

Linux & Bash for Data Engineers

Essential command-line skills every data engineer uses daily.

Last reviewed 29 Sep 2026 ·

Prerequisites: None — good place to start

Related: [Git for DE](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/git-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/git-for-de/) · [Docker](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

Overview

Challenge: Nearly every component of a data platform runs on Linux — orchestrator workers, Spark clusters, containers, CI runners, and cloud virtual machines. These environments have no graphical interface, so investigating a failed job means working in a terminal: reading logs, inspecting files, and checking processes.

Solution: The shell provides small, composable tools for inspecting files, filtering text, and automating repetitive tasks. A short pipeline of commands can answer an operational question — such as how many records failed in yesterday's run — in seconds, without writing a program.

The Unix idea: small tools, each doing one thing, connected by pipes

```
cat app.log | grep ERROR | cut -d' ' -f4 | sort |  
uniq -c | sort -rn | head  
read file      keep errors    pick 4th field  group  
count          rank          top 10
```

Typical uses: previewing large files without loading them, following the log of a running job, transferring files to and from object storage, writing wrapper scripts with retries, and scheduling simple jobs with cron.



On this page

Basics - [Navigating the Filesystem](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#navigating-the-filesystem) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#navigating-the-filesystem) - [Working with Files](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#working-with-files) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#working-with-files) - [Viewing & Searching File Contents](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#viewing--searching-file-contents) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#viewing--searching-file-contents) - [Permissions](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#permissions) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#permissions)

Intermediate - [Pipes & Redirection](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#pipes--redirection) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#pipes--redirection) - [Text Processing — grep, awk, sed, cut](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#text-processing) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#text-processing) - [Environment Variables](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#environment-variables) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#environment-variables) - [Processes & Jobs](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#processes--jobs) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#processes--jobs) - [SSH & Remote Servers](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#ssh--remote-servers) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#ssh--remote-servers)

Advanced - [Bash Scripting](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#bash-scripting) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#bash-scripting) - [Cron Jobs](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#cron-jobs) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#cron-jobs) - [Data Engineering Workflows](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#data-engineering-workflows) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#data-engineering-workflows) - [Useful One-Liners](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#useful-one-liners) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#useful-one-liners)

Reference - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#common-pitfalls) - [Cheat Sheet](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#cheat-sheet) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#cheat-sheet) - [Interview Questions](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#interview-questions) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#interview-questions) - [Further Reading](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#further-reading) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/linux-bash/#further-reading)

Navigating the Filesystem

```
pwd          # print working directory
ls           # list files
ls -la       # long format, including hidden files (. prefix)
```



```

ls -lh          # human-readable sizes (KB, MB, GB)
ls -lt          # sort by modification time (newest
first)
ls *.csv        # glob – list only CSV files

cd /path/to/dir # change directory
cd ~            # go to home directory
cd -            # go to previous directory
cd ..           # go up one level
cd ../..        # go up two levels

# Absolute vs relative paths
/home/alice/data/ # absolute – starts from root
./data/           # relative – relative to current
directory
../data/          # relative – one level up, then
into data/

# Find files
find /data -name "*.parquet" # find by
filename
find /data -name "orders_*.csv" -mtime -1 # modified in
last 1 day
find /data -size +100M        # files
larger than 100 MB
find /tmp -type d -empty      # empty
directories
find . -name "*.log" -exec rm {} \; # find and
delete

```

Working with Files

```

# Create
touch orders.csv          # create empty file (or
update timestamp)
mkdir data                # create directory
mkdir -p data/raw/2024/03 # create nested directories
(-p = no error if exists)

# Copy, move, delete

```

```

cp file.csv backup.csv          # copy file
cp -r src/ dest/                # copy directory recursively
mv file.csv archive/           # move / rename
rm file.csv                     # delete file
rm -rf directory/               # delete directory (CAREFUL – no undo)
rmdir empty_dir/                # delete empty directory

# Links
ln -s /data/warehouse link_name # create a symbolic link

# Archive and compress
tar -czf archive.tar.gz data/   # compress directory to .tar.gz
tar -xzf archive.tar.gz         # extract .tar.gz
tar -czf - data/ | gzip > data.tar.gz # compress to stdout

gzip file.csv                   # compress → file.csv.gz (replaces original)
gunzip file.csv.gz              # decompress
zcat file.csv.gz                # view compressed file without extracting

# Check disk usage
df -h                           # disk space on all filesystems
du -sh /data/                   # size of a directory
du -sh /data/*/                 # size of each subdirectory
du -sh * | sort -h              # sorted by size

```

Viewing & Searching File Contents

```

# View files
cat file.csv                    # print entire file
head -20 file.csv               # first 20 lines (default 10)
tail -20 file.csv               # last 20 lines
tail -f app.log                 # follow a growing file (live log streaming)
less file.csv                   # paginated viewer (q to quit, / to search)

```

```

# Word count
wc -l file.csv          # count lines
wc -w file.csv          # count words
wc -c file.csv          # count bytes

# Count CSV rows (excluding header)
wc -l orders.csv | awk '{print $1 - 1}'

# Search
grep "ERROR" app.log    # lines containing
                        "ERROR"
grep -i "error" app.log # case-insensitive
grep -n "ERROR" app.log # show line numbers
grep -r "password" /etc/ # recursive search in
                        directory
grep -v "DEBUG" app.log # invert – lines NOT
                        matching
grep -c "ERROR" app.log # count matching
                        lines
grep -A 3 "ERROR" app.log # 3 lines after match
grep -B 3 "ERROR" app.log # 3 lines before match
grep -E "ERROR|WARN" app.log # extended regex –
                        match either

# Combine view + search
cat app.log | grep "ERROR" | tail -50 # last 50 errors

```

Permissions

```

ls -la
# -rw-r--r-- 1 alice data-eng 1024 Mar 15 10:30 orders.csv
# ----- - file type + permissions
# -       - type: - file, d directory, l symlink
#  rw-    - owner: read + write
#    r--   - group: read only
#    r--   - others: read only

# Permission bits: r=4, w=2, x=1
chmod 644 orders.csv # owner: rw, group: r, others: r

```

```

chmod 755 script.sh      # owner: rwx, group: rx, others:
rx
chmod +x script.sh      # add execute permission for all
chmod -R 755 data/      # recursive

# Change owner / group
chown alice orders.csv
chown alice:data-eng orders.csv
chown -R alice:data-eng /data/

# Current user info
whoami                  # current username
id                      # user id, group id, groups
groups                  # list groups the user belongs to

```

Pipes & Redirection

```

# Pipe – pass stdout of one command to stdin of next
cat orders.csv | grep "shipped" | wc -l

# Redirection
command > output.txt      # stdout to file (overwrites)
command >> output.txt     # stdout to file (appends)
command 2> errors.txt     # stderr to file
command 2>&1 | tee output.txt # both stdout and stderr,
also print to screen
command > /dev/null       # discard stdout
command > /dev/null 2>&1   # discard all output

# tee – write to file AND stdout simultaneously
python pipeline.py | tee pipeline.log

# xargs – build commands from stdin
cat file_list.txt | xargs rm      # delete each file
listed
find . -name "*.tmp" | xargs rm -f

# Subshell substitution
echo "Today is $(date +%Y-%m-%d)"

```

```
files=$(ls /data/*.parquet | wc -l)
echo "Found $files parquet files"
```

Text Processing

grep — search

```
grep "pattern" file
grep -E "^2024-03" dates.txt      # regex: lines starting
with 2024-03
grep -oE "[0-9]+\.[0-9]+" file    # extract all decimal
numbers
grep -l "ERROR" *.log             # list files containing
"ERROR"
```

cut — extract columns from delimited files

```
cut -d',' -f1 orders.csv          # first column (comma-
delimited)
cut -d',' -f1,3 orders.csv        # columns 1 and 3
cut -d',' -f2- orders.csv         # column 2 to end
head -1 orders.csv | cut -d',' -f1-5 # first 5 header
columns
```

sort — sort lines

```
sort file.txt                    # alphabetical
sort -n file.txt                 # numeric sort
sort -rn file.txt                # reverse numeric
sort -t',' -k2 orders.csv        # sort by second CSV
column
sort -t',' -k3 -rn orders.csv    # sort by 3rd column,
numeric descending
sort -u file.txt                 # unique sort
(deduplicate)
```

uniq — deduplicate / count

```
sort file.txt | uniq          # remove duplicate
consecutive lines
sort file.txt | uniq -c      # count occurrences of
each line
sort file.txt | uniq -d      # show only duplicate
lines
cut -d',' -f2 orders.csv | sort | uniq -c | sort -rn #
frequency count
```

awk — process structured text / column math

```
# Print column 3 of a space-delimited file
awk '{print $3}' file.txt

# CSV: print columns 1 and 5
awk -F',' '{print $1, $5}' orders.csv

# Filter: print lines where column 3 > 100
awk -F',' '$3 > 100' orders.csv

# Sum column 3
awk -F',' '{sum += $3} END {print "Total:", sum}'
orders.csv

# Count lines matching a pattern
awk '/ERROR/ {count++} END {print count}' app.log

# Print header + matching rows
awk 'NR==1 || $5 == "shipped"' orders.csv

# Compute average
awk -F',' 'NR>1 {sum+=$3; count++} END {print "Avg:",
sum/count}' orders.csv
```

sed — stream editor

```
# Substitute (replace)
sed 's/old/new/' file.txt          # replace first
```

```

occurrence per line
sed 's/old/new/g' file.txt          # replace all
occurrences
sed 's/old/new/g' file.txt > new_file.txt # write to new
file
sed -i 's/old/new/g' file.txt       # in-place edit

# Delete lines
sed '/pattern/d' file.txt           # delete lines matching
pattern
sed '1d' file.txt                   # delete line 1 (remove
header)
sed -n '5,10p' file.txt             # print only lines 5-10

# Practical: fix CSV delimiter
sed 's/|/,/g' pipe_delimited.txt > comma_delimited.csv

```

Environment Variables

```

# View
env                                # all environment variables
echo $PATH                         # specific variable
printenv HOME                      # same

# Set (current session only)
export DB_HOST=localhost
export DB_PORT=5432
export DB_PASS="my_password"

# Unset
unset DB_PASS

# Persist (add to ~/.bashrc or ~/.zshrc)
echo 'export DB_HOST=localhost' >> ~/.bashrc
source ~/.bashrc                  # reload without restarting
terminal

# Load from .env file
export $(grep -v '^#' .env | xargs) # export all vars
from .env file

```

```

# Use in scripts
DB_HOST=${DB_HOST:-"localhost"}          # default value if
not set
DB_PORT=${DB_PORT:? "DB_PORT must be set"} # error if not
set

# Common env vars every DE uses
echo $HOME          # /home/alice
echo $USER           # alice
echo $PATH           # directories searched for executables
echo $PYTHONPATH     # Python module search path
echo $VIRTUAL_ENV    # active venv path

```

Processes & Jobs

```

# Running processes
ps aux                # all processes
ps aux | grep python  # find Python processes
top                   # live process monitor (q to
quit)
htop                  # better top (if installed)

# Kill a process
kill 12345            # send SIGTERM (graceful)
kill -9 12345         # send SIGKILL (force)
pkill -f "python pipeline.py" # kill by process
name/pattern

# Background jobs
python pipeline.py &  # run in background
jobs                  # list background jobs
fg 1                  # bring job 1 to foreground
bg 1                  # send job 1 to background
nohup python pipeline.py & # run in background, keep
running after logout

# Run in background and log output
nohup python pipeline.py > pipeline.log 2>&1 &
echo "PID: $!"        # print PID of last

```

```
background job
```

```
# Check if a process is running
pgrep -f "pipeline.py"          # returns PID if running
if pgrep -f "pipeline.py" > /dev/null; then echo "Running";
fi
```

SSH & Remote Servers

```
# Connect
ssh alice@192.168.1.100
ssh -i ~/.ssh/my_key.pem ec2-user@ec2-
xx.compute.amazonaws.com
ssh -p 2222 alice@host.example.com    # non-default port

# SSH config file (~/.ssh/config) – avoid typing long
commands
# Host dev-server
#     HostName 192.168.1.100
#     User alice
#     IdentityFile ~/.ssh/dev_key.pem
#     Port 22

ssh dev-server    # uses config above

# Copy files
scp orders.csv alice@server:/data/
scp alice@server:/data/output.parquet ./local/

# rsync – smarter copy (only transfers changed bytes)
rsync -avz data/ alice@server:/data/
rsync -avz --delete data/ alice@server:/data/ # mirror
(delete remote extras)

# Run a command remotely
ssh alice@server "python /opt/pipelines/run.py --date 2024-
03-15"

# SSH tunnel – forward a remote port to localhost
ssh -L 5432:db-server:5432 alice@jump-host
```

```
# Now connect to localhost:5432 to reach db-server:5432

# Generate SSH key
ssh-keygen -t ed25519 -C "alice@example.com"
ssh-copy-id alice@server      # copy public key to server
```

Bash Scripting

```
#!/usr/bin/env bash
# Always use this shebang – finds bash wherever it lives

# Exit on error, undefined vars, pipe failures – essential
# for DE scripts
set -euo pipefail

# — Variables —————
DATE=$(date +%Y-%m-%d)
YESTERDAY=$(date -d "yesterday" +%Y-%m-%d)
OUTPUT_DIR="/data/output/${DATE}"

# — Arguments —————
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <date> <env>"
    exit 1
fi
DATE=$1
ENV=$2

# — Conditionals —————
if [ "$ENV" == "prod" ]; then
    DB_HOST="prod-db.example.com"
else
    DB_HOST="dev-db.example.com"
fi

[ -d "$OUTPUT_DIR" ] || mkdir -p "$OUTPUT_DIR"    # create
if not exists
[ -f "config.yaml" ] || { echo "config.yaml missing"; exit
1; }
```

```

# — Loops —————
for file in /data/raw/*.parquet; do
    echo "Processing $file"
    python transform.py --input "$file" --output
"$OUTPUT_DIR"
done

# Loop over dates
start="2024-01-01"
end="2024-03-31"
current="$start"
while [[ "$current" < "$end" ]]; do
    echo "Processing $current"
    python pipeline.py --date "$current"
    current=$(date -d "$current + 1 day" +%Y-%m-%d)
done

# — Functions —————
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $*"
}

run_with_retry() {
    local cmd=$1
    local max_attempts=${2:-3}
    for ((attempt=1; attempt<=max_attempts; attempt++)); do
        log "Attempt $attempt/$max_attempts: $cmd"
        if eval "$cmd"; then
            return 0
        fi
        [ "$attempt" -lt "$max_attempts" ] && sleep
$((attempt * 5))
    done
    log "ERROR: All $max_attempts attempts failed for:
$cmd"
    return 1
}

# — Error handling —————
cleanup() {
    log "Cleaning up temp files..."
    rm -rf /tmp/pipeline_work_*
}

```

```

}
trap cleanup EXIT      # runs cleanup on any exit (normal or
                        # error)

# — Typical pipeline script —————
log "Starting pipeline for $DATE"

run_with_retry "python extract.py --date $DATE"
log "Extract complete"

run_with_retry "python transform.py --date $DATE"
log "Transform complete"

python load.py --date "$DATE" --env "$ENV" \
    >> "$OUTPUT_DIR/pipeline.log" 2>&1
log "Load complete"

log "Pipeline finished successfully for $DATE"

```

Cron Jobs

Cron is the Linux scheduler for recurring commands.

```

# Edit crontab
crontab -e      # edit
crontab -l      # list
crontab -r      # remove all

# Cron syntax:
# |_____ minute (0-59)
# | |_____ hour (0-23)
# | | |_____ day of month (1-31)
# | | | |_____ month (1-12)
# | | | | |_____ day of week (0=Sun, 6=Sat)
# | | | | |
# * * * * * command

0 2 * * *      /opt/pipelines/run_daily.sh      # 2am
daily
0 * * * *      /opt/pipelines/run_hourly.sh      # every

```

```

hour
*/15 * * * * /opt/pipelines/check_lag.sh          # every
15 min
0 2 * * 1    /opt/pipelines/weekly_report.sh      # 2am
Monday
0 2 1 * *    /opt/pipelines/monthly_load.sh       # 2am on
1st of month
@reboot      /opt/pipelines/startup.sh            # on
system restart

# Best practices for cron
0 2 * * * cd /opt/pipelines && ./run_daily.sh >>
/var/log/pipeline.log 2>&1
# Always: absolute paths, redirect output, use >> to append

# Check cron logs
grep CRON /var/log/syslog      # Ubuntu
tail -f /var/log/cron          # CentOS/RHEL

```

Data Engineering Workflows

Inspect a data file quickly

```

# CSV inspection
head -5 orders.csv          # preview
wc -l orders.csv            # row count
head -1 orders.csv | tr ',' '\n'    # list column
names (one per line)
awk -F',' 'NR==1{print NF}' orders.csv  # column count

# Parquet inspection (requires pyarrow CLI or python one-
liner)
python -c "import pandas as pd;
print(pd.read_parquet('file.parquet').head())"
python -c "import pyarrow.parquet as pq;
pq.read_schema('file.parquet').to_string()" | cat

# JSON inspection
cat data.json | python -m json.tool | head -50 # pretty-
print JSON

```

```
cat data.ndjson | head -3 | python -m json.tool # pretty-  
print first 3 NDJSON lines
```

Check S3 quickly

```
# AWS CLI
aws s3 ls s3://my-bucket/data/orders/          # list
files
aws s3 ls s3://my-bucket/data/ --recursive | wc -l # count
files
aws s3 ls s3://my-bucket/data/ --recursive --human-readable
# show sizes
aws s3 cp s3://bucket/file.parquet ./          #
download
aws s3 cp ./local.csv s3://bucket/path/        # upload
aws s3 sync ./local/ s3://bucket/remote/      # sync
directory
aws s3 rm s3://bucket/path/file.csv            # delete
```

Monitor a running pipeline

```
# Follow logs
tail -f /var/log/pipeline.log
tail -f /var/log/pipeline.log | grep -E "ERROR|WARN"

# Watch a process
watch -n 5 "ps aux | grep python"             # refresh every
5s
watch -n 10 "ls -lth /data/output/ | head" # watch output
files appear

# Check memory and CPU
free -h                                         # memory usage
vmstat 1 5                                     # system stats every 1s, 5 times
iostat -x 1 5                                  # I/O stats

# Check disk space before a large job
df -h /data
du -sh /data/warehouse/
```

Useful One-Liners

```
# Count rows in all CSVs in a directory
for f in /data/*.csv; do echo "$f: $(wc -l < $f) rows";
done

# Find the largest files in a directory
du -ah /data/ | sort -rh | head -20

# Remove duplicate lines from a file (unsorted)
awk '!seen[$0]++' file.txt > deduped.txt

# Extract unique values from a CSV column
cut -d',' -f3 orders.csv | sort -u

# Count distinct values in a column
cut -d',' -f3 orders.csv | sort | uniq -c | sort -rn

# Check if yesterday's file exists
yesterday=$(date -d "yesterday" +%Y-%m-%d)
[ -f "/data/orders_${yesterday}.parquet" ] && echo "Found"
|| echo "Missing"

# Wait for a file to appear (poll every 30s, timeout 1
hour)
timeout 3600 bash -c 'until [ -f /data/ready.flag ]; do
sleep 30; done'

# Split a large CSV into 1M-row chunks
split -l 1000000 large_file.csv chunk_

# Rename all .txt files to .csv
for f in *.txt; do mv "$f" "${f%.txt}.csv"; done

# Find and kill a specific process
ps aux | grep pipeline.py | grep -v grep | awk '{print $2}'
| xargs kill

# Check last N lines of a gzipped log
zcat app.log.gz | tail -100

# Count lines across all gzipped files
```

```
zcat /logs/*.gz | wc -l
```

```
# Run a command and time it
```

```
time python pipeline.py --date 2024-03-15
```

```
# Run multiple commands in parallel
```

```
parallel python process.py --date {} ::: 2024-01-01 2024-01-02 2024-01-03
```

Common Pitfalls

Pitfall	Symptom	Fix
Scripts without <code>set -euo pipefail</code>	A step fails, the script keeps going and loads partial data	Put <code>set -euo pipefail</code> at the top of every script
Unquoted variables (<code>rm -rf \$DIR/*</code>)	Filenames with spaces break; an empty <code>\$DIR</code> becomes <code>rm -rf /*</code>	Always quote: <code>"\$DIR"</code> ; use <code>\${DIR:?}</code> to fail if unset
Relative paths in cron jobs	Works in your terminal, fails silently under cron	Cron has a minimal environment: use absolute paths, set <code>PATH</code> , <code>cd</code> explicitly
Cron output not redirected	Failures vanish (or fill up local mail)	<code>>> /var/log/job.log 2>&1</code> on every cron line
<code>cat file grep / wc -l</code> on huge files for row counts	Slow, and <code>wc -l</code> counts newlines inside quoted CSV fields	Fine for quick checks; for real counts use DuckDB/pandas which parse CSV properly
Parsing CSV with <code>cut -d','</code>	Wrong columns when a field contains a quoted comma	Use <code>csvkit</code> , <code>mlr</code> (Miller), or DuckDB for real CSVs
<code>export \$(cat .env xargs)</code> with spaces/quotes in values	Variables get split or mangled	<code>set -a; source .env; set +a</code>
<code>kill -9</code> as the first resort	Process can't clean up: temp files, locks, half-written output	<code>kill</code> (SIGTERM) first; <code>-9</code> only if it doesn't exit

Pitfall	Symptom	Fix
<code>date -d</code> in scripts run on macOS	illegal option -- d	GNU <code>date</code> is Linux-only; use <code>gdate</code> on macOS or do date math in Python

Cheat Sheet

Task	Command
Preview a file	<code>head -5 f.csv · tail -f app.log · less f.csv</code>
Row count	<code>wc -l < f.csv</code>
Column names, one per line	<code>head -1 f.csv \ tr ',' '\n'</code>
Search logs	<code>grep -n -E "ERROR\ WARN" app.log</code>
Search recursively	<code>grep -rn "pattern" dir/</code>
Frequency count of a column	<code>cut -d, -f3 f.csv \ sort \ uniq -c \ sort -rn</code>
Sum a column	<code>awk -F, 'NR>1 {s+=\$3} END {print s}' f.csv</code>
Replace text in place	<code>sed -i 's/old/new/g' file</code>
Find big files	<code>du -ah dir \ sort -rh \ head -20</code>
Disk free	<code>df -h</code>
Find files changed today	<code>find /data -name "*.csv" -mtime -1</code>
Run in background, survive logout	<code>nohup cmd > out.log 2>&1 &</code>
Is it running?	<code>pgrep -fa pipeline.py</code>
Copy to/from server	<code>rsync -avz src/ user@host:/dst/</code>
Tunnel to a private DB	<code>ssh -L 5432:db:5432 user@bastion</code>
Load a <code>.env</code>	<code>set -a; source .env; set +a</code>

Task	Command
Safe script header	<code>#!/usr/bin/env bash + set -euo pipefail</code>

Redirection: `>` overwrite · `>>` append · `2>&1` merge stderr into stdout · `| tee f` write to file *and* screen

Cron: min hour day-of-month month day-of-week — `0 2 * * *` = 2am daily, `*/15 * * *`
`*` = every 15 min

Interview Questions

Q: What does `set -euo pipefail` do and why should every pipeline script have it? A: `-e` exits the script as soon as any command fails; `-u` treats unset variables as an error instead of silently expanding to an empty string; `-o pipefail` makes a pipeline fail if *any* command in it fails, not just the last one. Without these, a failed extract step can be ignored and the load step runs on empty or partial data — the worst kind of failure because nothing alerts.

Q: What's the difference between `>` and `>>`, and what does `2>&1` mean? A: `>` redirects stdout to a file, overwriting it; `>>` appends. File descriptor 1 is stdout and 2 is stderr, so `2>&1` sends stderr to wherever stdout is currently going. `cmd > log 2>&1` captures both streams in the log. Order matters: `cmd 2>&1 > log` sends stderr to the terminal, because stdout wasn't redirected yet when stderr was duplicated.

Q: A cron job works when you run it manually but fails under cron. Why? A: Cron runs with a minimal environment: a different `PATH`, no `.bashrc`, no activated virtualenv, and a different working directory (usually `$HOME`). Fixes: use absolute paths for commands and files, `cd` into the project directory, activate the venv or call its Python directly, load environment variables explicitly, and redirect output to a log so you can see the actual error.

Q: How would you find which process is using all the memory on a server? A: `top` or `htop` sorted by memory (M in top), or `ps aux --sort=-%mem | head`. `free -h` shows overall usage, including how much is just cache. If the process was killed, `dmesg | grep -i oom` shows whether the kernel's OOM killer did it — common with Spark executors or pandas jobs that load too much data.

Q: How would you quickly check whether yesterday's file landed in S3 and has data? A: `aws s3 ls s3://bucket/orders/dt=$(date -d yesterday +%F)/ --human-readable` lists the files and their sizes. Zero files or zero-byte files means the upstream failed. For a row count without downloading, query it with DuckDB or Athena. In a pipeline, this check belongs in an Airflow sensor, not a manual command.

Q: What's the difference between SIGTERM and SIGKILL? A: `kill <pid>` sends SIGTERM (15), which asks the process to stop — it can catch the signal, finish writing, release locks, and clean up. `kill -9` sends SIGKILL, which the kernel enforces immediately; the process gets no chance to clean up, which can leave partial files or stale lock files. Always try SIGTERM first.

Further Reading

- [GNU Bash manual](https://www.gnu.org/software/bash/manual/) (<https://www.gnu.org/software/bash/manual/>)
 - [ShellCheck](https://www.shellcheck.net/) (<https://www.shellcheck.net/>) — linter that catches most bash bugs; run it on every script
 - explainshell.com (<https://explainshell.com/>) — paste a command, see what each part does
 - *The Linux Command Line* — William Shotts (free online at [linuxcommand.org](https://linuxcommand.org/tlcl.php) (<https://linuxcommand.org/tlcl.php>))
 - [Miller \(`mlr` \)](https://miller.readthedocs.io/) (<https://miller.readthedocs.io/>) — `awk / sort / cut` that understands CSV and JSON properly
-

Previous: [Data Modeling](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) (<https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/>) • **Next:** [Git for DE](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/git-for-de/) (<https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/git-for-de/>) • **Back to:** [Index](https://sarangambekar1997.github.io/data-engineering-handbook/) (<https://sarangambekar1997.github.io/data-engineering-handbook/>)

 September 29, 2026