Full Command Book

> *Every pnpm, git, SvelteKit CLI, and workflow command used in the project — organized by category with syntax, description, flags, and examples.*

Table of Contents

---

pnpm Commands

Package Management

pnpm install

Install all dependencies from `package.json`.

```
pnpm install
```

Flag	Description
`--frozen-lockfile`	Fail if lockfile is out of date (CI use)
`--prod`	Install only production dependencies
`--dev`	Install only dev dependencies
`--prefer-offline`	Use cached packages when possible
`--no-optional`	Skip optional dependencies

```
# CI install (strict)
pnpm install --frozen-lockfile

# Production-only install
pnpm install --prod

# Offline-first install
pnpm install --prefer-offline
```

pnpm add

Add a new dependency.

```
pnpm add <package>
```

Flag	Description

`-D` / `--save-dev`	Add as devDependency
`-g`	Install globally
`-E` / `--save-exact`	Save exact version (no ^/~)
`--filter <package>`	Add to specific workspace package

```
# Add production dependency
pnpm add gsap

# Add dev dependency
pnpm add -D @sveltejs/adapter-vercel

# Add with exact version
pnpm add -E svelte@5.0.0

# Add multiple packages
pnpm add tailwindcss postcss autoprefixer

# Add global tool
pnpm add -g vercel
```

pnpm remove

Remove a dependency.

```
pnpm remove <package>
```

```
# Remove a package
pnpm remove lodash

# Remove dev dependency
pnpm remove -D old-tool
```

pnpm update

Update dependencies.

```
pnpm update
```

Flag	Description
`--latest`	Update to latest versions (ignore ranges)
`--interactive`	Interactively select updates
`-r`	Update recursively in workspaces

```
# Update all within ranges
pnpm update

# Update to latest versions
pnpm update --latest
```

```
# Update specific package
pnpm update svelte

# Interactive update
pnpm update --interactive
```

pnpm list

List installed packages.

```
pnpm list
```

Flag	Description
`--depth <n>`	Max display depth
`--dev`	List only devDependencies
`--prod`	List only dependencies
`--json`	Output as JSON

```
# List top-level dependencies
pnpm list --depth 0

# List as JSON
pnpm list --json --depth 0

# Check for specific package
pnpm list svelte
```

pnpm why

Show why a package is installed (dependency chain).

```
pnpm why <package>
```

```
pnpm why esbuild
pnpm why postcss
```

pnpm outdated

Check for outdated packages.

```
pnpm outdated
```

```
# Check all outdated
pnpm outdated

# Check specific package
pnpm outdated svelte
```

Script Execution

pnpm run

Execute a script defined in `package.json`.

```
pnpm run <script>
```

```
# Run dev server
pnpm run dev

# Shorthand (pnpm recognizes common scripts)
pnpm dev
pnpm build
pnpm preview
pnpm test
pnpm lint

# Pass arguments through
pnpm dev -- --host
pnpm build -- --verbose
```

pnpm exec

Execute a command from node_modules/.bin.

```
pnpm exec <command>
```

```
# Run a local binary
pnpm exec svelte-kit sync
pnpm exec playwright install

# Shorthand with pnpx
pnpx sv create my-app
```

pnpm dlx

Download and run a package without installing.

```
pnpm dlx <package> [args]
```

```
# Create new SvelteKit project
pnpm dlx sv create my-project

# Run one-off tools
pnpm dlx lighthouse https://mysite.com
pnpm dlx serve build/
```

Workspace Commands

```
# Run script in specific workspace package
pnpm --filter <package-name> <command>
```

```
# Run script in all packages
pnpm -r <command>

# Run build in all packages
pnpm -r build

# Run test in specific package
pnpm --filter @shipforge/ui test
```

Cache Management

```
# View cache directory
pnpm store path

# Prune unused packages from store
pnpm store prune

# Verify store integrity
pnpm store status
```

Git Commands

Repository Setup

git init

Initialize a new Git repository.

```
git init
```

```
# Initialize with default branch name
git init -b main

# Initialize in specific directory
git init my-project
```

git clone

Clone a remote repository.

```
git clone <url>
```

Flag	Description
`--depth <n>`	Shallow clone with n commits
`--branch <name>`	Clone specific branch
`--single-branch`	Clone only one branch

```
# Standard clone
git clone https://github.com/user/repo.git
```

```
# Shallow clone (faster for large repos)
git clone --depth 1 https://github.com/user/repo.git

# Clone specific branch
git clone --branch develop https://github.com/user/repo.git
```

Daily Workflow

git status

Show working tree status.

```
git status
```

Flag	Description
`-s` / `--short`	Short format output
`-b` / `--branch`	Show branch info in short format

```
git status
git status -sb   # Short format with branch
```

git add

Stage changes for commit.

```
git add <pathspec>
```

```
# Stage specific file
git add src/routes/+page.svelte

# Stage specific files
git add src/lib/components/Hero.svelte src/lib/styles/hero.css

# Stage all changes in a directory
git add src/lib/

# Stage all tracked modified files
git add -u

# Stage all changes (new, modified, deleted)
git add -A

# Stage with patch mode (interactive hunks)
git add -p
```

git commit

Record changes to the repository.

```
git commit -m "message"
```

Flag	Description
`-m "message"`	Commit message inline
`-a`	Auto-stage tracked modified files
`--amend`	Modify the last commit
`--no-verify`	Skip pre-commit hooks
`--allow-empty`	Allow commit with no changes

```
# Standard commit
git commit -m "feat: add hero animation component"

# Multi-line commit message
git commit -m "feat: add hero animation

- Implemented GSAP timeline for entrance
- Added ScrollTrigger for parallax effect
- Included reduced motion support"

# Amend last commit (change message or add files)
git add forgotten-file.js
git commit --amend -m "feat: add hero animation component"

# Commit all modified tracked files
git commit -am "fix: correct button alignment"
```

git diff

Show changes between commits, working tree, etc.

```
git diff
```

Flag	Description
`--staged` / `--cached`	Show staged changes
`--stat`	Show diffstat summary
`--name-only`	Show only changed filenames
`--name-status`	Show names and status (A/M/D)

```
# Working directory changes (unstaged)
git diff

# Staged changes
git diff --staged

# Changes between branches
git diff main..feature-branch

# Summary of changes
git diff --stat
```

```
# Specific file
git diff src/routes/+page.svelte
```

git log

Show commit history.

```
git log
```

Flag	Description
`--oneline`	Compact one-line format
`-n <number>`	Limit number of commits
`--graph`	ASCII graph of branch structure
`--all`	Show all branches
`--author="name"`	Filter by author
`--since="date"`	Commits since date
`--stat`	Show file change stats

```
# Last 10 commits, one line each
git log --oneline -10

# Pretty graph view
git log --oneline --graph --all -20

# Commits by specific author this week
git log --author="John" --since="1 week ago" --oneline

# Commits affecting specific file
git log --oneline -- src/routes/+page.svelte

# Detailed log with stats
git log --stat -5
```

Branching

git branch

List, create, or delete branches.

```
git branch
```

Flag	Description
`-a`	List all branches (local + remote)
`-d <branch>`	Delete branch (safe)
`-D <branch>`	Force delete branch

`-m <old> <new>`	Rename branch
`-v`	Show last commit on each branch

```
# List local branches
git branch

# List all branches including remote
git branch -a

# Create new branch
git branch feature/hero-animation

# Delete merged branch
git branch -d feature/old-feature

# Rename current branch
git branch -m new-name
```

git checkout / git switch

Switch branches or restore files.

```
git switch <branch>     # Modern (preferred)
git checkout <branch>   # Classic
```

```
# Switch to existing branch
git switch main
git switch feature/hero-animation

# Create and switch to new branch
git switch -c feature/seo-meta-tags

# Switch to previous branch
git switch -

# Classic equivalents
git checkout main
git checkout -b feature/seo-meta-tags
```

git merge

Merge branches.

```
git merge <branch>
```

Flag	Description
`--no-ff`	Create merge commit even if fast-forward possible
`--squash`	Squash all commits into one
`--abort`	Abort merge in progress

```
# Merge feature into main
git switch main
git merge feature/hero-animation

# Merge with merge commit (no fast-forward)
git merge --no-ff feature/seo-meta-tags

# Squash merge (combine all commits)
git merge --squash feature/small-fix
git commit -m "fix: apply small fix from feature branch"

# Abort conflicted merge
git merge --abort
```

git rebase

Reapply commits on top of another base.

```
git rebase <branch>
```

```
# Rebase feature branch onto latest main
git switch feature/hero-animation
git rebase main

# Continue after resolving conflicts
git rebase --continue

# Abort rebase
git rebase --abort

# Skip conflicted commit
git rebase --skip
```

Remote Operations

git remote

Manage remote repositories.

```
git remote
```

```
# List remotes
git remote -v

# Add remote
git remote add origin https://github.com/user/repo.git

# Change remote URL
git remote set-url origin https://github.com/user/new-repo.git

# Remove remote
git remote remove upstream
```

git fetch

Download objects and refs from remote.

```
git fetch
```

```
# Fetch from origin
git fetch origin

# Fetch all remotes
git fetch --all

# Fetch and prune deleted remote branches
git fetch --prune
```

git pull

Fetch and integrate remote changes.

```
git pull
```

Flag	Description
`--rebase`	Rebase instead of merge
`--no-rebase`	Force merge strategy

```
# Pull with merge (default)
git pull origin main

# Pull with rebase (cleaner history)
git pull --rebase origin main
```

git push

Upload local commits to remote.

```
git push
```

Flag	Description
`-u` / `--set-upstream`	Set upstream tracking
`--force-with-lease`	Safe force push
`--tags`	Push all tags
`--delete`	Delete remote branch

```
# Push to tracked remote branch
git push

# Push and set upstream
git push -u origin feature/hero-animation
```

```
# Safe force push (after rebase)
git push --force-with-lease

# Delete remote branch
git push origin --delete feature/old-branch

# Push all tags
git push --tags
```

Stashing

git stash

Temporarily save uncommitted changes.

```
git stash
```

```
# Stash current changes
git stash

# Stash with descriptive message
git stash push -m "WIP: hero animation refactor"

# Stash including untracked files
git stash -u

# List all stashes
git stash list

# Apply most recent stash (keep in stash list)
git stash apply

# Apply and remove most recent stash
git stash pop

# Apply specific stash
git stash apply stash@{2}

# Drop specific stash
git stash drop stash@{0}

# Clear all stashes
git stash clear

# Show stash contents
git stash show -p stash@{0}
```

Tags

git tag

Create, list, delete tags.

```
git tag
```

```
# List tags
git tag

# Create lightweight tag
git tag v1.0.0

# Create annotated tag (recommended)
git tag -a v1.0.0 -m "Release version 1.0.0"

# Tag specific commit
git tag -a v1.0.0 abc1234 -m "Release version 1.0.0"

# Delete local tag
git tag -d v1.0.0

# Delete remote tag
git push origin --delete v1.0.0

# Push single tag
git push origin v1.0.0

# Push all tags
git push --tags
```

Undoing Changes

```
# Discard unstaged changes to a file
git restore src/routes/+page.svelte

# Unstage a file (keep changes)
git restore --staged src/routes/+page.svelte

# Reset last commit (keep changes staged)
git reset --soft HEAD~1

# Reset last commit (keep changes unstaged)
git reset HEAD~1

# Reset last commit (discard changes) - DESTRUCTIVE
git reset --hard HEAD~1

# Revert a commit (create new commit that undoes it)
git revert <commit-hash>

# Revert without auto-commit
git revert --no-commit <commit-hash>
```

Cherry-Pick

```
# Apply specific commit to current branch
git cherry-pick <commit-hash>

# Cherry-pick without committing
```

```
git cherry-pick --no-commit <commit-hash>

# Cherry-pick a range
git cherry-pick <start-hash>..<end-hash>
```

Inspection

```
# Show specific commit details
git show <commit-hash>

# Show who last modified each line
git blame src/routes/+page.svelte

# Search commit messages
git log --grep="hero animation"

# Search code changes
git log -S "ScrollTrigger" --oneline

# Show branch merge history
git log --merges --oneline -10
```

SvelteKit CLI Commands

Project Creation

```
# Create new SvelteKit project (2026 method)
pnpm dlx sv create my-project
```

Interactive prompts will ask about:

- Template (minimal, demo, library)
- TypeScript (Yes/No)
- Add-ons (ESLint, Prettier, Playwright, Vitest, etc.)

```
# Navigate and install
cd my-project
pnpm install
```

SvelteKit Sync

Synchronize generated types and files.

```
pnpm exec svelte-kit sync
```

This regenerates:

- `.svelte-kit/` directory
- Type declarations for routes
- `$env` module types

```
# Usually runs automatically, but force it with:
pnpm exec svelte-kit sync
```

Development

```
# Start dev server
pnpm dev

# Start on specific port
pnpm dev -- --port 3000

# Expose to network (LAN access)
pnpm dev -- --host

# Start with HTTPS
pnpm dev -- --https

# Open browser automatically
pnpm dev -- --open
```

Building

```
# Production build
pnpm build

# Build with verbose output
pnpm build -- --verbose
```

Preview

```
# Preview production build locally
pnpm preview

# Preview on specific port
pnpm preview -- --port 4173

# Preview with network access
pnpm preview -- --host
```

Checking

```
# Type-check the project
pnpm check

# Watch mode type-checking
pnpm check -- --watch
```

Vite Commands

Vite Dev Server

```
# Start dev server (via SvelteKit)
pnpm dev

# Vite-specific flags (pass through --)
pnpm dev -- --host 0.0.0.0
pnpm dev -- --port 5173
pnpm dev -- --strictPort    # Fail if port in use
pnpm dev -- --open          # Open browser
pnpm dev -- --cors          # Enable CORS
```

Vite Build

```
# Production build (via SvelteKit)
pnpm build

# Analyze bundle
pnpm build -- --mode analyze

# Custom out directory
pnpm build -- --outDir custom-build
```

Vite Preview

```
# Preview build
pnpm preview

# Custom port
pnpm preview -- --port 8080
```

Vite Environment Variables

```
# .env files loaded by Vite
.env                # All modes
.env.local          # All modes, git-ignored
.env.development    # Dev mode only
.env.production     # Production mode only

# Access in code
# Public (exposed to client): VITE_*
# Private (server only): All others via $env/static/private
```

```
// src/lib/config.js
import { PUBLIC_SITE_URL } from '$env/static/public';
import { PRIVATE_API_KEY } from '$env/static/private';

export const SITE_URL = PUBLIC_SITE_URL;
// PRIVATE_API_KEY only available in server-side code
```

Playwright Commands

Installation

```
# Install Playwright browsers
pnpm exec playwright install

# Install specific browser
pnpm exec playwright install chromium

# Install with system dependencies
pnpm exec playwright install --with-deps
```

Running Tests

```
# Run all tests
pnpm exec playwright test

# Run specific test file
pnpm exec playwright test tests/seo.test.js

# Run tests matching pattern
pnpm exec playwright test -g "hero animation"

# Run in specific browser
pnpm exec playwright test --project=chromium
pnpm exec playwright test --project=firefox
pnpm exec playwright test --project=webkit

# Run in headed mode (see browser)
pnpm exec playwright test --headed

# Run in debug mode
pnpm exec playwright test --debug

# Run specific test by line number
pnpm exec playwright test tests/seo.test.js:15
```

Test Results

```
# Generate HTML report
pnpm exec playwright show-report

# Run with specific reporter
pnpm exec playwright test --reporter=list
pnpm exec playwright test --reporter=dot
pnpm exec playwright test --reporter=html

# Run with multiple reporters
pnpm exec playwright test --reporter=list,html
```

Code Generation

```
# Record new test interactively
pnpm exec playwright codegen http://localhost:5173

# Record for specific viewport
pnpm exec playwright codegen --viewport-size=375,812 http://localhost:5173
```

Playwright Configuration

```
# Show config
pnpm exec playwright test --config=playwright.config.js

# List available tests
pnpm exec playwright test --list

# Update snapshots
pnpm exec playwright test --update-snapshots
```

Deployment Commands

Vercel

```
# Install Vercel CLI
pnpm add -g vercel

# Login
vercel login

# Deploy preview
vercel

# Deploy to production
vercel --prod

# Link to existing project
vercel link

# Set environment variables
vercel env add PRIVATE_KEY
vercel env ls
vercel env rm PRIVATE_KEY

# Pull environment variables locally
vercel env pull .env.local

# View deployment logs
vercel logs <deployment-url>

# Rollback to previous deployment
vercel rollback
```

Cloudflare Pages

```
# Install Wrangler CLI
pnpm add -g wrangler

# Login
wrangler login

# Deploy
wrangler pages deploy build/

# Set environment variables
wrangler pages secret put API_KEY

# View deployment
wrangler pages deployment list
```

Netlify

```
# Install Netlify CLI
pnpm add -g netlify-cli

# Login
netlify login

# Deploy preview
netlify deploy

# Deploy to production
netlify deploy --prod

# Set environment variables
netlify env:set API_KEY "value"
netlify env:list
```

Docker

```
# Build Docker image
docker build -t shipforge .

# Run container
docker run -p 3000:3000 shipforge

# Run with environment variables
docker run -p 3000:3000 -e DATABASE_URL="..." shipforge

# Build and run with docker-compose
docker-compose up --build

# Stop containers
docker-compose down
```

Utility Commands

Linting & Formatting

```
# Run ESLint
pnpm lint

# Fix ESLint issues automatically
pnpm lint -- --fix

# Run Prettier check
pnpm format -- --check

# Fix formatting
pnpm format

# Specific files
pnpm exec prettier --write "src/**/*.svelte"
pnpm exec eslint "src/**/*.{js,svelte}" --fix
```

Type Checking

```
# SvelteKit type check
pnpm check

# Watch mode
pnpm check -- --watch

# TypeScript compiler check (if using tsc)
pnpm exec tsc --noEmit
```

File Operations

```
# Create directory structure
mkdir -p src/lib/components/ui
mkdir -p src/lib/actions
mkdir -p src/lib/animations
mkdir -p src/lib/utils
mkdir -p src/lib/stores
mkdir -p static/fonts
mkdir -p static/images

# Copy files
cp src/lib/components/Button.svelte src/lib/components/Button.backup.svelte

# Find files
find src -name "*.svelte" -type f
find src -name "*.test.js" -type f

# Count lines of code
find src -name "*.svelte" -o -name "*.js" | xargs wc -l

# Search in files
```

```
grep -r "ScrollTrigger" src/
grep -rn "TODO" src/ --include="*.svelte"
```

System & Network

```
# Check Node version
node --version

# Check pnpm version
pnpm --version

# Kill process on port
lsof -ti:5173 | xargs kill -9

# Check what's running on a port
lsof -i :5173

# Check disk usage
du -sh node_modules/
du -sh .svelte-kit/
du -sh build/

# Environment info
npx envinfo --system --binaries --browsers --npmPackages
```

Workflow Recipes

New Feature Branch Workflow

```
# 1. Start from latest main
git switch main
git pull --rebase origin main

# 2. Create feature branch
git switch -c feature/hero-animation

# 3. Develop (iterative)
pnpm dev
# ... make changes ...

# 4. Stage and commit
git add src/lib/components/HeroAnimation.svelte
git add src/lib/animations/hero.js
git commit -m "feat: add hero entrance animation with GSAP"

# 5. Push and create PR
git push -u origin feature/hero-animation
gh pr create --title "feat: hero entrance animation" --body "Added GSAP-powered hero animation
with ScrollTrigger"

# 6. After merge, clean up
git switch main
git pull --rebase origin main
git branch -d feature/hero-animation
```

Hotfix Workflow

```
# 1. Branch from main
git switch main
git pull --rebase origin main
git switch -c hotfix/fix-mobile-nav

# 2. Fix the issue
# ... make changes ...

# 3. Commit and push
git add -u
git commit -m "fix: mobile navigation overflow on small screens"
git push -u origin hotfix/fix-mobile-nav

# 4. Create PR with urgency
gh pr create --title "fix: mobile nav overflow" --body "Critical fix for mobile navigation"

# 5. After merge, tag if needed
git switch main
git pull
git tag -a v1.0.1 -m "Hotfix: mobile navigation"
git push --tags
```

Full Build & Deploy

```
# 1. Ensure clean state
git status
pnpm check
pnpm lint

# 2. Run tests
pnpm exec playwright test

# 3. Build
pnpm build

# 4. Preview locally
pnpm preview

# 5. Deploy
vercel --prod

# 6. Tag release
git tag -a v1.0.0 -m "Release v1.0.0"
git push --tags
```

Dependency Update Workflow

```
# 1. Check for outdated packages
pnpm outdated

# 2. Update packages
pnpm update

# 3. Or update to latest (breaking changes possible)
pnpm update --latest

# 4. Check nothing broke
pnpm check
pnpm lint
pnpm build
pnpm exec playwright test

# 5. Commit lockfile changes
git add pnpm-lock.yaml package.json
git commit -m "chore: update dependencies"
```

Environment Setup (New Machine)

```
# 1. Install Node.js (via nvm or fnm)
curl -fsSL https://fnm.vercel.app/install | bash
fnm install --lts
fnm use --lts

# 2. Install pnpm
corepack enable
corepack prepare pnpm@latest --activate

# 3. Clone the project
git clone https://github.com/user/shipforge.git
cd shipforge

# 4. Install dependencies
pnpm install

# 5. Setup environment
cp .env.example .env.local
# Edit .env.local with your values

# 6. Sync SvelteKit types
pnpm exec svelte-kit sync

# 7. Install Playwright browsers
pnpm exec playwright install

# 8. Start developing
pnpm dev
```

Performance Audit Workflow

```bash
# 1. Build production version
pnpm build

# 2. Start preview server
pnpm preview &

# 3. Run Lighthouse
pnpm dlx lighthouse http://localhost:4173 --output html --output-path ./lighthouse-report.html

# 4. Run Lighthouse for specific categories
pnpm dlx lighthouse http://localhost:4173 --only-categories=performance,seo,accessibility

# 5. Bundle analysis
pnpm dlx vite-bundle-visualizer

# 6. Check bundle sizes
du -sh build/
ls -lah build/client/_app/immutable/chunks/
```

Quick Reference Card

Most-Used Commands (Daily)

```bash
pnpm dev                    # Start dev server
pnpm build                  # Production build
pnpm preview                # Preview build
pnpm check                  # Type check
pnpm lint                   # Lint code

git status -sb              # Quick status
git add <files>             # Stage files
git commit -m "message"     # Commit
git push                    # Push to remote
git pull --rebase           # Pull with rebase
git switch -c <branch>      # New branch
git log --oneline -10       # Recent history
```

Commit Message Convention

```
Format: <type>: <description>

Types:
  feat:     New feature
  fix:      Bug fix
  docs:     Documentation only
  style:    Formatting, missing semi colons, etc.
  refactor: Code change that neither fixes a bug nor adds a feature
  perf:     Performance improvement
  test:     Adding tests
  chore:    Maintenance tasks (deps, config, etc.)

Examples:
  feat: add hero animation with GSAP ScrollTrigger
```

```
   fix: correct mobile navigation overflow
   docs: update README with deployment instructions
   refactor: extract SEO component from layout
   perf: lazy load below-fold images
   test: add Playwright tests for SEO meta tags
   chore: update dependencies to latest versions
```

End of Command Book