P)
grep -P "foo(?=\s+bar)" file.txt     # foo followed by bar
grep -P "(?<=export\s+)\w+" config.txt   # extract after user=
```

Use `-F` for literal strings. It is faster and avoids regex pitfalls.

4. Context and formatting

```
# Show 3 lines after each match
grep -A 3 "panic" kernel.log

# Show 2 lines before each match
grep -B 2 "panic" kernel.log

# Show 1 line before and after
grep -C 1 "panic" kernel.log

# Colorize matches when writing to a terminal
grep --color=auto -n "error" app.log

# Pipe to less and preserve color
grep --color=always -n "error" app.log | less -R
```

5. Files and directories

```
# Recursive search in current directory
grep -R "nginx" .

# Recursively search only certain file types
grep -R --include="*.{c,h}" "malloc" src/

# Exclude directories or files
grep -R --exclude-dir=".git" --exclude="*.min.js" "TODO" .

# Follow symlinks
grep -R "pattern" /path

# Ignore binary files
grep -IR "magic-bytes" /opt/data
```

With find and xargs

```
# Use find to target files, then grep
find . -type f -name "*.log" -print0 | xargs -0 grep -n "fatal"

# Handling spaces safely with NUL separators
find /etc -type f -print0 | xargs -0 grep -H "PermitRootLogin"
```

6. Output control

```
# Only filenames that contain matches
grep -Rl "api_key" .

# Only filenames with no matches
grep -RL "strict-transport-security" /etc/nginx/sites-available

# Count matches per file
grep -Rc "404" logs/

# Print only the matching part of lines
grep -oE "[0-9]{3}" status.log

# Suppress filename in output when multiple files are searched
grep -h "pattern" file1 file2

# Show filename even for single file
grep -H "pattern" file.txt

# Null-terminate filenames for safer piping
grep -Z3 "needle" -R . | tr '\0' '\n'
```

7. Extracting data (emails, IPs, URLs)

```
# Extract IPv4 addresses
grep -oE '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' access.log

# Extract likely email addresses
grep -oE '[[:alnum:]._%+-]+@[[:alnum:].-]+\.[a-zA-Z]{2,}' users.txt

# Extract http/https URLs
grep -oE 'https?://[^ ]+' page.html

# Extract ISO dates (YYYY-MM-DD)
grep -oE '\b[0-9]{4}-[0-9]{2}-[0-9]{2}\b' records.txt

# Extract JSON keys
grep -oE '"[^"]+":' data.json | cut -d'"' -f2
```

Real world data is messy. Validate with additional tools like `awk`, `jq` or custom scripts.

8. Practical recipes

Logs and observability

```
# Tail logs and highlight errors
tail -f app.log | grep --line-buffered --color=always -E "ERROR|WARN"

# Count unique 404 paths
grep " 404 " access.log | awk '{print $7}' | sort | uniq -c | sort -nr | head

# Find requests slower than 2s (Nginx combined logs)
awk '$NF > 2 {print}' access.log | grep -n ""
```

Code search

```
# Find function definitions in C
grep -R --include="*.c" -nE "^[A-Za-z_][A-Za-z0-9_]*\s*\(" src/

# TODOs except vendor dir
grep -R --exclude-dir={vendor,node_modules}* -n "TODO" .

# Find strings that look like secrets
grep -R -nE "(api[_-]?key|secret|token|password)" .
```

System administration

```
# SSH config checks
grep -nE '^(PermitRootLogin|PasswordAuthentication)' /etc/ssh/sshd_config

# Kernel messages with USB
dmesg | grep -i "usb"

# Services failing recently (journalctl)
journalctl -p err -S -2h | grep -n ""
```

Pipelines and process lists

```
# Process search excluding the grep process itself
ps aux | grep "[c]hrome"

# Network sockets for a process name
ss -tulpn | grep -i "nginx"

# Top memory consumers by process name filter
ps aux | grep -i "java" | sort -k4 -nr | head
```

Multiple patterns

```
# Any of several words
grep -E "urgent|critical|fatal" app.log

# Lines containing both foo and bar (order does not matter)
grep -E "foo.*bar|bar.*foo" file.txt

# Provide multiple -e options
grep -e "foo" -e "bar" -e "baz" file.txt
```

Whole word and whole line

```
# Whole word match
grep -w "error" file.txt

# Whole line match
grep -x "READY" status.txt
```

Binary handling

```
# Treat binary as text to search for ASCII fragments
grep -a "PNG" image.png

# Ignore binary files entirely when recursing
grep -IR "signature" /path
```

9. Scripting and exit codes

Exit codes: 0 match found, 1 no match, 2 error.

```
# Simple condition
if grep -q "READY" status.txt; then
    echo "Service is ready"
else
    echo "Not ready"
fi

# Fail a script if a forbidden pattern is present
if grep -R -nE "(FIXME|HACK)" src/; then
  echo "Forbidden markers found" >&2
  exit 1
fi

# Use -Z for safe piping of filenames with NUL separators
grep -lZ "needle" -R . | while IFS= read -r -d '' f; do
  echo "Found in: $f"
done
```

Performance sensitive scripting

```
# Use -F for fixed strings and set C locale for speed
LC_ALL=C grep -rF "needle" .

# Many patterns from a file
grep -Ff patterns.txt bigfile.txt

# Quiet mode in conditions
if grep -qE "needle" file.txt; then
  do_something
fi
```

10. Performance and safety

- Prefer `-F` for literal strings - it is faster and avoids regex surprises.
- Use `--include/--exclude` to limit file sets in recursive searches.
- Locale can impact speed. `LC_ALL=C` speeds up byte-wise searches for ASCII logs.
- Color and paging: `--color=always | less -R` keeps color in pagers.
- **Safety:** handle filenames with spaces using NUL separators (`-Z`, `-print0`, -0).
- PCRE with `-P` can be powerful but may vary by system. Have a fallback using `-E` where possible.

Be careful with `grep -r .` at filesystem root. It can be slow and noisy. Limit scope with target directories and `--exclude-dir`.

11. Quick reference

Task	Command
Case insensitive search	`grep -i "error" file`
Recursive search	`grep -R "pattern" dir/`
Whole word	`grep -w "word" file`
Whole line	`grep -x "exact line" file`
Show line numbers	`grep -n "needle" file`
Only filenames with matches	`grep -l "needle" *`
Only the match text	`grep -oE "re-gex" file`
Before/after context	`grep -B 2 -A 2 "needle" file`
Count matches	`grep -c "needle" file`
Fixed strings (fast)	`grep -F "literal" file`
Extended regex	`grep -E "a|b|c" file`
Perl regex	`grep -P "\bword\b" file`
Include specific extensions	`grep -R --include="*.log" "needle" .`
Exclude directory	`grep -R --exclude-dir=".git" "needle" .`
Ignore binary files	`grep -IR "needle" .`
Null-terminated filenames	`grep -lZ "needle" -R .`

12. Troubleshooting

- **No matches but you expect some**: check case sensitivity, quoting, and whether the pattern is interpreted as regex. Try `-F`.
- **Unicode characters**: ensure your locale is set correctly or use `LC_ALL=C` for byte-wise matching.
- **Pattern starts with a dash**: use `-e` like `grep -e "-foo"`.
- **Filtering grep from pipelines**: use the bracket trick `grep '[p]ing'` to avoid matching the grep process itself.
- **Slow recursive search**: constrain with `--include=` `--exclude`, and consider `-F` when possible.

Want more? Visit Ubuntu Free for more Linux cheat sheets.

Ubuntu Free - practical guides and free cheat sheets for Linux users.

Share this cheat sheet with your team. Feedback helps us improve.