# Text Viewing & Pipelines (Core)

Linux Commands Course · Section 5

#### Goal

Learn how to view, count, redirect, and connect text streams effectively.

Linux treats data as text flowing through pipelines — mastering this is key to powerful command-line work.

# **Viewing Text — less**

less is the most convenient pager for reading long text output or files.

less /etc/passwd

#### Inside less:

- Space / Page Down → next page
- b / Page Up → previous page
- /pattern → search
- n / N → next/previous match
- q → quit

You can pipe output into it too:

ls -l /etc | less

# Counting Text — wc



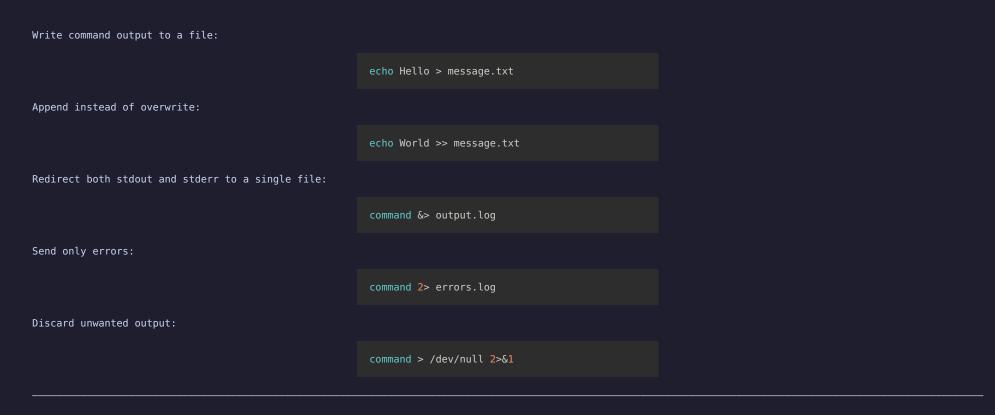
# Redirection Basics

Every command has three data streams:

- stdin (0) input
  stdout (1) normal output
  stderr (2) error output

You can redirect these streams to files.

# **Output Redirection**



# **Input Redirection**

Feed	а	file	as	input	to	а	command	:
------	---	------	----	-------	----	---	---------	---

sort < unsorted.txt</pre>

This reads from unsorted.txt instead of keyboard input.

# Pipelines - |

The pipe operator (|) connects one command's output to another's input.

cat /etc/passwd | grep bash | wc -l

#### This example:

- Reads /etc/passwd
   Filters lines containing "bash"
   Counts them

Pipelines chain tools to form complex processing flows.

### Here-Documents (<<)

A here-document feeds a block of text directly into a command.

cat <<EOF > welcome.txt
Welcome to Linux!
This file was generated from a here-doc.
EOF

Everything until EOF is sent as input to cat and saved into the file.

You can use any marker instead of EOF.

# Here-Strings (<<<)

Feed a single line of text as input:

cat <<< "Hello from here-string"

Equivalent to:

echo "Hello from here-string" | cat

# **Splitting and Merging Streams — tee**

tee writes output to both the terminal and a file simultaneously.

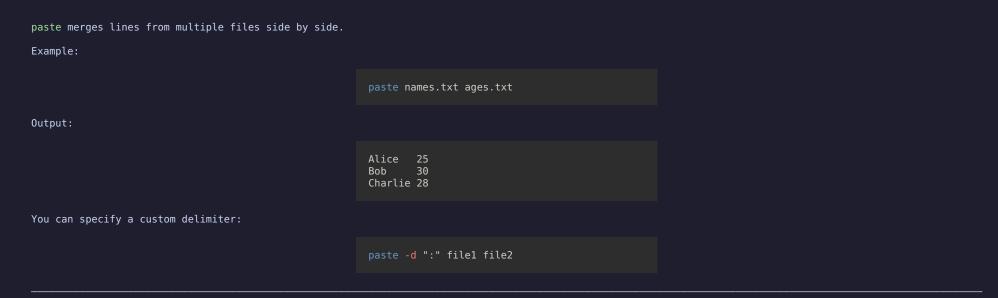
ls -l | tee listing.txt

Append instead of overwrite:

ls -l | tee -a all\_listings.txt

Useful for saving logs while still viewing output live.

# **Combine Multiple Sources — paste**



# **Compare Common Lines — comm**

comm compares two sorted files line by line.

comm fileA fileB

#### Columns:

- 1. Lines unique to fileA
- 2. Lines unique to fileB
  3. Lines common to both

Hide specific columns:

comm -12 fileA fileB # show only common lines

Files must be **sorted** beforehand.

# Join Files by Common Field — join

join merges two files based on a shared column (like a database join).

Example:

join users.txt departments.txt

Use -1 and -2 to choose which fields to join on:

join -1 1 -2 2 file1 file2

Sort both files first for reliable results.

# **Combining Tools in Pipelines**

Real power comes from chaining commands.

Example — count unique shell types:

cat /etc/passwd | cut -d: -f7 | sort | uniq -c

Another example — save and view results:

ps aux | grep ssh | tee ssh\_processes.txt | wc -l

#### Recap

- less scroll through text interactively
- wc count lines, words, or characters
- >, >>, 2>, &> redirect output and errors
- | pipe between commands
- <<, <<< here-docs and here-strings
- tee, paste, join, comm split, merge, and compare streams

Together, these form the foundation of Linux text processing.

#### **Practice**

- Count how many users have /bin/bash in /etc/passwd.
   Redirect all output of ls -lh /etc into a file.
- 3. Append the date to that same file using >>.
- 4. Merge two lists of names using paste.
- 5. Compare two text files for common lines using comm.
- 6. Use a pipeline with tee to save and count results simultaneously.

# Next Up

Text Processing (Core → Plus) — pattern matching and stream editing with grep, sed, and awk.