Utilities & Apps Index

Terminal Stack Notes

Single landing page. The raw, unsorted originals live in `notes/archive/raw/`.


```
[
  {"path": "10-setup", "focus": "Machines and bootstrap", "files": ["shell-and-dotfiles.md", "package-m
  {"path": "20-productivity", "focus": "Navigation, templates, helper apps", "files": ["shell-shortcuts
  {"path": "30-search-scripting", "focus": "Globs, loops, search", "files": ["globbing-and-chaining.md"
  {"path": "40-workspace", "focus": "tmux, yabai, editors", "files": ["tmux.md", "yabai-and-skhd.md", "
  {"path": "50-networking", "focus": "SSH, scans, process control", "files": ["ssh-and-remote-access.md
  {"path": "60-deploy", "focus": "Git, CDN, CPU, CLI recipes", "files": ["git-and-versioning.md", "cdn-
  {"path": "70-media-automation", "focus": "Video, image, AI tooling", "files": ["ffmpeg-and-video.md",
  {"path": "90-appendix", "focus": "Symbols + app index", "files": ["ascii-and-symbols.md", "utilities-
]
```


Quick search moves:

- `rg --heading "" notes` → headings by file
- `rg --files -g '*network*' notes` → locate files by glob
- `rg -n "ssh" notes/50-networking` → dive straight into a topic

Purpose

Keep one page of the stuff you actually forget: how to choose a manager, cleanly switch, and keep scripts moving. Everything else can be looked up when needed.

Pick One, Delete the Rest


```
{
  "npm": "boring but universal",
  "yarn": "legacy React Native / workspaces",
  "pnpm": "fastest install, strict linking (default here)"
}
```


Before installing, remove any other lockfile so the repo doesn't straddle tools:

```
rm -f package-lock.json yarn.lock pnpm-lock.yaml node_modules -rf
```

Bootstrap Habit

```
pnpm init
pnpm install          # or pnpm install <pkg>
pnpm run dev          # standard scripts
```

Swap `pnpm` with whatever manager you're keeping. The thing that matters: run *every* lifecycle command with the same tool so `node_modules` stays consistent.

When You Need a Scaffold

Use the upstream generators, not memorized syntax:

- Vite: `pnpm create vite`
- Next.js: `pnpm create next-app`

- Expo: `pnpm create expo`

They all prompt interactively now, so the only rule is remembering to stick with your manager afterward.

Busywork You Can Automate

- **Parallel scripts:** `pnpm run dev:site & pnpm run dev:cms`.
- **Blocking pipelines:** `pnpm run build && vercel deploy --prod`.
- **Temporary globals:** `pnpm dlx create-next-app` or `npx expo-doesnt-matter` instead of polluting `npm -g`.
- **Audits:** `pnpm audit --prod` after large bumps.
- **Cache cleanup:** `pnpm store prune` when disk blows up.

Switching Managers Later

```
rm -rf node_modules package-lock.json yarn.lock pnpm-lock.yaml
PNPM_HOME=~/Library/pnpm
export PATH="$PNPM_HOME:$PATH"
pnpm install
```

Same idea for npm/yarn—just set the right PATH and reinstall. If binaries still fail, run `which pnpm` to confirm the PATH order.

Troubleshooting

- `command not found`: add the manager's bin path (pnpm uses `~/Library/pnpm` on macOS).
- Permission denied installing globals: use `pnpm dlx`/`npx` or set `npm config set prefix ~/npm-global`.
- Lockfile conflicts in CI: delete caches, re-install locally, re-run pipeline so everyone shares the exact lockfile.

That's it—if you need to remember specific subcommands later, look them up. This page is just the friction remover.

Purpose

One command to clone + bootstrap and a single place to remember where every config lives. Use this file any time a new Mac arrives or when you need to explain how your environment stays reproducible.

Quick Start

```
git clone git@github.com:Tor-Grimsson/.dotfiles.git ~/.dotfiles
~/.dotfiles/bootstrap.sh
```

Validate GitHub trust right after:

```
ssh -T git@github.com
```

If SSH fails, fix keys before running anything else.

Repository Layout

```
~/.dotfiles
  Brewfile
  bootstrap.sh
  bin/                 # custom scripts, symlinked to ~/bin
  git/.gitconfig
  shell/.zshrc, .zprofile
  ssh/config           # no keys, only config
  vscode/              # settings, keymaps, extensions.txt
```

```
macos/defaults.sh
```

The repository *is* the system description; nothing outside of it is assumed.

Bootstrap Contract

Step	Command	Notes
1	`which brew || /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`	Script guards against missing Homebrew.
2	`brew bundle --file ~/.dotfiles/Brewfile`	Installs every CLI + app defined. Keep entries clean so the bundle succeeds on stock macOS.
3	`~/.dotfiles/bootstrap.sh`	Symlinks configs, restores VS Code, applies macOS defaults.
4	`ssh -T git@github.com`	Confirms SSH trust for this machine only.
5	`brew bundle cleanup`	Optional tidy-up after verifying installs.

The bootstrap script must remain idempotent: run it on a fresh machine and expect identical results every time.

SSH Keys & Identity

- Generate a unique key per device (`ssh-keygen -t ed25519 -C "device label"`), never copy keys between Macs.
- GitHub stores only the public key; revoking a machine is as easy as deleting that entry.
- `~/.ssh/config` lives inside the repo. Keep host aliases, `Include` directives, and preferred ciphers there. Never commit private keys.

Trust Flow Refresher

SSH exists to answer “Can this machine be trusted?” not “What’s the password?” Host keys prove the server, client keys prove you. GitHub leans on that to let automation push/pull without babysitting.

Oh My Zsh Integration

Item	Location	Reminder
Framework	`~/.oh-my-zsh`	Installed via bootstrap
Main script	`~/.oh-my-zsh/oh-my-zsh.sh`	Sourced from `.zshrc`
Themes	`~/.oh-my-zsh/themes/`	Keep custom themes under `custom/themes/`
Plugins	`~/.oh-my-zsh/plugins/`	Enable in `.zshrc`
Custom files	`~/.oh-my-zsh/custom/`	Safe for personal tweaks

Edit config with `code ~/.zshrc` or `nvim ~/.zshrc`, then `source ~/.zshrc`.

Bindkey Cheatsheet

Use `cat -v` to capture weird key sequences, then bind them:

```
bindkey '^[^?' backward-kill-word
bindkey '^[[3;3~' backward-kill-word
bindkey '^[b' backward-word
bindkey '^[f' forward-word
```

`Ctrl+W`, `Ctrl+U`, `Ctrl+K` still work out of the box; bindkeys extend behaviour for meta keys or custom keyboards.

Checklist Before Calling It "Done"

1. `brew doctor` returns clean output.
2. `which zsh` points to Homebrew-managed shell if that's your default.
3. `echo $PATH` lists `/opt/homebrew/bin` (Apple Silicon) or `/usr/local/bin` before `/usr/bin`.
4. `alias` shows your curated shortcuts.
5. `touch ~/Desktop/bootstrap-complete` (or similar) so you know which machines are fully provisioned.

Troubleshooting

- **Broken host key warning** → run `ssh-keygen -R host && ssh user@host` to accept the new fingerprint *only after* verifying on the server.
- **Bootstrap rerun fails** → ensure Brewfile paths are still valid; remove abandonware casks.
- **Keybindings don't work** → confirm your terminal sends meta characters (check `Profiles → Keyboard` inside iTerm/Ghostty) and re-run `cat -v`.

References

- `.dotfiles` repo README for script internals.
- Oh My Zsh documentation for plugin syntax.
- Apple System Settings → General → Sharing for hostname changes when enabling SSH.

---

bootstrap.sh

```
#!/usr/bin/env bash
set -e

DOT="$HOME/.dotfiles"

# Install Homebrew if missing
if ! command -v brew >/dev/null 2>&1; then
  /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
fi

# Install packages
brew bundle --file="$DOT/Brewfile"

# Ensure SSH dir exists
mkdir -p "$HOME/.ssh"

# Symlink shell + git
ln -sf "$DOT/shell/.zshrc"     "$HOME/.zshrc"
ln -sf "$DOT/shell/.zprofile" "$HOME/.zprofile"
ln -sf "$DOT/git/.gitconfig"  "$HOME/.gitconfig"
ln -sf "$DOT/bin" "$HOME/bin"
```

```
# SSH config (no keys)
if [ -f "$DOT/ssh/config" ]; then
  ln -sf "$DOT/ssh/config" "$HOME/.ssh/config"
  chmod 600 "$HOME/.ssh/config" || true
fi

# VS Code
if command -v code >/dev/null 2>&1; then
  mkdir -p "$HOME/Library/Application Support/Code/User"
  ln -sf "$DOT/vscode/settings.json" \
    "$HOME/Library/Application Support/Code/User/settings.json" 2>/dev/null || true
  ln -sf "$DOT/vscode/keybindings.json" \
    "$HOME/Library/Application Support/Code/User/keybindings.json" 2>/dev/null || true
  cat "$DOT/vscode/extensions.txt" | xargs -n 1 code --install-extension || true
fi

# macOS defaults
[ -x "$DOT/macos/defaults.sh" ] && "$DOT/macos/defaults.sh"
```

- `#!/usr/bin/env bash` lets the system locate whichever bash is first on `$PATH`, avoiding hard-coding `/bin/bash`.
- `set -e` makes the script bail on the first non-zero exit status so you don't continue with a half-broken setup.
- `if ! command -v brew >/dev/null 2>&1; then … fi` checks whether `brew` exists; the `!` negates the test so the installer runs only when missing. Sending stdout/stderr to `/dev/null` keeps the check quiet.
- All the `ln -sf source target` calls symlink files into `$HOME`; `-s` creates symlinks and `-f` overwrites existing ones so reruns stay idempotent.
- `[ -x "$DOT/macos/defaults.sh" ] && "$DOT/macos/defaults.sh"` is a short-circuit: run the macOS defaults script only if it exists and is executable.

Purpose

Stop retyping the same nested `mkdir` commands. This captures every structure you laid out for assets, projects, and exhibitions so you can paste once, tweak labels, and move on.

Quick Use

```
mkdir -p 04_assets--library/{category__logos,category__icons,category__textures,category__hdris,categor
```

Use brace expansion to enumerate folders, or stack single commands for clarity when needed.

Asset Libraries

```
mkdir -p category__images/{category__components,category__pages,category__features,category__blog,categ
mkdir -p category__video/{category__loops,category__overlays,category__b-roll,category__textures,catego
mkdir -p category__fonts/{category__brand-fonts,category__webfonts,category__experimental,category__arc
mkdir -p category__styles/{category__css,category__design-tokens,category__themes,category__utility-cla
mkdir -p category__components/{category__jsx-components,category__layouts,category__sections,category__
mkdir -p category__copy/{category__pages,category__features,category__components,category__blog,categor
mkdir -p category__video/{category__hero-loops,category__component-videos,category__page-videos,categor
```

Exhibition / Media Collections

```
mkdir -p {category__music-videos,category__video-art,category__music-releases,category__reels,category_
mkdir -p {category__solo,category__group,category__festival,category__institutional,category__online,ca
mkdir -p {category__music-releases,category__video-art,category__albums,category__eps,category__singles
mkdir -p category__exhibition-name-template/{category__works,category__video,category__audio,category__
```

Typeface Projects

```
mkdir -p category__typeface-<name>/{category__typeface,category__images,category__specimens,category__w
```

Project Libraries

```
mkdir -p project-alpha/{category__briefs,category__assets,category__media,category__deliverables,catego
```

For full library root:

```
library-root/
  01_projects--general/
    category__active-projects/
    category__completed-projects/
    category__incoming/
  02_projects--categories/
    category__music-videos, category__commercial, ...
  03_visuals--library/
    category__typography/, category__renders, ...
  04_assets--library/
  05_raw--inputs/
  06_documents--library/
  07_clients--library/
  08_reference--library/
  09_admin--library/
  99_archive--cold/
```

Concrete Example

```
01_projects--general/
  category__active-projects/
    project-alpha/
      category__briefs/project-alpha__brief__20251128.pdf
      category__assets/project-alpha__styleframes__20251127.png
      category__media/project-alpha__footage-selects__20251128.mp4
      category__deliverables/project-alpha__hero-edit__20251128.mp4
      category__notes/project-alpha__meeting-notes__20251125.txt
    project-beta/
      ...
```

Helpers

- Convert multiline text to comma-separated brace lists:

```
paste -sd, <(sed 's/^[[:space:]]*//' <<< "category__logos
category__icons
category__textures")
```

- `<<<` feeds the multiline block into `sed` (here-string), `sed` strips leading spaces, `<( … )` exposes that cleaned output as a temporary file, and `paste -sd,` joins the lines with commas so you can drop the result straight into a brace expansion.
- Drop brace expansion when you need to annotate each command for clarity.

Tips

- Use `mkdir -p` so rerunning commands is safe—no errors if the folder already exists.
- Pair with `tree -L 2` to verify the structure after creation.
- Keep log files inside each project (e.g., `project-alpha/README.md`) to record what each folder is for.

Purpose

If it's obvious enough to Google, drop it. This page only tracks the moves that keep you fast inside the shell—key chords, inline editing, and the handful of loops you forget if they're not written down.

Keyboard Muscle Memory

Action	Shortcut	Why it matters
Delete previous/next word	`Ctrl+W` / `Alt+D`	Rewrite long paths without reaching for arrow keys
Delete to start/end	`Ctrl+U` / `Ctrl+K`	Clean the line and rebuild
Cancel command	`Ctrl+C`	Kill runaway processes instantly
Literal newline	`Ctrl+V`, `Enter`	Build multi-line commands without leaving the prompt
Autocomplete	`Tab` (double-tap for menu)	No more typing entire filenames

Custom Bindings

- Capture weird sequences with `cat -v`, then bind them in `.zshrc`:

```
bindkey '^[^?' backward-kill-word
bindkey '^[[3;3~' backward-kill-word
```

- Karabiner snippet for a Hyper key (one modifier to rule them all):

```
{
  "description": "Right Command to Hyper",
  "manipulators": [
    {
      "from": { "key_code": "right_command", "modifiers": { "optional": ["any"] } },
      "to": [{ "key_code": "left_shift", "modifiers": ["left_command", "left_control", "left_optio
      "type": "basic"
    }
  ]
}
```

Move Without Thinking

- `cd -` flips between two dirs; `pushd`/`popd` when juggling more.
- Use globs: `cd ~/Down*`, `ls **/*.m3u8`, `rm ./frames/{01..10}.png`.
- `take new-dir` (from Oh My Zsh) = `mkdir -p && cd`.
- `tree -I 'node_modules|dist|.git' -L 3` before tiling panes so you know what's inside each project.
- Brace expansion for quick scaffolds: `mkdir -p src/{components,hooks}/shared`.

Edit Inline

- Remember the rule: `>` overwrites, `>>` appends, `touch` just timestamps.
- Prepend without editors:

```
echo "first line" | cat - file.md > /tmp/tmpfile && mv /tmp/tmpfile file.md
```

- Follow logs with filtering: `tail -f app.log | rg --line-buffered ERROR`.
- Use `Ctrl+V` newline when building loop bodies directly in the prompt.

Search & Kill (fast version)

- Text search: `rg --heading "TODO" src`.
- Find stray dev servers: `pgrep -a node`, then `kill 12345`.
- Reserve `pkill chrome` (or Slack, etc.) for apps you *want* to nuke entirely; otherwise kill PIDs.
- Background something before closing the tab: `nohup long-task.sh >/tmp/task.log 2>&1 &`.

More depth/tips live in `notes/30-search-scripting/`.

Rename & Batch

```
rename -n 's/ /-/g' *          # dry-run rename
rename 'y/A-Z/a-z/' *          # lowercase

for f in *.png; do             # folder-per-file
  mkdir -p "${f%.*}"
  mv "$f" "${f%.*}/"
done
```

Use loops anytime you catch yourself copy/pasting similar commands. Keep dry runs with `-n` or by prefixing `echo`.

Keep It Lean

- Link out to tmux/yabai/network docs instead of duplicating commands.
- Add new tricks only when they directly speed up navigation, editing, or process control.

Purpose

The apps, terminals, and quick references that sit outside the shell but support it—rename CLI, Finder replacements, zellij hints, etc.

Rename CLI (Perl)

```
rename -n 's/ /-/g' *       # Dry-run replace spaces
rename 'y/A-Z/a-z/' *       # Lowercase filenames
rename 's/^prefix_//' *     # Drop prefix
rename 's/_/-/g' *          # Underscore → dash
rename 's/\.jpeg$/.jpg/' *  # Extension change
```

Flags: `-n` dry run, `-v` verbose, `-f` overwrite existing targets. Always dry-run before mutating a whole directory.

Desktop Utilities

Tool	Purpose	Link
Supercharge	Finder context menus & automations	https://sindresorhus.com/supercharg
Marta	Dual-pane file manager	https://marta.sh
Ghostty / iTerm / Warp	Terminal emulators	https://ghostty.org / https://iterm2.com / https://www.warp.dev
Broot / Zazi	Terminal navigators	—
OrbStack	Docker + lightweight VMs	https://orbstack.dev
Gifski	GIF converter	https://gif.ski
ImageOptim	Lossless image compressor	https://imageoptim.com/mac
Hidden Bar	Menubar cleaner	—

Tool	Purpose	Link
MTMR	Touch Bar customization	https://mtmr.app
Kdenlive	Open-source video editor	https://kdenlive.org
Tmate	Terminal sharing	https://terminaltrove.com/tmate
macFUSE	Mount foreign filesystems	https://github.com/macfuse/macfuse

Terminal Utilities

- `ascii-image-converter`, ImageMagick, Pillow (Python) for quick asset tweaks.
- `Zellij` alternate multiplexer:
  - Modes: `Ctrl+p` (panes), `Ctrl+t` (tabs), `Ctrl+n` (resize), `Ctrl+h` (move), `Ctrl+s` (scroll), `Ctrl+o` (sessions), `Ctrl+g` toggle status bar, `Esc` back to normal.
  - Pane actions: `Ctrl+p n` (new), `Ctrl+p x` (close), arrow keys to move.
  - Tabs: `Ctrl+t n` new, `Ctrl+t <number>` jump.
  - Restore status bar/hints with `Alt+f`.
- Remember `alt + [` maps to `alt + *` in some configs—document weird mappings here as you discover them.

Finder Folder Reminders

Create quick config files:

```
touch ~/.yabairc && chmod +x ~/.yabairc
mkdir -p ~/.config/zellij/layouts
touch ~/.config/zellij/layouts/finder.kdl
echo 'theme "tokyo-night"' > ~/.config/zellij/config.kdl
```

Keeping these centralized prevents the “where did I put that config” shuffle.

Purpose

The “grep goblin” playbook. Covers how the shell expands patterns, when to use `&&` vs `||`, what `$?` actually tells you, and how to dry-run destructive commands before they blitz your files.

Globs & Brace Expansion

Pattern	Meaning	Example
`*`	Any characters in current directory (no slash)	`ls *.png`
`**`	Recursive match (zsh globstar)	`rg -n "TODO" **/*.ts`
`?`	Single character	`ls file?.txt`
`[abc]`	Character class	`ls file[12].txt`
`{a,b,c}`	Brace expansion → literal list	`mkdir {src,lib,test}`

Order of operations: brace expansion → tilde expansion → parameter expansion → command substitution → arithmetic → globbing → word splitting. Understand this so you can predict what `for f in */*.m3u8` really loops over.

Chaining Commands

Syntax	Behavior	Use when…
`cmd1 && cmd2`	Run `cmd2` **only if** `cmd1` exits with code `0` (success)	Deploy steps that must succeed sequentially

Syntax	Behavior	Use when…
`cmd1 || cmd2`	Run `cmd2` **only if** `cmd1` fails (non-zero exit)	Provide fallback / notification
`cmd1 ; cmd2`	Always run both, ignore exit codes	Logging, best avoided in scripts
`cmd &`	Background job, shell continues	Run watchers/uploaders
`cmd1 | cmd2`	Pipe stdout of `cmd1` into stdin of `cmd2`	Filter + transform data (`rg pattern | less`)

Exit Codes

- `$?` holds the exit status of the last command.
- `true` returns `0`, `false` returns `1`.
- In bash/zsh, success == 0, failure == non-zero.
- `set -euo pipefail` makes scripts die on errors, undefined vars, and failed pipes.

```
if ./build.sh; then
  echo "Build succeeded"
else
  echo "Build failed with exit code $?"
fi
```

Dry Runs & Safety Nets

Technique	Example	Why
`echo` prefix	`for f in *.png; do echo mv "$f" dest/; done`	See exactly what would run
`rename -n`	`rename -n 's/ /-/g' *`	Built-in dry run flag
`rsync --dry-run`	`rsync -av --dry-run src/ dst/`	File sync preview
`set -x`	`set -x; script; set +x`	Trace commands as they execute
`git status --short`	Before destructive loops	Ensure clean worktree

Progress Flags

- `pv file | script` to show throughput in pipelines.
- `rsync -P` or `curl -#` for inline progress bars.
- Many copy loops accept `--progress` implicitly (e.g., `cp -v`).

Tests & Conditionals

Test	Meaning	Example
`[ -f file ]`	file exists	`if [ -f "$f" ]; then ... fi`
`[ -d dir ]`	directory exists	`for d in */; do [ -d "$d" ] || continue; done`
`[ -n "$VAR" ]`	non-empty string	Guard required env vars
`[ "$a" = "$b" ]`	string equal	Quote everything

Prefer `[[ ... ]]` in zsh/bash for safer comparisons (`[[ $var == foo* ]]` handles globs without escaping).

Common Chains

```
# Rebuild + deploy only when build passes
npm run build && vercel deploy --prod

# Remove stale rclone folders
for f in folder1 folder2; do
  rclone purge kolkrabbi:path/$f || echo "Failed on $f"
done

# Process multiple formats
for f in *.{png,jpg,jpeg}; do
  ./batch-web-thumb.sh "$f" || break
done
```

Use `set -e` inside scripts to short-circuit on failure, but add `|| true` to commands you're okay failing (`grep pattern || true`).

Debug Checklist

- Run loops with `for f in *.png; do printf '%s\n' "$f"; done` before invoking destructive commands.
- Use `: "${VAR:?Set VAR first}"` to enforce required env vars.
- When pipelines fail silently, add `set -o pipefail`.
- Document expected exit codes (0 success, 1 expected failure, >1 fatal) so future you knows why a script stops.

Examples

```
# 1. Find all m3u8 playlists and sync them
for f in **/*.m3u8; do
  rsync -av --dry-run "$f" deploy@cdn:/playlists/ || break
done

# 2. Kill whatever is on 5173, but show it first
pgrep -a node && kill $(lsof -t -i :5173)

# 3. Rebuild assets; send Slack alert if build fails
npm run build && ./deploy.sh || curl -X POST "$SLACK_WEBHOOK" -d 'build failed'

# 4. Process mixed images with progress
for f in *.{png,jpg}; do
  echo "processing $f"
  ./convert-web-image.sh "$f" 2560 || { echo "failed on $f"; break; }
done

# 5. Remove empty directories safely
find . -type d -empty -print -delete

# 6. Loop through folders and purge rclone targets
for folder in footage renders assets; do
  rclone purge kolkrabbi:library/$folder && echo "cleared $folder"
done

# 7. Copy only when source exists
[[ -f hero.mp4 ]] && cp hero.mp4 deploy/ || echo "hero missing"

# 8. Preview a destructive rename
rename -n 's/ /-/g' *
rename 's/ /-/g' *
```

```bash
# 9. Watch tail output and fail on keyword
tail -f server.log | while read -r line; do
  echo "$line"
  [[ "$line" == *"CRASH"* ]] && pkill server
done

# 10. One-liner to expand multiple folders
mkdir -p 04_assets--library/{category__logos,category__icons,category__textures}
```

Purpose

All the "for f in *" tricks, `find -exec`, `xargs`, and rclone batching you keep reusing. This is the muscle memory chapter for mass renames, migrations, and cleanup jobs.

Loop Patterns

Basic `for` loops

```bash
for f in *.png; do
  ./batch-web-thumb.sh "$f"
done
```

Mixed formats:

```bash
for f in *.{png,jpg,jpeg}; do
  ./convert-web-image.sh "$f" 2560
done
```

Subset:

```bash
for f in 01_mg-type-abstract.png 02_mg-type-abstract.png; do
  ./batch-web-thumb.sh "$f"
done
```

Process & Move

```bash
for f in *.png; do
  mkdir -p "${f%.*}"
  mv "$f" "${f%.*}/"
done
```

`${f%.*}` strips the shortest `.*` suffix (extension), handy for per-file folders.

Dry Run Template

```bash
for f in *.mp4; do
  echo ffmpeg -i "$f" -vf "fps=60" "60fps/$f"
done
```

Swap `echo` for the real command after verifying.

find + -exec

Pattern	Explanation
`find . -type d -name "web" -exec sh -c 'mv "$1"/* "$1/.." && rmdir "$1"' _ {} \;`	Flatten `web` directories
`find . -type f -name "*.m3u8" -exec dirname {} \;`	Locate playlist folders
`find . -maxdepth 2 -type f -print0 | xargs -0 ls -lh`	Null-safe piping

`-type d/f/l` filters directories/files/symlinks. `-exec sh -c '...' _ {}` lets you write multi-step commands per match.

xargs

```
ls -1v *.png | xargs -I {} magick "{}" -resize 2000x2000 "resized/{}"
```

- `-0`/`-print0` for files with spaces.
- `-P 4` for parallel execution when commands are CPU-bound (careful with shared outputs).

rclone Chains

```
# Multiple purges
for f in folder1 folder2; do
  rclone purge kolkrabbi:path/$f
done

# Tree listing with excludes
rclone tree kolkrabbi:path --exclude "segment_*.ts"
# or filter syntax
rclone tree kolkrabbi:path --filter "- segment_*.ts"
```

`rclone` doesn't accept multiple paths in a single `purge`, hence the `for` pattern.

Symlinks Refresher

```
ln -s /path/to/original /path/to/symlink
```

Use symlinks when one source must appear in multiple paths (fonts, configs, version switching). Pair with `tree -F` to verify `@` markers.

Process Kill / Restart Loops

```
for svc in skhd yabai; do
  launchctl bootout gui/$(id -u)/com.koekeishiya.$svc 2>/dev/null || true
  brew services restart $svc
done
```

Keeps service restarts consistent without copying commands from old notes.

Tips

- Prefer `while IFS= read -r line; do ... done < filelist.txt` when looping over arbitrary lists.
- `seq 1 10` or brace expansion (`for n in {1..10}`) for numeric loops.
- Document long-running loops with comments; add `sleep` inside watchers to avoid CPU spikes.

Purpose

Keep all the `rg`, `grep`, `find`, and process search nuggets in one place so you can channel the "search bashing" energy without rewiring the muscle memory each time.

Ripgrep (`rg`)

```
rg --heading '^# ' notes        # List headings in the repo
rg -n "ssh" notes/50-networking # Match text with line numbers
rg -g '*.md' "magick"           # Restrict to Markdown files
rg --files -g '*network*'       # List files by glob
```

Useful flags: `-n` line numbers, `-C2` context, `--json` structured output (pipe into `jq`), `-g` glob filter, `--iglob` case-insensitive glob.

Grep / Egrep

```
grep -R "TODO" src/                    # Recursive search
grep -n "pattern" file                 # Single file with line number
grep -E "foo|bar" file                 # Extended regex (alternation)
grep -v "ignore" file                  # Invert match
```

When exploring interactively, pair with `less`:

```
rg "error" log.txt | less -R
```

Find

Task	Command
Find files by name	`find . -name "*.jpg"`
Case-insensitive	`find . -iname "readme.md"`
Prune node_modules	`find . -name "node_modules" -prune -o -name "*.ts" -print`
Execute command	`find . -type f -name "*.m3u8" -exec dirname {} \;`

Remember `-print` vs implicit print; when you use `-exec`, the default print is suppressed unless you add `-print`.

Tree

```
tree -I 'node_modules|dist|build|.git' -L 3
find . -print | sed -e 's;[^/]*/;|__ ;g;s;__|;  |;g'
```

Use `ls **/*.txt` (zsh glob) for quick recursive glances when `tree` isn't installed.

Process Search

```
ps aux | grep node
pgrep -fl vite
pgrep -a node          # show command lines
kill 15112             # kill specific PID
pkill node             # kill all node processes (careful!)
```

Workflow:

1. `pgrep -a node` to list what would die.
2. `kill PID1 PID2` to surgically stop servers.
3. `pkill chrome` for apps where you actually want *everything* gone.

Network Discovery Snippets

From the old "terminal-network" files:

```
networksetup -listallhardwareports      # Interfaces on macOS
ipconfig getifaddr en0                  # Ethernet IP
arp -a                                  # ARP table
nmap -sS 192.168.1.0/24                 # SYN scan (requires nmap)
sudo lsof -i :3000                      # What's listening on port 3000
```

Keep more in `notes/50-networking`, but link it here for quick search references.

Pattern Cookbook

Pattern	When to use
`ls **/*.m3u8`	Find CDN playlists anywhere under current tree
`rg --files -g '*.sh'`	List shell scripts
`find . -type d -empty -delete`	Remove empty directories
`grep -R "ssh biskup" notes`	Check where credentials appear
`lsof -iTCP -sTCP:LISTEN`	All listening sockets

Tips

- Pipe file lists into `fzf` when you want fuzzy selection: `rg --files | fzf`.
- Combine `rg --json` with `jq` for structured automation.
- Practice toggling `setopt extended_glob` in zsh for advanced glob patterns (`^` (negate), `(.)#` for repetition).

Purpose

Bootstrap Lazy.nvim, remember the essential modes, and capture the key mappings you keep using so the editor stays consistent across machines.

Install Flow

```
git clone https://github.com/folke/lazy.nvim ~/.local/share/nvim/lazy/lazy.nvim
mkdir -p ~/.config/nvim
nvim ~/.config/nvim/init.lua
```

Add Lazy to the runtime path and set up plugins:

```
vim.opt.rtp:prepend("~/.local/share/nvim/lazy/lazy.nvim")

require("lazy").setup({
  {
    "nvim-tree/nvim-tree.lua",
    dependencies = { "nvim-tree/nvim-web-devicons" },
    config = function()
      require("nvim-tree").setup({})
    end,
  },
})
```

Install plugins inside Neovim: `:Lazy sync`.

Keybindings

```
vim.keymap.set("n", "<C-n>", ":NvimTreeToggle<CR>")
```

- Open Neovim in project root: `nvim .`
- Save & quit: `:w`, `:q`, `:wq`, `:q!`

Modes

Mode	Enter	Exit	Notes
Normal	default	`Esc`	Navigate, delete, copy
Insert	`i` / `a` / `A`	`Esc`	Type text
Visual	`v` / `V` / `Ctrl+v`	`Esc`	Select text
Command	`:`	`Enter`	Run commands

Movement reminders (normal mode):

```
w / b          # forward/back word
0 / $          # start/end of line
gg / G         # top/bottom of file
dw / dd        # delete word / line
```

Common Actions

Task	Command
Open file tree	`Ctrl+n` (mapping above)
New line below	`o` (normal mode)
Undo / redo	`u` / `Ctrl+r`
Search	`/pattern` then `n` / `N`
Replace	`:%s/old/new/g`

Troubleshooting

- If Lazy.nvim can't clone, check git access or remove `~/.local/share/nvim/lazy` and re-run.
- Ghostty/iTerm must send proper keycodes (`option` as meta) for custom mappings.
- Keep Neovim config under `.dotfiles` for reproducibility.

Purpose

Everything you need to manage sessions, panes, and quick config without scanning two separate tmux docs.

Quick Start

```
tmux new -s tor_server_type
```

- `Ctrl+b d` → detach
- `tmux attach -t tor_server_type` → reattach
- `tmux ls` → list sessions
- `tmux kill-session -t tor_server_type` → nuke session

Pane & Window Controls

Action	Keys
Split vertically	`Ctrl+b %`
Split horizontally	`Ctrl+b "`
Switch panes	`Ctrl+b o` or arrow keys
Resize pane	`Ctrl+b` then `Alt` + arrows (or `Ctrl+b :resize-pane -D 5`)
Close pane	`Ctrl+b x`
New window	`Ctrl+b c`
Next/Prev window	`Ctrl+b n` / `Ctrl+b p`
Rename window	`Ctrl+b ,`
Detach	`Ctrl+b d`

Session Management

```
tmux new -s name               # create
tmux attach -t name            # attach
tmux switch -t other           # swap
tmux kill-session -t name      # kill
tmux kill-server               # kill every session
```

Use naming conventions (`client_task`, `lab_notes`) so `tmux ls` stays readable.

Configuration Tips

- Set default terminal to 256 colors in `.tmux.conf`: `set -g default-terminal "tmux-256color"`.
- Enable mouse support: `set -g mouse on`.
- Activity alerts: `setw -g monitor-activity on`.
- Center window list: `set -g status-justify centre`.
- Maximize / restore pane: bind something like `bind-key m resize-pane -Z`.

Tree Helper

Pair tmux sessions with `tree -I 'node_modules|dist|build|.git' -L 3` so you always know project layout when hopping between panes.

Troubleshooting

- If Ghostty/iTerm keybindings conflict, check terminal settings for `Option sends Esc`.
- If attachments fail, ensure `$TERM` outside tmux matches inside (`tmux show -g -w`). Use `tmux -u` if unicode glitches occur.
- Keep tmux config under `.dotfiles` so new machines pick it up automatically.

Purpose

Condenses the quick Yabai checklist, layout cheatsheet, and troubleshooting steps so you can reset the tiling stack fast.

Setup Checklist

```
touch ~/.yabairc && chmod +x ~/.yabairc
```

Yabai executes `.yabairc` as a script, so it must be executable. Keep SKHD bindings under `~/.skhdrc`.

Service Control

```
brew services stop skhd
brew services start skhd

brew services stop yabai
brew services start yabai
```

When things get weird:

```
launchctl bootout gui/$(id -u)/com.koekeishiya.skhd 2>/dev/null
launchctl bootout gui/$(id -u)/com.koekeishiya.yabai 2>/dev/null
pkill -9 skhd
pkill -9 yabai
killall Dock
```

Use `pgrep -fl skhd` / `pgrep -fl yabai` to see running instances before killing them.

Default Layouts & Actions

Action	Command
Layouts	`yabai -m config layout bsp|stack|float`
Focus windows	`yabai -m window --focus west|east|north|south`
Move windows	`yabai -m window --warp west|east|north|south`
Resize	`yabai -m window --resize left:50:0` (example)
Toggle float	`yabai -m window --toggle float`

Action	Command
Rotate tree	`yabai -m space --rotate 90`
Balance windows	`yabai -m space --balance`

Padding & Gaps

```
yabai -m config window_gap 12
yabai -m config top_padding 32
yabai -m config left_padding 12
```

Mouse + Focus

```
yabai -m config mouse_follows_focus on
yabai -m config focus_follows_mouse autofocus
```

Quick Diagnostics

Need	Command
See shell, env, aliases, options, PATH	`echo $SHELL`, `printenv`, `alias`, `setopt`, `echo $PATH`
Restart SKHD without full reboot	`pkill skhd && skhd &`
Hard kill	`sudo killall -9 skhd`, `sudo killall -9 yabai`, `sudo killall Dock`

`-9` sends SIGKILL (“no cleanup, just die”). Use it when `brew services stop` can’t stop the daemons.

Config Tips

- Keep hotkeys descriptive in `.skhdrc` and comment sections (`# Window movement`, `# Apps`).
- Document every `yabai -m rule --add ...` in this file to avoid duplicate tiling exceptions.
- Add `yabai --check` to login items or use `launchctl bootstrap` once the config is stable.

Troubleshooting

- Missing key hints in Zellij? Toggle with `Ctrl+g` or `Alt+f`.
- Figma capturing keys? `pkill skhd` temporarily, `skhd &` when done.
- Need clean restart: `launchctl bootout ...` (above) + `brew services start`.
- When `.yabairc` changes don’t apply, run `yabai --load-sa && yabai --restart-service`.

Purpose

Condenses the “ping/kill server” notes into one table for managing processes, freeing ports, and knowing when to use `pkill` vs `kill`.

Ping & Link Checks

```
[
  {"task": "list interfaces", "cmd": "networksetup -listallhardwareports"},
  {"task": "ethernet ip", "cmd": "ipconfig getifaddr en0"},
  {"task": "wifi ip", "cmd": "ipconfig getifaddr en1"},
  {"task": "ping iPad", "cmd": "ping 192.168.2.2"},
```

```
  {"task": "arp table", "cmd": "arp -a"},
  {"task": "trace route", "cmd": "traceroute 192.168.2.2"},
  {"task": "static ip (Mac)", "path": "System Settings → Network → Ethernet → Configure IPv4 → Manual"}
  {"task": "static ip (iPad)", "path": "Settings → Ethernet → Configure IP → Manual"}
]
```

Vite LAN config: `server: { host: true, port: 5173 }` then access via `http://192.168.2.1:5173`.

kill vs pkill

```
[
  {"cmd": "pgrep -a name", "use": "Preview targets", "example": "pgrep -a node"},
  {"cmd": "kill PID", "use": "Single process only", "example": "kill 15112"},
  {"cmd": "kill PID1 PID2", "use": "Multiple PIDs", "example": "kill 12345 67890"},
  {"cmd": "pkill name", "use": "Kill all by name", "example": "pkill node"},
  {"cmd": "pkill -9 name", "use": "Force everything", "example": "pkill -9 yabai"}
]
```

Workflow:

1. `pgrep -a node`
2. Kill only what you need: `kill 12345 45678`
3. Reserve `pkill node` for "nuke all dev servers"

Chrome is a good `pkill` target (many helper processes, safe to restart). Dev servers? Use `kill PID`.

Background Watchers

Example CPU limiter for Claude (`~/limitclaude.sh`):

```
#!/bin/zsh
while true; do
  if pgrep claude >/dev/null; then
    echo "Claude detected - applying CPU limit..."
    sudo cpulimit -e claude -l 40
  else
    sleep 5
  fi
done
```

Run detached: `nohup ~/limitclaude.sh >/tmp/limitclaude.log 2>&1 &`.

Port Cleanup

```
sudo lsof -i :5173            # check Vite port
kill $(lsof -t -i :5173)      # kill culprit
lsof -nP -iTCP -sTCP:LISTEN   # all listening ports
```

When to Use `pkill -9`

- Yabai/SKHD hung after config change
- Apps ignoring polite SIGTERM
- Services that respawn automatically (Dock)

Remember `-9` = immediate termination. No cleanup, no autosave. Confirm with `pgrep` before pulling the trigger.

Logging

- Keep a `network-log.md` entry when debugging so you know which device had which manual IP.
- Use `tee` to capture command output: `ping 192.168.2.2 | tee logs/ping-ipad.log`.

Purpose

Centralizes the `terminal-network-03/04/05` notes: nmap scans, lsof/netstat combos, service control, and what commands map to which processes.

Quick Reference

```
[
  {"task": "listening ports", "command": "sudo lsof -nP -iTCP -sTCP:LISTEN"},
  {"task": "alt (Linux)", "command": "sudo netstat -plant"},
  {"task": "alt (macOS)", "command": "sudo lsof -PiTCP -sTCP:LISTEN"},
  {"task": "modern Linux", "command": "ss -tulpn"},
  {"task": "node servers", "command": ["lsof -i :3000", "pgrep -a node", "ps aux | grep node"]},
  {"task": "kill PID", "command": "kill <PID>"},
  {"task": "force kill", "command": "kill -9 <PID>"}
]
```

nmap Recipes

```
# Scan all hosts on subnet
nmap -sn 192.168.1.0/24

# Scan ports with service detection
sudo nmap -sS -sV 192.168.1.38

# Find open ports with scripts
sudo nmap -sC -sV target

# Ping sweep
nmap -sn 192.168.1.0/24
```

Install via `brew install nmap` (macOS) or `sudo apt install nmap` (Linux).

Process Management

```
[
  {"step": "list", "cmd": "pgrep -a node"},
  {"step": "see command", "cmd": "ps -p <PID> -o command"},
  {"step": "kill one", "cmd": "kill <PID>"},
  {"step": "kill multiple", "cmd": "kill <PID1> <PID2>"},
  {"step": "kill by name", "cmd": "pkill node"},
  {"step": "force", "cmd": ["kill -9 <PID>", "pkill -9 node"]}
]
```

Use `pgrep -a` first so you know what you are about to nuke.

Device Discovery

```
arp -a                          # ARP table
networksetup -listallhardwareports
ipconfig getifaddr en0          # IPv4 Mac Ethernet
traceroute 192.168.2.2          # Path to device
```

Example Workflows

Find stray dev servers

```
# Show port 5173 usage
sudo lsof -i :5173
# Kill if needed
kill <PID>
```

Audit Node servers

```
pgrep -a node           # list
lsof -i :3000           # check port 3000
kill $(lsof -t -i :3000)  # kill running process on port 3000
```

Scan subnet for hosts

```
nmap -sn 192.168.1.0/24
# or only specific range
nmap -sS 192.168.1.10-30
```

Tools

- `lsof`: open files and sockets (macOS friendly).
- `netstat`: older but everywhere.
- `ss`: modern Linux alternative.
- `nmap`: scanning and service fingerprinting.

Tips

- Always run scans with permission; `sudo` only when necessary.
- Keep `nmap` outputs in logs (`nmap -oN scan.txt ...`) for historical tracking.
- Use `watch -n 2 "lsof -i :3000"` to monitor ports over time.
- Big networks? Use `masscan` or `zmap`, but log results responsibly.

Purpose

Everything from "what's my IP?" to "why is SSH yelling about host keys?", plus when to use VPN, Tailscale, or self-hosted tunnels.

Quick SSH Connect

```
ssh username@mac_ip_address
ssh biskup@192.168.1.38
```

Find the target IP:

- Terminal: `ifconfig` → look at `en0` (Wi-Fi) or `en1` (Ethernet)
- `ip a show` (if `iproute2mac` installed)
- GUI: → System Settings → Network → pick active interface

Local hostname (e.g., `kolkrabbi.local`) lives in System Settings → General → Sharing. Toggle "Use dynamic global hostname" there if you want macOS to update DNS automatically.

Understanding Host Key Prompts

```
The authenticity of host '192.168.1.38' can't be established.
ED25519 key fingerprint is SHA256:...
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

1. On the server, display fingerprints:

```
ssh-keygen -l -f /etc/ssh/ssh_host_ed25519_key.pub
ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
```

2. Compare with the prompt; if they match, type `yes`.
3. To replace stale keys: `ssh-keygen -R 192.168.1.38`.

If you immediately get `Broken pipe`, bounce Remote Login, confirm firewall permissions, run `log stream --predicate 'sender == "sshd"' --info` to see why, and temporarily move `~/.zshrc` if startup scripts are crashing the shell.

IP, MAC, Username


```
{
  "ip": {
    "ethernet": "ipconfig getifaddr en0",
    "wifi": "ipconfig getifaddr en1"
  },
  "mac-address": "System Settings → Network → Details → Hardware",
  "username": ["whoami", "id -un", "Finder → Go → Home"],
  "rename-user": "Login with another admin → System Settings → Users & Groups → Control-click → Advance
}
```


VPN Primer


```
[
  {"type": "tailscale", "install": ["Laptop", "iMac", "Ubuntu VM"], "result": "Each device gets 100.x.x
  {"type": "traditional (Mullvad/Proton)", "flow": "You → provider server → Internet", "bestFor": "geo-
  {"type": "self-hosted (WireGuard/OpenVPN)", "location": "Wherever the server lives", "bestFor": "full
]
```


Direct Device Map


```
[
  {"device": "MBP 13\"", "ip": "192.168.1.36", "router": "192.168.1.254"},
  {"device": "iMac", "ip": "192.168.1.38", "hostname": "kolkrabbi.local"},
  {"device": "Jellyfin", "ip": "192.168.1.151", "url": "http://192.168.1.151:8096"}
]
```


VPN Decision Tree

1. Need remote access to home gear? → Tailscale.
2. Need to appear in another country for pricing/content? → Commercial VPN (Mullvad, Proton).
3. Need total control? → Self-host WireGuard on a cloud server in the desired region.

Troubleshooting Checklist

- SSH prompt loops after `yes`? Check `~/.ssh/known_hosts` permissions (should be 600).
- `ssh_dispatch_run_fatal: Broken pipe` → server-side logs, firewall, shell configs.
- VPN not routing? `tailscale status` or provider's CLI for connection info.
- Always keep a local admin account in case you rename usernames incorrectly.

Purpose

Collects the iperf3 snippet, SMB copy sanity checks, and quick reference for connected devices/networks.

iperf3

```
brew install iperf3

# On iMac (server)
iperf3 -s

# On MacBook (client)
iperf3 -c <iMac IP>
```

If iMac is on Ethernet via dock:

- Check System Settings → Network → Ethernet (yellow) to read the 169.254.x.x address (Thunderbolt Bridge) or the DHCP address if connected to router.
- Expect ~100–115 MB/s on gigabit, more on Thunderbolt bridge.

When wireless: test by copying a large file over Finder (`smb://iMac.local`) and watching Finder's transfer speed.

Device / Network Matrix


```
{
  "devices": [
    {"name": "MBP 13\"", "ip": "192.168.1.36", "router": "192.168.1.254", "wifi-mac": "ba:c9:1e:c1:2f:f
    {"name": "iMac", "ip": "192.168.1.38", "router": "192.168.1.254", "wifi-mac": "5a:1a:24:7a:d4:05",
    {"name": "Jellyfin", "ip": "192.168.1.151", "url": "http://192.168.1.151:8096"}
  ],
  "networks": {
    "primary_ssid": "5TAGVKM9CF",
    "extender_ssid": "2840",
    "nas_ssid": "1B-9A-A7-D2"
  }
}
```


Keep the table updated so scripts know which host to target.

Ping Reference

```
networksetup -listallhardwareports
ipconfig getifaddr en0     # Ethernet
ipconfig getifaddr en1     # Wi-Fi
ping 192.168.2.2
```

Use manual link-local addressing for direct Mac ↔ iPad Ethernet:

- Mac: 192.168.2.1 / 255.255.255.0
- iPad: 192.168.2.2 / 255.255.255.0

Notes

- If iperf3 throws permission errors, ensure firewall allows incoming connections for iperf3.
- Keep SMB credentials stored in Keychain for faster Finder reconnects.

Purpose

Replaces the multiple `terminal-cdn-*.md` variants with one source of truth for preparing masters, generating HLS ladders, and keeping folder structures consistent.

Library Layout


```
{
  "/cdn": {
    "project-alpha": {
      "subproject-01": ["master.m3u8", "index.m3u8", "poster.jpg", "720p/segment_000.ts…", "1080p/segme
      "subproject-02": ["master.m3u8", "1080p/segment_000.ts…"],
      "subproject-03": ["hero.mp4", "hero.webp", "thumbnail.jpg"],
      "subproject-04": ["audio.mp3", "audio.aac", "waveform.json", "cover.jpg"],
      "subproject-05": ["image_1x.jpg", "image_2x.jpg", "image_4x.jpg"]
    },
    "project-beta": "same pattern"
  }
}
```

Keep posters, thumbnails, and playlists beside the segments they describe.

Workflow

1. Export master from Resolve (ProRes 422 / DNxHR / high bitrate H.264).
2. Ensure frame rate matches timeline.
3. Run script to generate HLS variants.
4. Upload to CDN (rclone/s3/ftp).

make_hls.sh

```
#!/bin/bash

INPUT=$1
OUTPUT=hls_output

mkdir -p "$OUTPUT"

ffmpeg -y -i "$INPUT" \
  -filter:v:0 "scale=640:-1"   -c:v:0 libx264 -b:v:0 800k  -preset slow -profile:v:0 high \
  -filter:v:1 "scale=854:-1"   -c:v:1 libx264 -b:v:1 1500k -preset slow -profile:v:1 high \
  -filter:v:2 "scale=1280:-1"  -c:v:2 libx264 -b:v:2 2500k -preset slow -profile:v:2 high \
  -filter:v:3 "scale=1920:-1"  -c:v:3 libx264 -b:v:3 4500k -preset slow -profile:v:3 high \
  -map 0:v -map 0:v -map 0:v -map 0:v \
  -an \
  -pix_fmt yuv420p \
  -f hls \
  -hls_time 4 \
  -hls_playlist_type vod \
  -hls_segment_filename "$OUTPUT/%v/segment_%03d.ts" \
  -master_pl_name master.m3u8 \
  -var_stream_map "v:0,name:360p v:1,name:480p v:2,name:720p v:3,name:1080p" \
  "$OUTPUT/%v/index.m3u8"
```

Usage:

```
chmod +x make_hls.sh
./make_hls.sh master.mov
```

Upload Helpers

- `rclone copy hls_output/ remote:cdn/project/subproject`
- `rclone tree remote:cdn/project --exclude "segment_*.ts"` to inspect without noise.
- Use loops from `loops-and-batch-ops.md` for multiple targets.

Validation

- `ffprobe -select_streams v -show_entries stream=avg_frame_rate,r_frame_rate -of default=nk=1 input.mp4` before conversion.
- `mediastreamvalidator master.m3u8` (macOS) for HLS compliance.
- Watch the `.m3u8` in VLC to verify variants.

Notes

- Keep a `poster.jpg` and `thumbnail.jpg` per stream for web usage.
- Document bitrates/resolutions per project so designers know what to expect.
- For audio-only, keep `.aac` and `.json` waveform assets adjacent to playlists.

Purpose

All the tiny scripts and shell snippets you kept in `terminal-cli-library.md`, consolidated with explanations.

Image/Web Batches

```
for f in *.png; do ./batch-web-thumb.sh "$f"; done
for f in *.{png,jpg,jpeg}; do ./batch-web-thumb.sh "$f"; done
for f in *.png; do ./convert-web-image.sh "$f" 2560; done
```

Flatten Subfolders

```
find . -type d -name "web" -exec sh -c 'mv "$1"/* "$1/.." && rmdir "$1"' _ {} \;
# Move assets up two levels
mv */web/* . && mv */*.jpg . && rm -r */
```

`-exec sh -c '...' _ {}` opens a shell per match so you can chain multiple commands.

Symlink Refresher

```
ln -s /path/to/original /path/to/symlink
```

Benefits: single source of truth, shorter paths, easy version switching, shared assets between projects.

rm -rf

`rm -rf` = recursive + force. Triple-check path before running. Prefer `trash` CLI when possible.

rclone Examples

```
rclone purge kolkrabbi:kolkrabbi/website/asset-library/home-about
rclone purge kolkrabbi:path/folder1 && rclone purge kolkrabbi:path/folder2
rclone tree kolkrabbi:path --exclude "segment_*.ts"
rclone tree kolkrabbi:path --filter "- segment_*.ts"
```

Note: rclone doesn't support multiple paths per command, so chain or loop.

Loop Concepts

```
for f in folder1 folder2; do
  rclone purge kolkrabbi:path/$f
done
```

Use this for anything repetitious; see `loops-and-batch-ops.md` for more.

Misc Helpers

```
# Create folder per file and move file inside
for f in *.png; do mkdir -p "${f%.*}" && mv "$f" "${f%.*}/"; done

# Exclude segments with tree
tree -I "segment_*.ts"
```

Concept Reminders

- Chains vs loops: `cmd1 && cmd2` ensures sequential success, `for` loops iterate explicitly.
- Use `rename -n` for dry runs whenever you're about to rename/move hundreds of files.
- Document new recipes here to keep everything central.

Purpose

Reference for `cpulimit`, watcher scripts, and when to clamp runaway processes (Claude, ffmpeg, etc.).

cpulimit Basics

Step	Command	Notes
Install	`brew install cpulimit`	One-time
Check process	`ps aux | grep claude` or `pgrep -fl claude`	Confirm it's running
Limit CPU (foreground)	`sudo cpulimit -e claude -l 40`	Caps at 40%
Limit CPU (background)	`sudo cpulimit -e claude -l 40 &`	Frees terminal
Lazy mode	`sudo cpulimit -e claude -l 40 -z &`	Exit quietly if not running

Watcher Script

Save as `~/limitclaude.sh`:

```
#!/bin/zsh
while true; do
  if pgrep claude >/dev/null; then
    echo "Claude detected - applying CPU limit..."
    sudo cpulimit -e claude -l 40
  else
    sleep 5
  fi
done
```

Make executable: `chmod +x ~/limitclaude.sh`.

Run detached:

```
nohup ~/limitclaude.sh >/tmp/limitclaude.log 2>&1 &
```

Stop watcher: `pkill -f limitclaude.sh`.

Aliases

`.zshrc` entry:

```
alias limitclaude='sudo cpulimit -e claude -l 40 &'
```

Source file (`source ~/.zshrc`) after editing.

Tips

- `cpulimit -p <PID>` if you need PID-specific control.
- Combine with `renice` to lower process priority.
- For GPU-bound tasks, check `powermetrics` or vendor-specific tools.

Purpose

Keep the workflows you actually use—not a full Git book. This is the bootstrapping flow plus the handful of commands that speed up daily commits.

Bootstrap a Repo

```
git init
git add .
git commit -m "Initial commit"
git remote add origin git@github.com:you/project.git
git branch -M main
git push -u origin main
```

Need identity? `git config --global user.name "Your Name"` and `.email "...@..."` once per machine.

Day-to-Day Loop

```
git status -sb            # short+clean status
git add -p src/file.ts    # stage only what changed
git diff --stat           # sanity check
git commit -m "scope: detail"
```

- `git switch -c feature/x` before you start so work stays isolated.
- `git stash push -m "context"` if you need to hop branches mid-change.
- `git log --oneline --graph --decorate --all | head` for a quick history view.

Share Work

```
git push -u origin feature/x
```

`-u` saves the upstream so future pushes are just `git push`.

Need to change remotes? `git remote set-url origin git@github.com:new/repo.git`.

When Things Go Sideways

- **Push rejected** (you're behind): `git pull --rebase origin main`, resolve, then `git push`.
- **Detached HEAD:** `git switch -c rescue-work`, then merge or cherry-pick as needed.
- **Committed a secret:** rotate the secret *and* rewrite history (`git filter-repo` or BFG). Don't just delete in the next commit.

Keep Handy

- `.gitignore` templates: https://gitignore.io
- `pre-commit` hooks for lint/test automation if you want guardrails.
- Always run `git status -sb` before closing a pane so you know what's left dirty.

Purpose

One script (`make-notes-pdf.sh`) that walks every Markdown file in `notes/`, concatenates them in a deterministic order, and feeds the result to `pandoc` so you can snapshot the entire collection as a single PDF.

Tooling

```
{
  "pandoc": {
    "install": "brew install pandoc",
```

```
    "purpose": "convert Markdown → PDF/HTML/Docx",
    "pdf-engines": ["wkhtmltopdf", "xelatex", "weasyprint", "default (pdflatex)"],
    "notes": "pass --pdf-engine=<engine> if you prefer something other than the default"
  }
}
```

Pandoc is the heavy lifter: it reads Markdown (including frontmatter) and emits the PDF. Install once via Homebrew (or your package manager of choice). The script below calls `pandoc` with `xelatex` (bundled with `mactex-no-gui/basictex`) and sets the main font to Menlo so Unicode box characters survive the export.

Other quick conversions

```
# Markdown → HTML (single file)
pandoc notes/10-setup/shell-and-dotfiles.md -o shell-and-dotfiles.html

# Markdown → Word / DOCX
pandoc notes/30-search-scripting/globbing-and-chaining.md -o globbing.docx

# Entire folder → EPUB (ebook)
find notes -name '*.md' -print0 | sort -z | xargs -0 pandoc -o notes.epub

# Slides (Reveal.js)
pandoc deck.md -t revealjs -s -o deck.html
```

Use these as starting points if you need formats beyond PDF.

Script

```
#!/usr/bin/env bash
set -euo pipefail

OUT="notes.pdf"
TMP_DIR=$(mktemp -d)
trap 'rm -rf "$TMP_DIR"' EXIT

COMBINED="$TMP_DIR/combined.md"
touch "$COMBINED"

find notes -name '*.md' -print0 | sort -z | while IFS= read -r -d '' file; do
  printf '\n\n<!-- %s -->\n\n' "$file" >> "$TMP_DIR/combined.md"
  cat "$file" >> "$TMP_DIR/combined.md"
  printf '\n' >> "$TMP_DIR/combined.md"
done

pandoc "$TMP_DIR/combined.md" \
  --pdf-engine=xelatex \
  -V mainfont="Menlo" \
  -V geometry:a4paper,margin=2cm \
  -V colorlinks=true \
  -V linkcolor=blue \
  -V urlcolor=blue \
  -V hyperrefoptions=breaklinks=false \
  -o "$OUT"
echo "Wrote $OUT"
```

Save it at repo root (`make-notes-pdf.sh`) and `chmod +x` so it's executable.

Usage

```
brew install pandoc                          # Markdown converter
brew install --cask mactex-no-gui            # provides xelatex + fonts (once)
```

```
./make-notes-pdf.sh                    # produces notes.pdf
open notes.pdf                         # optional preview on macOS
```

The script:

- Uses `find … | sort -z` to ensure files are processed in a consistent order.
- Writes a temporary combined Markdown file with HTML comments indicating the source of each section.
- Calls `pandoc` to generate `notes.pdf` in the repo root.
- Cleans up the temp directory automatically via `trap`.

If you want a different PDF engine (`wkhtmltopdf`, `xelatex`, etc.) pass `--pdf-engine=…` to the `pandoc` command inside the script.

Purpose

One table covering Claude Code, Codex, Gemini, CrewAI, and Simon Willison's `llm` CLI so you don't have to jump between docs.

Quick Reference


```
[
  {
    "tool": "Claude Code",
    "install": "pip install claude-code",
    "commands": ["claude", "claude \"prompt\"", "claude --model sonnet", "claude --version"]
  },
  {
    "tool": "Codex CLI",
    "install": "bundled with Codex",
    "commands": ["codex", "codex \"prompt\"", "codex --help"]
  },
  {
    "tool": "Gemini CLI",
    "install": "pip install google-generativeai",
    "commands": ["gemini", "gemini chat", "gemini \"prompt\"", "gemini --model pro"]
  },
  {
    "tool": "CrewAI",
    "install": "pip install 'crewai[tools]'",
    "commands": ["crewai create crew my_project", "crewai run", "crewai train", "crewai test", "crewai
  },
  {
    "tool": "LLM",
    "install": "pip install llm",
    "commands": ["llm \"prompt\"", "llm -m model \"prompt\"", "llm models", "llm keys set openai", "llm
  }
]
```


Examples

```
claude "refactor this component for better performance"
echo "console.log('hello')" | llm "explain this code"
llm -m claude-3-sonnet "write unit tests"
crewai create crew research_team && cd research_team && crewai run
gemini chat
```

Tips

- Use environment variables for API keys (`CLAUDE_API_KEY`, `OPENAI_API_KEY`, etc.).

- Keep prompts inside scripts when building repeatable flows (see `loops-and-batch-ops.md` for piping patterns).
- Document agent outputs in project folders for traceability.

Purpose

Collects the 60fps conversion recipes and HLS-related verifications.

Inspect Frame Rate

```
ffprobe -select_streams v -show_entries stream=avg_frame_rate,r_frame_rate -of default=nk=1 input.mp4
```

Constant 60 fps (Recommended)

```
ffmpeg -i input.mp4 \
  -vf "fps=60" \
  -vsync cfr \
  -c:v libx264 -preset slow -crf 18 \
  -pix_fmt yuv420p \
  -c:a copy \
  output_60fps.mp4
```

- Duplicates frames when source <60 fps.
- Drops frames if source >60 fps.
- Keeps audio in sync; no interpolation artifacts.

Motion Interpolation (Slower, Higher Quality)

```
ffmpeg -i input.mp4 \
  -vf "minterpolate=fps=60:mi_mode=mci:mc_mode=aobmc:me_mode=bidir" \
  -c:v libx264 -preset slow -crf 18 \
  -pix_fmt yuv420p \
  -c:a copy \
  output_60fps_interp.mp4
```

Use when you must smooth 24/30 fps footage and can afford the render time.

Notes

- Keep masters in mezzanine codecs (ProRes/DNxHR) before encoding for HLS or streaming.
- Confirm final bitrate/resolution before uploading to CDN.
- Combine with `cdn-and-infra.md` for playlist generation.

Purpose

All the image-to-PDF conversions and crop scripts in one place.

Images → PDF

ImageMagick (portable)

```
magick *.jpg output.pdf
magick *.png *.jpg *.jpeg output.pdf      # mixed formats
magick *.jpg -quality 90 output.pdf       # control JPEG quality
magick *.jpg -resize 2000x2000 output.pdf # resize before bundling
```

Order is filename-based; use `ls -1v + xargs` if you need natural sort.

img2pdf (lossless)

```
img2pdf *.jpg -o output.pdf
img2pdf --auto-orient *.jpg -o output.pdf
ls -1v *.png *.jpg | xargs img2pdf -o output.pdf
```

Install: `sudo apt install img2pdf` (Linux). Keeps resolution + compression intact.

macOS Preview via CLI

```
sips -s format pdf *.jpg --out output.pdf
```

Mixed Pipeline Example

```
ls -1v *.jpg | xargs magick -quality 92 -resize 2400x2400 portfolio.pdf
```

Crop Scripts (ImageMagick)

```
#!/usr/bin/env bash
set -euo pipefail

SOURCE="${1:?Usage: crop2000x2500 source.jpg [output.jpg]}"
OUT="${2:-${SOURCE%.*}_2000x2500.${SOURCE##*.}}"

magick "$SOURCE" \
  -gravity center \
  -crop 2000x2500+0+0 \
  +repage \
  "$OUT"
```

Gravity Variants

```
magick "$SOURCE" \
  -gravity north \
  -crop 2000x2500+0+0 \
  +repage \
  "$OUT"
```

Gravity map:

```
northwest   north   northeast
west        center  east
southwest   south   southeast
```

Parameterized Script

```
#!/usr/bin/env bash
set -euo pipefail

SOURCE="${1:?Usage: crop2000x2500 image.(tif|tiff|png|jpg) [gravity] [scale:50|75|100] [output.jpg]}"
GRAVITY="${2:-center}"
SCALE="${3:-100}"
OUT="${4:-${SOURCE%.*}_2000x2500_${GRAVITY}_${SCALE}pct.jpg}"

magick "$SOURCE" \
  -auto-orient \
  -resize "${SCALE}%" \
  -gravity "$GRAVITY" \
  -crop 2000x2500+0+0 \
  +repage \
  -flatten \
  -strip \
```

```
  -colorspace sRGB \
  -depth 8 \
  -quality 80 \
  "$OUT"
```

Usage:

```
crop2000x2500 scan.tiff north 50
crop2000x2500 portrait.png center 75 hero.jpg
```

Misc

- `pdf2svg input.pdf page_%03d.svg all` to convert PDFs to SVGs.
- Deploy these scripts via `.dotfiles/bin` so they're on `$PATH`.

Purpose

Record odd ASCII art snippets, symbol meanings, and any character tables you reference when scripting.

Classification (`ls -F`)

Symbol	Meaning
/	Directory
*	Executable
@	Symlink
=	Socket
{ }	Brace expansion delimiters

ASCII Helpers

- Use `printf '\xE2\x94\x82'` style hex escapes for box-drawing characters when needed.
- `cat -v` shows control characters (`^[` = Esc).

Table Template

```
LEFT  >>  command alt seperate  <<  RIGHT
```

Keep any future ASCII art or layout experiments here so the rest of the repo stays clean.

Purpose

Directory of desktop + CLI utilities referenced elsewhere. Use this when reinstalling or recommending tools.

Category	Tool	Notes / Link
Finder replacements	Supercharge, Marta	https://sindresorhus.com/supercharg https://marta.sh
Terminals	Ghostty, iTerm2, Warp	Lightweight vs IDE-style
Navigation	Broot, Zazi	Terminal file browsers
Multiplexers	tmux (see 40-workspace/tmux.md), Zellij (utility-tools.md)	
Container runtimes	OrbStack	https://orbstack.dev
Automation	Tmate, shellinabox	Remote terminal sharing

Category	Tool	Notes / Link
Media	Gifski, ImageOptim, Kdenlive	Encoding + editing
UI helpers	Hidden Bar, MTMR	Menubar + Touch Bar
Filesystems	macFUSE	Mount non-native formats
Terminal apps	ascii-image-converter, ImageMagick, Pillow	For visuals

Keep this list synced with `utility-tools.md` so onboarding steps remain accurate.