Table of Contents

$ iconv

Intermediate

Convert text between character encodings

The iconv command converts text from one character encoding to another. It handles conversions between hundreds of encoding formats including UTF-8, UTF-16, ASCII, ISO-8859-1 (Latin-1), Windows-1252, Shift_JIS, EUC-KR, GB2312, and many more.

Character encoding issues are among the most common pr...

Options & Flags

`-f ENCODING`	Source (from) encoding
`-t ENCODING`	Target (to) encoding
`-o FILE`	Write output to file instead of stdout
`-l`	List all supported encodings
`-c`	Silently discard characters that cannot be converted
`//TRANSLIT`	Transliterate characters that cannot be converted
`//IGNORE`	Ignore characters that cannot be converted

Practical Examples

Example: Convert Windows to UTF-8

```
$ iconv -f Windows-1252 -t UTF-8 report.csv > report-utf8.csv
```

Convert a Windows-encoded CSV file to UTF-8. Common when processing Excel exports on Linux.

Example: Convert Latin-1 to UTF-8

```
$ iconv -f ISO-8859-1 -t UTF-8 -o output.txt input.txt
```

Convert ISO-8859-1 (Latin-1) text to UTF-8. Use -o for in-place-like conversion.

Example: Detect encoding first

```
$ file -bi document.txt && chardetect document.txt
text/plain; charset=iso-8859-1
```

Detect the encoding before converting. file -bi shows MIME type with charset. chardetect (Python) provides confidence score.

Example: Convert to ASCII with transliteration

```
$ echo "cafÃ© rÃ©sumÃ© naÃ¯ve" | iconv -f UTF-8 -t ASCII//TRANSLIT
cafe resume naive
```

Convert accented characters to their ASCII approximations.

Example: Strip non-ASCII characters

```
$ iconv -f UTF-8 -t ASCII//IGNORE < input.txt > clean.txt
```

Remove all non-ASCII characters. Useful for cleaning data for systems that only support ASCII.

Tips & Best Practices

Pro Tip: Always detect encoding first: Use file -bi file.txt to detect encoding before converting. Converting from the wrong source encoding corrupts the output. Install chardet for better detection.

Warning: Cannot convert in-place: iconv cannot read and write the same file. Always write to a new file: iconv -f X -t Y input.txt > output.txt && mv output.txt input.txt

Note: UTF-8 is the standard: UTF-8 is the universal encoding for modern systems. When in doubt, convert everything to UTF-8. It supports all languages and is backwards-compatible with ASCII.

Pro Tip: Use //TRANSLIT for lossy conversion: When converting to ASCII, use //TRANSLIT to approximate characters (é?e, ü?u) instead of failing or dropping them.

$ jq

Intermediate

Command-line JSON processor

The jq command is a lightweight and flexible command-line JSON processor. It is like sed, awk, and grep for JSON data - allowing you to slice, filter, map, and transform structured data with ease.

jq is essential for modern system administration and DevOps work, where JSON is the standard format...

Options & Flags

`.`	Identity filter - output the entire input unchanged
`.key`	Extract a specific key from an object
`-r`	Raw output - strip quotes from strings
`-c`	Compact output - no pretty printing
`-e`	Exit with error if output is null or false
`-s`	Slurp - read all inputs into an array
`-S`	Sort keys in objects alphabetically
`--arg name val`	Pass a string variable into the jq program
`-n`	Null input - do not read any input
`--rawfile name file`	Read file contents into a variable as string

Practical Examples

Example: Extract a field from JSON

```
$ echo '{"name":"nginx","version":"1.25"}' | jq -r .name
nginx
```

Extracts the name field and outputs raw text without quotes.

Example: Get nested values

```
$ echo '{"server":{"host":"10.0.0.1","port":8080}}' | jq '.server.port'
8080
```

Access nested objects using dot notation.

Example: Filter array elements

```
$ echo '[{"name":"a","active":true},{"name":"b","active":false}]' | jq '.[] | select(.active==true) | .name'
"a"
```

Iterate array elements, filter by condition, and extract field.

Example: Transform JSON structure

```
$ echo '{"first":"John","last":"Doe"}' | jq '{fullname: (.first + " " + .last)}'
{"fullname": "John Doe"}
```

Create a new JSON object with computed values.

Example: Parse curl API response

```
$ curl -s https://api.github.com/repos/jqlang/jq | jq '{name: .name, stars: .stargazers_count, language: .language}'
```

Fetch JSON from an API and extract specific fields into a new object.

Tips & Best Practices

Pro Tip: Always use -r for scripting: When assigning jq output to shell variables, use -r to get raw strings without quotes: VERSION=$(jq -r .version package.json)

Warning: Null handling: jq returns "null" (string) for missing keys. Use the // operator for defaults: jq '.missing // "default"' to avoid null in your scripts.

Note: jq as a JSON validator: Use jq . file.json to validate and pretty-print JSON. If the file has syntax errors, jq will report the exact error location.

Pro Tip: Combine with other tools: jq works great in pipelines: kubectl get pods -o json | jq '.items[] | {name: .metadata.name, status: .status.phase}'

$ less

Beginner

View file contents with scrolling and search

less is an interactive file viewer that displays text one screen at a time. Unlike cat which dumps entire files, less allows scrolling, searching, and navigating through files of any size efficiently.

less loads files lazily - it does not need to read the entire file before displaying, making it...

Options & Flags

`-N`	Show line numbers
`-S`	Chop (truncate) long lines instead of wrapping
`-i`	Case-insensitive search
`-F`	Quit if file fits in one screen
`-R`	Display ANSI color codes (raw control characters)
`+F`	Follow mode (like tail -f, press Ctrl+C to stop)
`-X`	Do not clear screen on exit

Practical Examples

Example: View a file

```
$ less /var/log/syslog
```

Opens the file for interactive viewing. Use q to quit.

Example: View with line numbers

```
$ less -N /etc/nginx/nginx.conf
```

Displays the file with line numbers on the left side.

Example: Search within file

```
$ less /var/log/auth.log\n/failed
```

Open file, then type /failed to search forward. Use n for next match, N for previous.

Example: Follow a growing log

```
$ less +F /var/log/syslog
```

Follows the file like tail -f. Press Ctrl+C to stop following and browse normally.

Example: View colored output

```
$ git log --oneline --graph --color | less -R
```

Preserves ANSI color codes in piped output for readable display.

Tips & Best Practices

Pro Tip: Essential navigation keys: Space/f = page down, b = page up, g = start, G = end, /pattern = search forward, ?pattern = search backward, n = next match, q = quit.

Note: Mark positions: Press m followed by a letter to mark a position. Press ' followed by the same letter to return to it. Great for navigating large files.

Pro Tip: Highlight all matches: After searching with /pattern, all matches stay highlighted as you scroll. Use ESC-u to toggle highlighting off.

$ more

Beginner

View file contents one screen at a time

The more command is a simple file viewer that displays text one screenful at a time. It is the predecessor to less, offering basic forward pagination through files and piped output.

more allows you to scroll forward through a file, search forward for patterns, and jump to specific positions. How...

Options & Flags

`-d`	Display help prompt instead of bell on invalid input
`-f`	Count logical rather than screen lines
`-p`	Clear screen before displaying each page
`-s`	Squeeze multiple blank lines into one
`-n`	Display N lines per screenful
`+/pattern`	Start display at first occurrence of pattern

Practical Examples

Example: Page through a file

```
$ more /etc/services
```

Displays the file one screen at a time. Press Space to advance.

Example: Start at a pattern

```
$ more +/ssh /etc/services
```

Opens the file scrolled to the first line matching "ssh".

Example: Page through command output

```
$ ls -la /usr/bin | more
```

Paginates a long directory listing for easy reading.

Example: Squeeze blank lines

```
$ more -s document.txt
```

Compresses multiple consecutive blank lines into a single blank line.

Example: Start at specific line

```
$ more +100 largefile.txt
```

Starts displaying from line 100.

Tips & Best Practices

Pro Tip: Navigation keys: Space = next page, Enter = next line, b = back one page (some implementations), /pattern = search, q = quit.

Note: Use less instead: In most situations, less is superior to more. It supports backward scrolling, better search, and does not read the entire file into memory.

Warning: No backward scroll in pipes: When more receives piped input, backward scrolling is not possible even in implementations that support it for files.

$ nl

Beginner

Number lines of files

The nl command numbers lines of files, providing more formatting control than cat -n. It supports different numbering styles, formats, and can selectively number only non-empty lines or lines matching specific patterns.

nl divides input into logical sections (header, body, footer) and can apply ...

Options & Flags

`-b a`	Number all lines including blank lines
`-b t`	Number only non-empty lines (default)
`-b p`	Number only lines matching a regex pattern
`-n ln`	Left-justified line numbers
`-n rz`	Right-justified with leading zeros
`-w`	Set width of line number field
`-s`	Set separator string after number

Practical Examples

Example: Number all lines

```
$ nl -ba script.sh
     1 #!/bin/bash
     2
     3 echo "Hello"
```

Numbers every line including blank lines.

Example: Number with leading zeros

```
$ nl -nrz -w4 data.txt
0001 First line
0002 Second line
```

Right-justified numbers with leading zeros, 4 digits wide.

Example: Number only non-empty lines

```
$ nl config.conf
```

Numbers only lines with content, leaving blank lines unnumbered (default).

Example: Custom separator

```
$ nl -s') ' -w2 todo.txt
 1) Buy groceries
 2) Clean house
```

Uses ") " as separator for numbered list formatting.

Example: Number matching lines only

```
$ nl -bp'^def ' module.py
```

Numbers only lines that start with "def " (function definitions).

Tips & Best Practices

Pro Tip: Use nl for numbered lists: nl -s'. ' -w2 creates numbered lists like 1. First item. Use in scripts to format output.

Note: nl vs cat -n: cat -n is simpler but numbers all lines with fixed format. nl offers customizable numbering style, width, separator, and selective numbering.

Warning: Default skips blank lines: By default, nl does not number blank lines. Use -ba if you need all lines numbered (like cat -n behavior).

$ paste

Intermediate

Merge lines of files side by side

The paste command merges lines from multiple files side by side, combining corresponding lines with a delimiter (tab by default). It is the complement to cut - while cut extracts columns, paste joins them.

paste can also serialize a file by converting its lines into columns, effectively transpos...

Options & Flags

`-d`	Set delimiter(s) between merged fields
`-s`	Serialize - paste lines of each file in sequence
`-z`	Use null character as line delimiter
`-`	Read from stdin (use - for each column)
`-d list`	Cycle through multiple delimiters
`multiple files`	Merge corresponding lines from N files

Practical Examples

Example: Merge two files side by side

```
$ paste names.txt scores.txt
Alice 95
Bob 87
```

Combines corresponding lines from both files, separated by tab.

Example: Use comma as delimiter

```
$ paste -d',' first.txt last.txt email.txt
John,Doe,john@example.com
```

Creates CSV-like output from three separate files.

Example: Convert list to single line

```
$ paste -sd, names.txt
Alice,Bob,Charlie,Dave
```

Serializes all lines into one comma-separated line.

Example: Create columns from stream

```
$ seq 9 | paste - - -
1 2 3
4 5 6
7 8 9
```

Groups sequential numbers into 3 columns using stdin placeholders.

Example: Merge with numbered lines

```
$ seq 5 | paste -d': ' - - < names.txt
```

Combines line numbers with file content.

Tips & Best Practices

Pro Tip: Transpose rows to columns: Use paste -s to convert a column of data into a single row. This is useful for creating comma-separated lists: paste -sd, file.txt

Note: paste vs join: paste merges by line position (line 1 with line 1). join merges by matching key values (like SQL JOIN). Use join when files share a common key column.

Warning: Uneven files: If files have different numbers of lines, paste uses empty strings for the shorter file. This can create trailing delimiters.

$ rev

Beginner

Reverse characters in each line of a file

The rev command reverses the character order of each line in a file or input stream. Each line is reversed independently - the first character becomes the last and vice versa.

rev is useful in text processing pipelines for tasks like extracting file extensions, reversing field order in combinati...

Options & Flags

`(no flags)`	rev has no flags - it simply reverses each line
`file`	Reverse lines from a file
`stdin`	Reverse lines from piped input
`multiple files`	Process multiple files sequentially
`with cut`	Common pattern: extract last field
`palindrome check`	Compare original with reversed to check palindromes

Practical Examples

Example: Reverse a string

```
$ echo "Hello World" | rev
dlroW olleH
```

Reverses the characters in the string.

Example: Extract file extension

```
$ echo 'archive.tar.gz' | rev | cut -d'.' -f1 | rev
gz
```

Gets the last extension by reversing, cutting the first field, and reversing back.

Example: Get filename from path

```
$ echo '/var/log/nginx/access.log' | rev | cut -d'/' -f1 | rev
access.log
```

Extracts the filename from a full path using rev+cut pattern.

Example: Reverse each line in a file

```
$ rev /etc/hostname
```

Reverses the characters of each line in the file.

Example: Check for palindrome

```
$ echo "racecar" | rev
racecar
```

If the output matches the input, the word is a palindrome.

Tips & Best Practices

Pro Tip: Last field extraction: The rev | cut -d'x' -f1 | rev pattern extracts the last delimited field. This is one of the most useful rev tricks.

Note: rev vs tac: rev reverses characters within each line. tac reverses the order of lines in a file. Different tools for different reversal tasks.

Warning: Multi-byte characters: rev may not correctly handle multi-byte UTF-8 characters on all implementations. Test with your locale if working with non-ASCII text.

$ sed

Intermediate

Stream editor for filtering and transforming text

sed (stream editor) is a powerful non-interactive text editor that processes text line by line. It is primarily used for find-and-replace operations, text transformations, and line-level editing in files and pipelines.

sed reads input, applies editing commands, and writes the result to standard ...

Options & Flags

`-i`	Edit files in place (modify the original file)
`-i.bak`	Edit in place with backup
`-e`	Add multiple editing commands
`-n`	Suppress automatic output (use with p command)
`-E (-r)`	Use extended regular expressions
`s///g`	Substitute command with global flag (replace all)
`d`	Delete matching lines

Practical Examples

Example: Simple find and replace

```
$ sed 's/foo/bar/g' input.txt
```

Replaces all occurrences of foo with bar. Output goes to stdout.

Example: In-place edit with backup

```
$ sed -i.bak 's/localhost/192.168.1.100/g' config.yml
```

Modifies the file directly, saving the original as config.yml.bak.

Example: Delete comment lines

```
$ sed '/^#/d' /etc/ssh/sshd_config
```

Removes all lines starting with # (comments) from the output.

Example: Delete empty lines

```
$ sed '/^$/d' file.txt
```

Removes all blank lines from the output.

Example: Print specific line range

```
$ sed -n '10,20p' logfile.txt
```

Prints only lines 10 through 20 from the file.

Tips & Best Practices

Warning: Always test before -i: Run sed without -i first to preview changes in stdout. Only add -i when confident the result is correct, or use -i.bak for safety.

Pro Tip: Use different delimiters: When patterns contain slashes (like file paths), use a different delimiter: sed 's|/var/www|/srv/http|g' avoids escaping slashes.

Note: sed vs perl -pe: For complex regex with lookaheads/lookbehinds not supported in sed, use perl -pe 's/pattern/replacement/g' instead.

$ sort

Beginner

Sort lines of text files

The sort command arranges lines of text in a specified order. It supports alphabetical, numerical, reverse, and custom sorting. sort is one of the fundamental Unix text processing tools, often combined with uniq, head, and other commands in pipelines.

sort can handle multiple sort keys, allowing...

Options & Flags

`-n`	Sort numerically instead of alphabetically
`-r`	Reverse the sort order
`-k`	Sort by specific field (column)
`-t`	Set field delimiter
`-u`	Remove duplicate lines (unique)
`-h`	Sort human-readable numbers (1K, 2M, 3G)
`-f`	Case-insensitive sort
`-V`	Version number sort
`-o`	Write result to file (can be the same file)

Practical Examples

Example: Sort alphabetically

```
$ sort names.txt
```

Sorts lines in alphabetical order (default behavior).

Example: Sort numerically (largest first)

```
$ sort -rn scores.txt
```

Sorts numbers in descending order. -n treats values as numbers, -r reverses.

Example: Sort by second column

```
$ sort -t',' -k2 -n employees.csv
```

Sorts CSV by the second column numerically using comma as delimiter.

Example: Find largest files

```
$ du -sh /var/log/* | sort -rh
2.1G\t/var/log/journal\n500M\t/var/log/syslog
```

Sorts disk usage by human-readable size (G > M > K) in descending order.

Example: Sort and remove duplicates

```
$ sort -u emails.txt
```

Sorts and removes duplicate lines in one step.

Tips & Best Practices

Pro Tip: Sort in-place: Use -o flag to write back to the same file: sort -o file.txt file.txt. This is safe and atomic.

Note: LC_ALL affects sorting: Locale settings affect sort order. For consistent byte-level sorting across systems, use: LC_ALL=C sort file.txt

Warning: Numeric vs alphabetic: Without -n, sort treats numbers as strings: 9 > 80 > 700. Always use -n for numerical data.

$ tac

Beginner

Concatenate and print files in reverse (last line first)

tac (cat reversed) displays files with lines in reverse order - the last line becomes the first and the first becomes the last. It is the line-level complement to rev (which reverses characters within a line).

tac is useful for viewing log files (most recent entries first), reversing sorted outp...

Options & Flags

`-b`	Attach separator before instead of after
`-r`	Use regex as separator
`-s`	Use STRING as separator instead of newline

Practical Examples

Example: Reverse file lines

```
$ tac file.txt
```

Displays the file with lines in reverse order.

Example: View recent log entries first

```
$ tac /var/log/syslog | head -20
```

Shows the 20 most recent log lines first.

Example: Reverse sort output

```
$ sort file.txt | tac
```

Reverses sorted output (equivalent to sort -r but works for any input).

Example: Reverse numbered list

```
$ seq 1 10 | tac
10
9
8
7
```

Counts from 10 down to 1.

Example: Process file bottom-up

```
$ tac script.sh | head -5
```

Shows the last 5 lines of a file (like tail but in reversed order).

Tips & Best Practices

Pro Tip: tac vs tail: tail -n 20 shows last 20 lines in order. tac file | head -20 shows last 20 lines in reverse. Use tac for reverse chronological views.

Note: tac vs rev: tac reverses line ORDER (last line first). rev reverses characters WITHIN each line. They are different operations.

Warning: Memory usage: tac reads the entire file into memory. For very large files (multi-GB), use tail or sed for viewing the end.