Bash Shell Cheatsheet

Basic Structure and Scripting

Bash (Bourne Again SHell) is the default command processor for most Linux distributions.

- Shebang: `#!/bin/bash` (First line of a script)
- Execution:
  - `chmod +x script.sh`: Add execute permission.
  - `./script.sh`: Run the script.
- Variables:
  - Declaration: `foo="value"` (No spaces around `=`. `foo = bar` is interpreted as calling `foo` with arguments).
  - Access: `${foo}` or `$foo`.
- Quotes:
  - `' '` (Single quotes): Literals, no expansion. (`'$foo'` -> `$foo`)
  - `" "` (Double quotes): Variable expansion. (`"$foo"` -> `value`)
- Substitutions:
  - Command: `$(CMD)` runs command and replaces it with its output.
  - Process: `<(CMD)` runs command and replaces it with a temporary file path containing the output. (`diff <(ls a) <(ls b)`)
- Special Variables:
  - `$0`: Script name.
  - `$1` - `$9`: Script arguments.
  - `$#`: Number of arguments.
  - `$@`: All arguments.
  - `$?`: Exit status of last command (0 = Success).
  - `$$`: PID of current script.
  - `!!`: Entire last command (e.g., `sudo !!`).
  - `$_`: Last argument of the last command.

Navigation and File Operations

- `pwd`: Print current working directory.
- `ls`: List files and directories (`-l` for details, `-a` for hidden).
- `cd [dir]`: Change directory.
- `mkdir [dir]`: Create a directory.
- `cp [src] [dest]`: Copy files or directories (`-r` for recursive).
- `mv [old] [new]`: Move or rename.
- `rm [file]`: Remove files (`-r` for directories, `-f` to force).
- `touch [file]`: Create an empty file.

File Permissions (chmod)

- Structure: `[Owner][Group][Others]`
- Modes: `r` (4, read), `w` (2, write), `x` (1, execute).
- Examples:
  - `chmod 755 script.sh`: Owner can do all; others can read/execute.
  - `chmod 644 file.txt`: Owner can read/write; others can only read.

Redirection and Pipes

- `>`: Redirect output to a file (overwrite).
- `>>`: Append output to a file.
- `2>`: Redirect error output.
- `&>`: Redirect both output and error.
- `<`: Take input from a file.
- `|`: Pipe output of one command as input to another.

Core Utilities for Text Processing

- `grep [pattern] [file]`: Search for a pattern.
- `find [path] -name [pattern]`: Search for files by name.
- `awk '{print $1}' [file]`: Text processing and data extraction.
- `sed 's/old/new/g' [file]`: Stream editor for filtering and transforming text.
- `head -n [N] [file]`: View first N lines.
- `tail -n [N] [file]`: View last N lines.
- `cat`, `less`, `more`: View file content.

Control Flow

- `if` Statements:

```
if [[ $VAR -eq 10 ]]; then
    # Logic
elif [[ $VAR -lt 10 ]]; then
    # Logic
else
    # Logic
fi
```

- Comparison Operators:
  - Numbers: `-eq`, `-ne`, `-lt`, `-le`, `-gt`, `-ge`.
  - Strings: `==`, `!=`, `-z` (empty), `-n` (not empty).
- Loops:
  - `for i in {1..5}; do ... done`
  - `while [[ condition ]]; do ... done`

Process Management

- `ps aux`: List all running processes.
- `top` / `htop`: Real-time system monitoring.
- `kill [PID]`: Terminate a process.
- `&`: Run a command in background.
- `jobs`: List background jobs.
- `fg %1`: Bring job 1 to foreground.

Networking

- `curl -O [URL]`: Download a file.
- `wget [URL]`: Download a file.
- `ping [host]`: Check network connectivity.
- `ssh [user]@[host]`: Secure shell login.
- `scp [src] [dest]`: Secure copy between hosts.
- `netstat -tuln`: List listening ports.

Google Style Guide

Basic Rules

- Use `#!/bin/bash` as the shebang.
- Limit scripts to small, simple tasks. For complex logic, consider Python or Go.
- Use 2-space indentation (no tabs).
- Wrap all variable expansions in double quotes: `"$VAR"`.

Functions

- Declare functions using `func_name() { ... }` (avoid the `function` keyword).
- Always use `local` for variables inside functions.
- Keep functions short and focused on a single task.

Variables and Constants

- Use `UPPER_CASE` for global constants or environment variables.
- Use `lower_case` for local variables.
- Avoid using global variables for data passing between functions.

Command Substitution

- Use `$(command)` instead of backticks `command` .

Conditionals and Loops

- Use `[[ ... ]]` instead of `[ ... ]` or `test` for better safety and features.
- Always check the exit status of important commands or use `set -e`.

Documentation and Comments

- Provide a header comment for every script (purpose, usage, author).
- Document all functions (description, arguments, return values).
- Use comments to explain “why” rather than “what” for complex logic.