Essential Linux Command-Line Tools and Redirection – Hal Pomeranz Guide

1. Core Commands

awk

- Purpose: Advanced text/file processing (pattern scanning, field extraction, reformatting).
- Example:

```
awk '{print $1}' file.txt
# Prints the first field of each line in file.txt
```

cut

- Purpose: Extracts sections/fields from lines.
- Example:

```
cut -d: -f1 /etc/passwd
# Extracts the first field (username) using ':' as a delimiter
```

grep

- Purpose: Pattern-based searching in files or streams.
- Example:

```
grep 'pattern' file.txt
# Finds lines matching 'pattern'
```

sort

- Purpose: Sorts lines of text (alphabetical, numerical, etc.).
- Example:

```
sort file.txt
# Sorts lines in file.txt
```

uniq

- Purpose: Filters out repeated lines (usually after sorting).

- Example:

  ```
  sort file.txt | uniq
  # Shows unique lines
  ```

head

- Purpose: Outputs the first N lines (default: 10).
- Example:

  ```
  head -20 file.txt
  # First 20 lines
  ```

tail

- Purpose: Outputs the last N lines (default: 10).
- Example:

  ```
  tail -f /var/log/syslog
  # Live-follow log updates
  ```

wc

- Purpose: Counts lines, words, and characters.
- Example:

  ```
  wc -l file.txt
  # Line count
  ```

ls

- Purpose: Lists directory contents.
- Example:

  ```
  ls -l /path/to/dir
  # Long listing
  ```

ps

- Purpose: Displays current process status.
- Example:

  ```
  ps aux | grep ssh
  # Shows SSH processes
  ```

md5sum

- Purpose: Computes MD5 hash of files.
- Example:

  ```
  md5sum file.txt
  # MD5 checksum
  ```

---

2. Output Redirection and File Descriptors

- **Standard Output (**`**` and `**`**):**
  - > sends the output of a command to a file (overwrites the file).

    ```
    echo "System scan complete" > results.txt
    ```

  - >> appends the output to the end of a file (does not overwrite).

    ```
    echo "Log entry" >> logfile.txt
    ```

- **Standard Error (`` ``):**
  - Redirects error messages to a file, leaving normal output on the screen.

    ```
    grep root /var/log/* 2> errors.txt
    ```

- **Suppressing Output to `` ``:**
  - Sends output you don't care about into the void.

    ```
    grep root /var/log/* 2>/dev/null
    ```

- **Combining Output and Error Streams (**`**, **`**):**
  - Sends both standard output and standard error to the same file.

    ```
    ls /notreal > out.txt 2>&1
    # Or (Bash only):
    ls /notreal &> out.txt
    ```

  - **Order matters:** `> file 2>&1` works, `2>&1 > file` does not do what you expect.

---

3. The `tee` Command

- Use `tee` to send output to both a file and the screen.

  ```
  dmesg | tee boot.log
  echo "done" | tee -a status.log
  ```

---

4. Command Substitution (`$()`, Backticks)

- Preferred syntax: `$(command)` — modern and nestable.

```
echo "Hostname is: $(hostname)"
# Files in current directory:
echo "Files: $(ls $(pwd))"
```

- Legacy: `` `command` `` (backticks) — older, not as easy to nest.

---

5. Inline Math with `$(( ))`

- Do arithmetic directly in the shell.

```
echo $(( 2 * 4096 ))
# For logic, offsets, quick math, etc.
```

---

6. Shell History Shortcuts

- `!!` — repeat the previous command.

```
apt update
sudo !!
```

- `!$` — expands to the last argument of the previous command.

```
cat report.txt
vim !
```

---

7. Random Numbers and IP Generation

- `$RANDOM` produces a pseudorandom integer.

```
echo $RANDOM
```

- Generate random IP address:

```
echo "$((RANDOM%256)).$((RANDOM%256)).$((RANDOM%256)).$((RANDOM%256))"
```

---

8. Practical Log Processing

- Use pipes and commands for log analysis:

```
awk '{print $1}' access.log | sort | uniq -c | sort -nr
# Prints unique IPs from a log, sorted by frequency
```

---

9. Loops in Linux Shell Scripting

Loops let you automate repetitive tasks or process many items efficiently.

- **Syntax:**

  ```
  for file in *.gz; do
      echo "Processing $file"
      gunzip "$file"
  done
  # Or one-liner:
  for f in *.log; do grep ERROR "$f"; done
  ```

- **Variable Usage:** Use `$file` (or your chosen variable name) to reference the current item.
- **Multiple Commands:** Separate with newlines or semicolons inside the loop.
- **Output Formatting:**

  ```
  echo -n "No newline"
  echo -e "Tab\tSeparated\nNewline"
  ```

- **Pipeline Integration:** Pipe loop output to other commands:

  ```
  for f in *.log; do cat "$f"; done | grep 192.168
  ```

- **Flexibility:** Use loops for repetitive or bulk actions—renaming files, parsing logs, running any command over a list.

Mastering loops lets you automate, scale, and save tons of manual effort.

---

10. Extra Tips & Tricks

- Use `history | grep keyword` to search your command history quickly.
- Use `!!:n` to reuse the nth argument from the previous command (e.g., `!!:2`).
- Use `CTRL+R` for reverse search in shell history.
- Try xargs for building commands from output, e.g., `cat list.txt | xargs rm`.
- Use `basename` and `dirname` to extract filename or directory from a path.
