Text Processing (Core → Plus)

Linux Commands Course · Section 6

IDSchool

Goal

Learn how to **extract, filter, transform, and summarize text** using command-line tools.

You’ll move from basic searches to structured reporting and automation-ready processing.

Filtering Lines — grep

grep searches for patterns inside text files or input streams.

```
grep "root" /etc/passwd
```

Shows all lines containing “root”.

Case-insensitive search:

```
grep -i "bash" /etc/passwd
```

Show line numbers:

```
grep -n "student" /etc/passwd
```

Recursive search through directories:

```
grep -R "error" /var/log
```

Regular Expressions (regex)

grep -E enables extended regex for more expressive matching.

Examples:

```
grep -E "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+$" emails.txt
```

→ matches email-like lines.

Regex basics:

Symbol	Meaning
.	any single character
^	start of line
$	end of line
[]	character class
*, +, ?	repetition quantifiers
`	`

Use -v to invert (show non-matching lines).

Extracting Columns — cut

Split text into fields and extract specific columns.

```
cut -d: -f1,7 /etc/passwd
```

→ prints username and shell columns.

Here, `-d:` sets delimiter to `:` and `-f` specifies which fields to output.

Extract fixed-width positions:

```
cut -c1-10 filename.txt
```

Transforming Characters — tr

tr replaces, deletes, or squeezes characters.

Uppercase to lowercase:

```
cat names.txt | tr '[:upper:]' '[:lower:]'
```

Remove digits:

```
cat file.txt | tr -d '0-9'
```

Replace spaces with tabs:

```
cat file.txt | tr ' ' '	'
```

Sorting and Uniqueness – sort, uniq

Sort alphabetically:

```
sort names.txt
```

Sort numerically and by human sizes:

```
sort -h sizes.txt
```

Eliminate duplicates (must be sorted first):

```
sort names.txt | uniq
```

Count repeated lines:

```
sort names.txt | uniq -c | sort -nr
```

Shows how many times each entry occurs.

Editing Streams – sed

sed edits text as it flows through a pipeline.

Substitute “foo” with “bar”:

```
sed 's/foo/bar/' file.txt
```

Replace globally on each line:

```
sed 's/foo/bar/g' file.txt
```

In-place modification:

```
sed -i 's/error/ERROR/g' logfile.txt
```

Delete specific lines (e.g., 2–4):

```
sed '2,4d' file.txt
```

Print only certain lines:

```
sed -n '1,5p' file.txt
```

Reporting Language — awk

awk is a text-based data extraction and reporting DSL.

Print the first field of each line:

```
awk -F: '{print $1}' /etc/passwd
```

Use multiple fields and text:

```
awk -F: '{print "User:", $1, "Shell:", $7}' /etc/passwd
```

Conditionals:

```
awk -F: '$3 >= 1000 {print $1, $3}' /etc/passwd
```

Perform arithmetic and aggregation:

```
awk '{sum += $2} END {print "Total:", sum}' data.txt
```

Power Combinations – xargs

xargs converts input lines into command arguments.

Example – delete found files:

```
find . -name "*.tmp" | xargs rm -v
```

Count lines of all .txt files:

```
ls *.txt | xargs wc -l
```

Safer with spaces:

```
find . -name "*.txt" -print0 | xargs -0 wc -l
```

Process Substitution <()

Run two commands in parallel and compare results without temporary files.

```
diff <(sort a.txt) <(sort b.txt)
```

Also useful with join, comm, or paste to feed preprocessed data.

Encoding Tools – iconv, dos2unix

Convert between character encodings with iconv:

```
iconv -f ISO-8859-1 -t UTF-8 old.txt -o new.txt
```

Fix Windows line endings (CRLF) in text files:

```
dos2unix script.sh
```

Makes scripts compatible on Linux systems.

JSON

JSON is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of name–value pairs and arrays. It is a commonly used data format with diverse uses in electronic data interchange, including that of web applications with servers.

```json
{
    "name": "Elnur",
    "job": [
        "Teacher",
        "Cyber Security Engineer"
    ],
    "age": 22
}
```

JSON Processing — jq

jq is a lightweight command-line JSON processor.

Format JSON neatly:

```
jq . data.json
```

Extract specific fields:

```
jq '.users[].name' data.json
```

Filter with conditions:

```
jq '.users[] | select(.age > 25)' data.json
```

Combine with other commands:

```
curl -s https://api.github.com/users/torvalds | jq '.name, .public_repos'
```

Practical Example

Count how many users use /bin/bash:

```
grep '/bin/bash' /etc/passwd | cut -d: -f1 | wc -l
```

Or display usernames sorted by shell:

```
awk -F: '{print $7, $1}' /etc/passwd | sort
```

Convert all text to uppercase while filtering certain lines:

```
grep "info" logs.txt | tr '[:lower:]' '[:upper:]' | tee filtered.txt
```

Recap

- **grep** – match/filter text using regex
- **cut**, **tr**, **sort**, **uniq** – extract and transform columns
- **sed** – substitute or delete text patterns
- **awk** – structured reporting and logic
- **xargs**, **<()** – advanced composition
- **iconv**, **dos2unix**, **jq** – encoding and JSON utilities

Together, these make Linux text processing infinitely flexible.

Practice

1. Print only usernames from `/etc/passwd` using `cut`.
2. Find all lines containing “error” in `/var/log/syslog`.
3. Replace “failed” with “FAILED” in-place using `sed -i`.
4. Print fields 1 and 7 of `/etc/passwd` with `awk`.
5. Use `iconv` to convert a file from Latin-1 to UTF-8.
6. Parse JSON output from an API using `jq`.
7. Combine `grep`, `tr`, and `tee` into one pipeline to create uppercase filtered logs.

Next Up

Archiving & Compression (Core) — tar, gzip, zip, and beyond.