

GSD

Command Reference

Every command, every flag, every file

Tibsfox

For GSD v1.35.0

Audience: all GSD users — pin this to the wall

Contents

How to Read This Reference2

1 Core Workflow	3
2 Phase Management	5
3 Navigation	6
4 Utilities	6
5 Diagnostics	9
6 Workstream Management	9
7 Configuration Commands	9
8 Brownfield & Analysis	10
9 AI Integration	10
10 Update Commands	11
11 Code Quality	11
12 Fast & Inline	12
13 Backlog, Seeds & Threads	12
14 State Management (CLI)	13
15 Community	13
16 Configuration Reference	14
17 Core Settings	14
18 Workflow Toggles	14
19 Planning, Hooks & Features	15
20 Parallelization	15
21 Git Branching	16
22 Gates, Safety & Review	16
23 Manager Passthrough Flags	16
24 Model Profiles	16
25 Environment Variables	17
26 Global Defaults	17
27 File Manifest	17

How to Read This Reference

Commands appear in their canonical v1.35.0 skill form (`/gsd:command`). The older dash form (`/gsd-command`) remains a valid alias in every supported runtime. Argument and flag syntax follows standard POSIX conventions: angle brackets `<name>` mark required arguments, square brackets `[name]` mark optional ones, and ellipses `...` indicate a repeating list. Flag tables list only flags documented in the upstream `docs/COMMANDS.md` source for v1.35.0.

Runtime invocation prefixes. GSD v1.35.0 supports fourteen host runtimes: Claude Code, Open-Code, Gemini CLI, Kilo, Codex, GitHub Copilot, Cursor, Windsurf, Antigravity, Augment, Trae, Qwen Code, CodeBuddy, and Cline. Most of them use `/gsd-*` at a prompt. Codex uses `$gsd-*`. Skill-style invocation (`/gsd:*`) is available on the runtimes whose skill surfaces expose it. Pick whichever form your runtime accepts; the behavior is identical.

1 Core Workflow

/gsd:new-project Bootstrap a new project with deep context gathering — runs research, produces all planning artifacts, and writes CLAUDE.md.

Syntax: /gsd:new-project [--auto @file.md]

Produces: PROJECT.md, REQUIREMENTS.md, ROADMAP.md, STATE.md, config.json, research/, CLAUDE.md

Ex: /gsd:new-project --auto @prd.md

Flag	Description
--auto @file.md	Auto-extract from a document, skip interactive questions

/gsd:new-workspace Create an isolated multi-repo workspace with its own *.planning/* directory and repo copies (worktrees or clones).

Syntax: /gsd:new-workspace --name <n> [--repos r1,r2] [--path /t] [--strategy worktree|clone] [--branch] [--auto]

Produces: WORKSPACE.md, .planning/, repo copies

Ex: /gsd:new-workspace --name feature-b --repos hr-ui,ZeymoAPI

Flag	Description
--name <name>	Workspace name (required)
--repos r1,r2	Comma-separated repo paths or names
--path /target	Target directory (default: <i>~/gsd-workspaces/<name></i>)
--strategy worktree clone	Copy strategy (default: <i>worktree</i>)
--branch <name>	Branch to check out (default: <i>workspace/<name></i>)
--auto	Skip interactive questions

/gsd:list-workspaces List active GSD workspaces and their status. Scans *~/gsd-workspaces/* for WORKSPACE.md manifests.

Syntax: /gsd:list-workspaces

Ex: /gsd:list-workspaces

/gsd:remove-workspace Remove a workspace and clean up git worktrees. Refuses removal if any repo has uncommitted changes; requires name confirmation.

Syntax: /gsd:remove-workspace <name>

Ex: /gsd:remove-workspace feature-b

/gsd:discuss-phase Capture implementation decisions before planning through a Socratic question loop.

Syntax: /gsd:discuss-phase [N] [--auto] [--batch] [--analyze] [--power]

Produces: {phase}-CONTEXT.md, {phase}-DISCUSSION-LOG.md

Ex: /gsd:discuss-phase 2 --analyze

Flag	Description
--auto	Auto-select recommended defaults for all questions
--batch	Group questions for batch intake instead of one-by-one
--analyze	Add trade-off analysis during discussion
--power	File-based bulk answers from a prepared answers file

/gsd:ui-phase Generate a UI design contract for a frontend phase before planning or execution.

Syntax: /gsd:ui-phase [N]

Produces: {phase}-UI-SPEC.md

Ex: /gsd:ui-phase 2

/gsd:plan-phase Research, plan, and verify a phase. The workhorse of the main workflow.

Syntax: /gsd:plan-phase [N] [--auto] [--research|--skip-research] [--gaps] [--skip-verify] [--prd <f>] [--reviews] [--validate] [--bounce|--skip-bounce]

Produces: {phase}-RESEARCH.md, {phase}-{N}-PLAN.md, {phase}-VALIDATION.md

Ex: /gsd:plan-phase 1 --bounce

Flag	Description
--auto	Skip interactive confirmations
--research	Force re-research even if RESEARCH.md exists
--skip-research	Skip domain research step entirely
--gaps	Gap closure mode (reads VERIFICATION.md, skips research)
--skip-verify	Skip plan-checker verification loop
--prd <file>	Use a PRD file instead of discuss-phase for context
--reviews	Replan incorporating cross-AI review feedback from REVIEWS.md
--validate	Run state validation before planning begins
--bounce	Run external plan bounce validation after planning
--skip-bounce	Skip plan bounce even if enabled in config

/gsd:execute-phase Execute all plans in a phase with wave-based parallelization, or run a single wave.

Syntax: /gsd:execute-phase <N> [--wave N] [--validate] [--cross-ai|--no-cross-ai]

Produces: {phase}-{N}-SUMMARY.md per plan, commits, {phase}-VERIFICATION.md

Ex: /gsd:execute-phase 1 --wave 2

Flag	Description
--wave N	Execute only the Nth wave in the phase
--validate	Run state validation before execution begins
--cross-ai	Delegate execution to an external AI CLI
--no-cross-ai	Force local execution even if cross-AI is enabled

/gsd:verify-work User acceptance testing with auto-diagnosis — runs after a phase has been executed.

Syntax: /gsd:verify-work [N]

Produces: {phase}-UAT.md, fix plans if issues found

Ex: /gsd:verify-work 1

/gsd:next Automatically advance to the next logical workflow step. Reads project state and dispatches to the right command (discuss, plan, execute, verify, or complete-milestone).

Syntax: /gsd:next

Ex: /gsd:next

/gsd:session-report Generate a session report with work summary, outcomes, and estimated resource usage.

Syntax: /gsd:session-report

Produces: .planning/reports/SESSION_REPORT.md

Ex: /gsd:session-report

/gsd:ship Create a pull request from completed phase work with an auto-generated PR body.

Syntax: /gsd:ship [N|version] [--draft]

Produces: GitHub PR, STATE.md updated

Ex: /gsd:ship 4 --draft

Flag	Description
--draft	Create as a draft PR

/gsd:ui-review Retroactive 6-pillar visual audit of an implemented frontend. Runs stand-alone — no active GSD project required.

Syntax: /gsd:ui-review [N]

Produces: {phase}-UI-REVIEW.md, screenshots under .planning/ui-reviews/

Ex: /gsd:ui-review 3

/gsd:audit-uat Cross-phase audit of all outstanding UAT and verification items, producing a categorized report with a human test plan.

Syntax: /gsd:audit-uat

Produces: Categorized audit report

Ex: /gsd:audit-uat

/gsd:audit-milestone Verify a milestone met its definition of done. Runs gap analysis over all executed phases.

Syntax: /gsd:audit-milestone

Produces: Audit report with gap analysis

Ex: /gsd:audit-milestone

/gsd:complete-milestone Archive a milestone and tag the release.

Syntax: /gsd:complete-milestone

Produces: MILESTONES.md entry, git tag

Ex: /gsd:complete-milestone

/gsd:milestone-summary Generate a comprehensive project summary from milestone artifacts (for team onboarding and review). Includes architecture decisions, phase breakdown, requirements coverage, tech debt, and a getting-started guide.

Syntax: /gsd:milestone-summary [version]

Produces: .planning/reports/MILESTONE_SUMMARY-v{version}.md

Ex: /gsd:milestone-summary v1.0

/gsd:new-milestone Start the next version cycle. Updates PROJECT.md and creates a new REQUIREMENTS.md and ROADMAP.md.

Syntax: /gsd:new-milestone [name] [--reset-phase-numbers]

Produces: Updated PROJECT.md, new REQUIREMENTS.md, new ROADMAP.md

Ex: /gsd:new-milestone "v2.0 Mobile"

Flag	Description
--reset-phase-numbers	Restart the new milestone at Phase 1 and archive old phase dirs

2 Phase Management

/gsd:add-phase Append a new phase to the roadmap. Interactive — describe the phase to the prompter.

Syntax: /gsd:add-phase

Produces: Updated ROADMAP.md

Ex: /gsd:add-phase

/gsd:insert-phase Insert urgent work between existing phases using decimal numbering (3 becomes 3.1, etc.).

Syntax: /gsd:insert-phase [N]

Produces: Updated ROADMAP.md with decimal phase

Ex: /gsd:insert-phase 3

/gsd:remove-phase Remove a future phase and renumber subsequent phases.

Syntax: /gsd:remove-phase [N]

Produces: Updated ROADMAP.md, renumbered phase dirs

Ex: /gsd:remove-phase 7

/gsd:list-phase-assumptions Preview the assumed approach for a phase before committing to plan-phase work.

Syntax: /gsd:list-phase-assumptions [N]

Ex: /gsd:list-phase-assumptions 2

/gsd:analyze-dependencies Analyze phase dependencies and suggest Depends on entries for ROADMAP.md. Run this before **/gsd:manager** when phases have empty Depends on fields.

Syntax: /gsd:analyze-dependencies

Produces: Dependency suggestion table; optionally updates ROADMAP.md

Ex: /gsd:analyze-dependencies

/gsd:plan-milestone-gaps Create phases to close gaps surfaced by a milestone audit.

Syntax: /gsd:plan-milestone-gaps

Produces: New phases in ROADMAP.md

Ex: /gsd:plan-milestone-gaps

/gsd:research-phase Deep ecosystem research only — standalone entry point. Usually run **/gsd:plan-phase** instead, which invokes research as its first step.

Syntax: /gsd:research-phase [N]

Produces: {phase}-RESEARCH.md

Ex: /gsd:research-phase 4

/gsd:validate-phase Retroactively audit and fill Nyquist validation gaps for a phase.

Syntax: /gsd:validate-phase [N]

Produces: Updated {phase}-VALIDATION.md

Ex: /gsd:validate-phase 2

3 Navigation

/gsd:progress Show project status and next recommended steps.

Syntax: /gsd:progress

Ex: /gsd:progress

/gsd:resume-work Restore full session context after a context reset or when starting a new session.

Syntax: /gsd:resume-work

Ex: /gsd:resume-work

/gsd:pause-work Save a context handoff when stopping mid-phase. Creates *continue-here.md*.

Syntax: /gsd:pause-work

Produces: continue-here.md

Ex: /gsd:pause-work

/gsd:manager Interactive command center for managing multiple phases from one terminal. Dispatches discuss inline; dispatches plan and execute as background agents.

Syntax: /gsd:manager

Ex: /gsd:manager

The manager respects the `manager.flags.discuss`, `manager.flags.plan`, and `manager.flags.execute` passthrough flags in `.planning/config.json` — see the Configuration Reference chapter.

/gsd:help Show all commands and a quick usage guide.

Syntax: /gsd:help

Ex: /gsd:help

4 Utilities

/gsd:explore Socratic ideation session — guides an idea through probing questions, optionally spawns research, then routes output to the right GSD artifact (notes, todos, seeds, research questions, requirements, or a new phase).

Syntax: /gsd:explore [topic]

Produces: Routed output (varies by destination)

Ex: /gsd:explore authentication strategy

/gsd:undo Safe git revert for GSD phase or plan commits. Uses the phase manifest for dependency checks and always surfaces a confirmation gate before reverting.

Syntax: /gsd:undo (--last N | --phase NN | --plan NN-MM)

Produces: Revert commits

Ex: /gsd:undo --phase 03

Flag	Description
--last N	Show recent GSD commits for interactive selection
--phase NN	Revert all commits for a phase
--plan NN-MM	Revert all commits for a specific plan

/gsd:import Ingest an external plan file into the GSD planning system. Detects conflicts against PROJECT.md decisions before writing.

Syntax: /gsd:import --from <filepath>

Produces: New PLAN.md validated by gsd-plan-checker

Ex: /gsd:import --from /tmp/team-plan.md

Flag	Description
--from <filepath>	Path to the external plan file (required)

/gsd:from-gsd2 Reverse migration from GSD-2 format (.gsd/) back to v1 .planning/ format. Flattens Milestone/Slice hierarchy to sequential phase numbers.

Syntax: /gsd:from-gsd2 [--dry-run] [--force] [--path <dir>]

Produces: PROJECT.md, REQUIREMENTS.md, ROADMAP.md, STATE.md, sequential phase dirs

Ex: /gsd:from-gsd2 --dry-run

Flag	Description
--dry-run	Preview migration without writing
--force	Overwrite existing .planning/ directory
--path <dir>	Specify the GSD-2 root (default: current dir)

/gsd:quick Execute an ad-hoc task with GSD guarantees — composable flags turn quick-tasks into full phases on demand.

Syntax: /gsd:quick [--full] [--discuss] [--research]

Produces: .planning/quick/NNN-slug/ with PLAN.md and SUMMARY.md

Ex: /gsd:quick --discuss --research --full

Flag	Description
--full	Enable plan checking (2 iterations) and post-execution verification
--discuss	Lightweight pre-planning discussion
--research	Spawn a focused researcher before planning

/gsd:autonomous Run all remaining phases autonomously. Respects workflow.skip_discuss to bypass the discuss step.

Syntax: /gsd:autonomous [--from N] [--to N] [--interactive]

Produces: Phase artifacts and commits for every executed phase

Ex: /gsd:autonomous --from 3 --to 5

Flag	Description
--from N	Start from a specific phase number
--to N	Stop after completing a specific phase number
--interactive	Lean context with periodic user input checkpoints

/gsd:do Route freeform text to the right GSD command. After invocation, describe what you want.

Syntax: /gsd:do

Ex: /gsd:do

/gsd:note Zero-friction idea capture. Append, list, or promote notes to todos.

Syntax: /gsd:note [text | list | promote N] [--global]

Produces: Project or global notes file

Ex: /gsd:note "Consider caching for API responses"

Flag	Description
--global	Use global scope for the note operation

/gsd:debug Systematic debugging with persistent cross-session state. Subcommands manage active debug sessions.

Syntax: /gsd:debug [--diagnose] <description> | list | status <slug> | continue <slug>

Produces: Debug session artifacts with Evidence, Eliminated, Resolution, TDD checkpoint

Ex: /gsd:debug "Login button not responding on mobile Safari"

Flag	Description
--diagnose	Diagnosis-only mode — investigate without attempting fixes

/gsd:add-todo Capture an idea or task for later.

Syntax: /gsd:add-todo [description]

Produces: todos/entry

Ex: /gsd:add-todo "Consider adding dark mode support"

/gsd:check-todos List pending todos and select one to work on.

Syntax: /gsd:check-todos

Ex: /gsd:check-todos

/gsd:add-tests Generate tests for a completed phase.

Syntax: /gsd:add-tests [N]

Produces: Test files committed atomically

Ex: /gsd:add-tests 2

/gsd:stats Display project statistics dashboard.

Syntax: /gsd:stats

Ex: /gsd:stats

/gsd:profile-user Generate a developer behavioral profile from Claude Code session analysis across 8 dimensions. Produces artifacts that personalize subsequent GSD responses.

Syntax: /gsd:profile-user [--questionnaire] [--refresh]

Produces: USER-PROFILE.md, /gsd-dev-preferences command, CLAUDE.md profile section

Ex: /gsd:profile-user --questionnaire

Flag	Description
--questionnaire	Use interactive questionnaire instead of session analysis
--refresh	Re-analyze sessions and regenerate the profile

/gsd:health Validate *.planning/* directory integrity. With *--repair*, auto-fix recoverable issues.

Syntax: /gsd:health [--repair]

Produces: Integrity report, optional repair actions

Ex: /gsd:health --repair

Flag	Description
--repair	Auto-fix recoverable integrity issues

/gsd:cleanup Archive accumulated phase directories from completed milestones.

Syntax: /gsd:cleanup

Produces: Archived phase dirs

Ex: /gsd:cleanup

5 Diagnostics

/gsd:forensics Post-mortem investigation of failed or stuck GSD workflows. Checks git history, artifact integrity, STATE.md anomalies, and uncommitted work; covers at least 4 anomaly types.

Syntax: /gsd:forensics [description]

Produces: .planning/forensics/report-`{timestamp}`.md

Ex: /gsd:forensics "Phase 3 execution stalled"

/gsd:extract-learnings Extract reusable patterns, anti-patterns, and architectural decisions from completed phase work.

Syntax: /gsd:extract-learnings <N> [--all] [--format markdown|json]

Produces: .planning/learnings/`{phase}`-LEARNINGS.md

Ex: /gsd:extract-learnings 3

Flag	Description
--all	Extract learnings from all completed phases
--format	Output format: markdown (default) or json

6 Workstream Management

/gsd:workstreams Manage parallel workstreams for concurrent work on different milestone areas. Subcommands select the action.

Syntax: /gsd:workstreams [list | create <n> | status <n> | switch <n> | progress | complete <n> | resume <n>]

Produces: Workstream directories under .planning/, per-workstream state tracking

Ex: /gsd:workstreams create backend-api

Subcommand	Description
list	List all workstreams with status (default if no subcommand)
create <name>	Create a new workstream
status <name>	Detailed status for one workstream
switch <name>	Set active workstream
progress	Progress summary across all workstreams
complete <name>	Archive a completed workstream
resume <name>	Resume work in a workstream

7 Configuration Commands

/gsd:settings Interactive configuration of workflow toggles and model profile.

Syntax: /gsd:settings

Produces: Updated .planning/config.json

Ex: /gsd:settings

/gsd:set-profile Quick switch of the model profile.

Syntax: /gsd:set-profile <profile>

Produces: Updated .planning/config.json

Ex: /gsd:set-profile budget

Flag	Description
quality	Opus for all decision-making, Sonnet for verification
balanced	Opus for planning, Sonnet for everything else (default)
budget	Sonnet for code-writing, Haiku for research and verification
inherit	All agents use the current session model

8 Brownfield & Analysis

/gsd:map-codebase Analyze an existing codebase with four parallel mapper agents. Use before **/gsd:new-project** on a brownfield repo.

Syntax: /gsd:map-codebase [area]

Produces: .planning/codebase/ analysis files

Ex: /gsd:map-codebase auth

/gsd:scan Rapid single-focus codebase assessment — lightweight alternative to map-codebase (one agent instead of four).

Syntax: /gsd:scan [--focus tech|arch|quality|concerns|tech+arch]

Produces: Targeted document(s) in .planning/codebase/

Ex: /gsd:scan --focus quality

Flag	Description
--focus <area>	Focus area: tech, arch, quality, concerns, tech+arch (default)

/gsd:intel Query, inspect, or refresh queryable codebase intelligence files. Requires intel.enabled: true in config.json.

Syntax: /gsd:intel [query <term> | status | diff | refresh]

Produces: .planning/intel/ JSON files (stack, api-map, dependency-graph, file-roles, arch-decisions)

Ex: /gsd:intel query authentication

/gsd:onboard Brownfield onboarding workflow that pairs map-codebase output with a new GSD project scaffold.

Syntax: /gsd:onboard

Produces: PROJECT.md, REQUIREMENTS.md, ROADMAP.md tailored to the mapped codebase

Ex: /gsd:onboard

/gsd:migrate Staged migration planner for legacy code — produces migration phases in the roadmap.

Syntax: /gsd:migrate

Produces: Migration phases in ROADMAP.md

Ex: /gsd:migrate

9 AI Integration

/gsd:ai-integration-phase AI framework selection wizard for integrating AI/LLM capabilities into a phase. Spawns 3 parallel specialist agents (domain researcher, framework selector, ai researcher, eval planner).

Syntax: /gsd:ai-integration-phase [N]

Produces: {phase}-AI-SPEC.md with framework recommendation, implementation guidance, evaluation strategy

Ex: /gsd:ai-integration-phase 3

/gsd:eval-review Retroactive audit of an implemented AI phase's evaluation coverage. Scores each eval dimension as COVERED/PARTIAL/MISSING.

Syntax: /gsd:eval-review [N]

Produces: {phase}-EVAL-REVIEW.md with findings, gaps, and remediation guidance

Ex: /gsd:eval-review 3

10 Update Commands

/gsd:update Update GSD with a changelog preview before applying.

Syntax: /gsd:update

Ex: /gsd:update

/gsd:reapply-patches Restore local modifications after a GSD update. Merges back any local changes that the updater replaced.

Syntax: /gsd:reapply-patches

Ex: /gsd:reapply-patches

11 Code Quality

/gsd:code-review Review source files changed during a phase for bugs, security vulnerabilities, and code quality problems. Three depth levels.

Syntax: /gsd:code-review <N> [--depth=quick|standard|deep] [--files f1,f2,...]

Produces: {phase}-REVIEW.md with severity-classified findings

Ex: /gsd:code-review 2 --depth=deep

Flag	Description
--depth=quick	Pattern-matching only (~2 min)
--depth=standard	Per-file analysis with language-specific checks (default, 5–15 min)
--depth=deep	Cross-file analysis including import graphs and call chains (15–30 min)
--files f1,f2	Explicit file list; skips SUMMARY and git scoping

/gsd:code-review-fix Auto-fix issues found by code-review. Reads REVIEW.md, spawns a fixer agent, commits each fix atomically.

Syntax: /gsd:code-review-fix <N> [--all] [--auto]

Produces: {phase}-REVIEW-FIX.md

Ex: /gsd:code-review-fix 3 --auto

Flag	Description
--all	Include Info findings (default: Critical + Warning only)
--auto	Fix and re-review iteration loop, capped at 3 iterations

/gsd:audit-fix Autonomous audit-to-fix pipeline. Runs an audit, classifies findings, fixes auto-fixable issues with test verification, and commits each fix atomically.

Syntax: /gsd:audit-fix [--source <audit>] [--severity high|medium|all] [--max N] [--dry-run]

Produces: Fix commits with test verification; classification report

Ex: /gsd:audit-fix --severity high

Flag	Description
--source <audit>	Which audit to run (default: audit-uat)
--severity	Minimum severity to process: high, medium, all (default: medium)
--max N	Maximum findings to fix (default: 5)
--dry-run	Classify findings without fixing

/gsd:secure-phase Retroactively verify threat mitigations for a completed phase. Three operating modes: audit existing SECURITY.md, generate from PLAN.md threat model, or exit with guidance if phase not executed.

Syntax: /gsd:secure-phase [N]

Produces: {phase}-SECURITY.md with threat verification results

Ex: /gsd:secure-phase 5

/gsd:docs-update Generate or update project documentation verified against the codebase. Each doc writer explores the codebase directly — no hallucinated paths.

Syntax: /gsd:docs-update [--force] [--verify-only]

Produces: Up to 9 doc files: README, architecture, API, getting-started, development, testing, configuration, deployment, contributing

Ex: /gsd:docs-update --verify-only

Flag	Description
--force	Skip preservation prompts, regenerate all docs
--verify-only	Check existing docs for accuracy, no generation

12 Fast & Inline

/gsd:fast Execute a trivial task inline — no subagents, no planning overhead. For typo fixes, config changes, small refactors, forgotten commits.

Syntax: /gsd:fast [task description]

Produces: Direct edit and commit

Ex: /gsd:fast "fix typo in README"

fast is not a replacement for **/gsd:quick**. Use **/gsd:quick** for anything needing research, multi-step planning, or verification.

/gsd:review Cross-AI peer review of phase plans from external AI CLIs. Produces REVIEWS.md consumable by **/gsd:plan-phase --reviews**.

Syntax: /gsd:review --phase N [--gemini] [--claude] [--codex] [--coderabbit] [--opencode] [--qwen] [--cursor] [--all]

Produces: {phase}-REVIEWS.md

Ex: /gsd:review --phase 3 --all

Flag	Description
--phase N	Phase number to review (required)
--gemini	Include Gemini CLI review
--claude	Include Claude CLI review (separate session)
--codex	Include Codex CLI review
--coderabbit	Include CodeRabbit review
--opencode	Include OpenCode review (via GitHub Copilot)
--qwen	Include Qwen Code review
--cursor	Include Cursor agent review
--all	Include all available CLIs

/gsd:pr-branch Create a clean PR branch by filtering out *.planning/* commits so reviewers see only code changes.

Syntax: /gsd:pr-branch [target-branch]

Produces: Clean git branch

Ex: /gsd:pr-branch develop

13 Backlog, Seeds & Threads

/gsd:add-backlog Add an idea to the backlog parking lot using 999.x numbering. Creates the phase directory immediately so discuss-phase and plan-phase work on it.

Syntax: /gsd:add-backlog <description>

Produces: 999.x phase directory

Ex: /gsd:add-backlog "GraphQL API layer"

/gsd:review-backlog Review and promote backlog items to the active milestone. Per-item actions: Promote, Keep, Remove.

Syntax: /gsd:review-backlog

Produces: Updated ROADMAP.md if any backlog promoted

Ex: /gsd:review-backlog

/gsd:plant-seed Capture a forward-looking idea with trigger conditions — surfaces automatically at the right milestone. Solves context rot for ideas that don't belong to the current phase.

Syntax: /gsd:plant-seed [idea summary]

Produces: .planning/seeds/SEED-NNN-slug.md

Ex: /gsd:plant-seed "Real-time collab when WebSocket infra is in"

/gsd:thread Manage persistent context threads for cross-session work. Lighter weight than pause-work.

Syntax: /gsd:thread [name | description]

Produces: .planning/threads/{name}.md

Ex: /gsd:thread fix-deploy-key-auth

14 State Management (CLI)

State-management commands are invoked through the *gsd-tools.cjs* CLI rather than the workflow surface. They are still part of the v1.35.0 command inventory.

/gsd:tools state validate Detect drift between STATE.md and the actual filesystem.

Syntax: node gsd-tools.cjs state validate

Produces: Validation report showing any drift

Ex: node gsd-tools.cjs state validate

/gsd:tools state sync Reconstruct STATE.md from actual project state on disk.

Syntax: node gsd-tools.cjs state sync [--verify]

Produces: Updated STATE.md

Ex: node gsd-tools.cjs state sync --verify

Flag	Description
--verify	Dry-run mode — show proposed changes without writing

/gsd:tools state planned-phase Record state transition after plan-phase completes (Planned / Ready to execute).

Syntax: node gsd-tools.cjs state planned-phase --phase N --plans N

Produces: Updated STATE.md with post-planning state

Ex: node gsd-tools.cjs state planned-phase --phase 3 --plans 2

15 Community

/gsd:join-discord Open the Discord community invite.

Syntax: /gsd:join-discord

Ex: /gsd:join-discord

Community Hooks

Optional git and session hooks gated behind `hooks.community: true` in `.planning/config.json`. All are no-ops unless explicitly enabled.

Hook	Purpose
<code>gsd-validate-commit.sh</code>	Enforce Conventional Commits format on git commit messages
<code>gsd-session-state.sh</code>	Track session state transitions
<code>gsd-phase-boundary.sh</code>	Enforce phase boundary checks

Enable with:

```
{ "hooks": { "community": true } }
```

16 Configuration Reference

GSD stores project settings in `.planning/config.json`. The file is created by `/gsd:new-project` and can be updated interactively via `/gsd:settings` or per-key via `node gsd-tools.cjs config-set <key> <value>`.

17 Core Settings

Setting	Type	Default
<code>mode</code>	<code>interactive yolo</code>	<code>interactive</code>
<code>granularity</code>	<code>coarse standard fine</code>	<code>standard</code>
<code>model_profile</code>	<code>quality/balanced/budget/inherit</code>	<code>balanced</code>
<code>project_code</code>	string prefix for phase dirs	<code>(none)</code>
<code>response_language</code>	language code (pt, ko, ja)	<code>(none)</code>
<code>context_profile</code>	<code>dev research review</code>	<code>(none)</code>
<code>claude_md_path</code>	custom CLAUDE.md output path	<code>(none)</code>
<code>security_enforcement</code>	boolean	<code>true</code>
<code>security_asvs_level</code>	<code>1 2 3</code>	<code>1</code>
<code>security_block_on</code>	<code>high medium low</code>	<code>high</code>

18 Workflow Toggles

All workflow toggles follow the **absent = enabled** pattern. Missing keys default to `true` unless otherwise noted.

Setting	Type	Default
<code>workflow.research</code>	boolean	<code>true</code>
<code>workflow.plan_check</code>	boolean	<code>true</code>
<code>workflow.verifier</code>	boolean	<code>true</code>
<code>workflow.auto_advance</code>	boolean	<code>false</code>
<code>workflow.nyquist_validation</code>	boolean	<code>true</code>
<code>workflow.ui_phase</code>	boolean	<code>true</code>
<code>workflow.ui_safety_gate</code>	boolean	<code>true</code>
<code>workflow.node_repair</code>	boolean	<code>true</code>
<code>workflow.node_repair_budget</code>	number	<code>2</code>
<code>workflow.research_before_questions</code>	boolean	<code>false</code>
<code>workflow.discuss_mode</code>	<code>discuss assumptions</code>	<code>discuss</code>

Setting	Type	Default
workflow.skip_discuss	boolean	false
workflow.text_mode	boolean	false
workflow.use_worktrees	boolean	true
workflow.code_review	boolean	true
workflow.code_review_depth	quick/standard/deep	standard
workflow.plan_bounce	boolean	false
workflow.plan_bounce_script	string path	(none)
workflow.plan_bounce_passes	number	2
workflow.code_review_command	shell command	(none)
workflow.tdd_mode	boolean	false
workflow.cross_ai_execution	boolean	false
workflow.cross_ai_command	shell command	(none)
workflow.cross_ai_timeout	seconds	300

19 Planning, Hooks & Features

Setting	Type	Default
planning.commit_docs	boolean	true
planning.search_gitignored	boolean	false
hooks.context_warnings	boolean	true
hooks.workflow_guard	boolean	false
hooks.community	boolean	false
features.thinking_partner	boolean	false
features.global_learnings	boolean	false
intel.enabled	boolean	false
learnings.max_inject	number	10
agent_skills	object	{}

If `.planning/` is in `.gitignore`, `planning.commit_docs` is automatically forced to `false` regardless of `config.json`. This prevents git errors.

20 Parallelization

Setting	Type	Default
parallelization.enabled	boolean	true
parallelization.plan_level	boolean	true
parallelization.task_level	boolean	false
parallelization.skip_checkpoints	boolean	true
parallelization.max_concurrent_agents	number	3
parallelization.min_plans_for_parallel	number	2

When parallelization is enabled, executor agents commit with `--no-verify` to avoid build-lock contention (e.g., cargo lock fights in Rust projects). The orchestrator validates hooks once after each wave completes. Set `parallelization.enabled: false` if you need hooks to run per-commit.

21 Git Branching

Setting	Type	Default
git.branching_strategy	none phase milestone	none
git.phase_branch_template	string template	gsd/phase- <code>{phase}</code> - <code>{slug}</code>
git.milestone_branch_template	string template	gsd/ <code>{milestone}</code> - <code>{slug}</code>
git.quick_branch_template	string template null	null

Template variables: `{phase}` (zero-padded), `{slug}` (lowercase, hyphenated), `{milestone}` (e.g., v1.0), `{num}` / `{quick}` (quick task ID).

22 Gates, Safety & Review

Setting	Type	Default
gates.confirm_project	boolean	true
gates.confirm_phases	boolean	true
gates.confirm_roadmap	boolean	true
gates.confirm_breakdown	boolean	true
gates.confirm_plan	boolean	true
gates.execute_next_plan	boolean	true
gates.issues_review	boolean	true
gates.confirm_transition	boolean	true
safety.always_confirm_destructive	boolean	true
safety.always_confirm_external_services	boolean	true
review.models.gemini	string	(CLI default)
review.models.claude	string	(CLI default)
review.models.codex	string	(CLI default)
review.models.opencode	string	(CLI default)
review.models.qwen	string	(CLI default)
review.models.cursor	string	(CLI default)

23 Manager Passthrough Flags

Setting	Type	Default
manager.flags.discuss	string	(none)
manager.flags.plan	string	(none)
manager.flags.execute	string	(none)

Flags are appended to each dispatched command. Invalid flag tokens are sanitized and logged as warnings; only recognized GSD flags are passed through.

24 Model Profiles

Agent	quality	balanced	budget	inherit
gsd-planner	Opus	Opus	Sonnet	inherit
gsd-roadmapper	Opus	Sonnet	Sonnet	inherit
gsd-executor	Opus	Sonnet	Sonnet	inherit
gsd-phase-researcher	Opus	Sonnet	Haiku	inherit
gsd-project-researcher	Opus	Sonnet	Haiku	inherit
gsd-research-synthesizer	Sonnet	Sonnet	Haiku	inherit
gsd-debugger	Opus	Sonnet	Sonnet	inherit
gsd-codebase-mapper	Sonnet	Haiku	Haiku	inherit
gsd-verifier	Sonnet	Sonnet	Haiku	inherit
gsd-plan-checker	Sonnet	Sonnet	Haiku	inherit
gsd-integration-checker	Sonnet	Sonnet	Haiku	inherit
gsd-nyquist-auditor	Sonnet	Sonnet	Haiku	inherit

Override specific agents via `model_overrides`. Valid values: `opus`, `sonnet`, `haiku`, `inherit`, or any fully-qualified model ID (`openai/o3`, `google/gemini-2.5-pro`, etc.). Non-Claude runtimes ship with `resolve_model_ids`: `"omit"` so each agent inherits the runtime's default model unless overridden.

25 Environment Variables

Variable	Purpose
<code>CLAUDE_CONFIG_DIR</code>	Override default config directory (<code>~/.claude/</code>)
<code>GEMINI_API_KEY</code>	Detected by context monitor to switch hook event name
<code>WSL_DISTRO_NAME</code>	Detected by installer for WSL path handling
<code>GSD_SKIP_SCHEMA_CHECK</code>	Skip schema drift detection during <code>execute-phase</code>
<code>GSD_PROJECT</code>	Override project root for multi-project workspace support

26 Global Defaults

Save settings as global defaults for future projects at `~/.gsd/defaults.json`. When `/gsd:new-project` creates a new `config.json`, it reads global defaults and merges them as the starting configuration. Per-project settings always override globals.

27 File Manifest

GSD artifacts live under `.planning/` at the project root (or at `$GSD_PROJECT/.planning/` if overridden). The canonical layout for a typical v1.35.0 project:

```
.planning/
  PROJECT.md           # Project vision, always loaded
  REQUIREMENTS.md     # Scoped v1/v2 requirements with phase traceability
  ROADMAP.md          # Phase list, status, dependencies
  STATE.md            # Decisions, blockers, position across sessions
  config.json         # Workflow, model, git, gate, safety configuration
  MILESTONES.md       # Archive of completed milestones
  CLAUDE.md           # Generated; written at project root by default
  continue-here.md    # Written by /gsd:pause-work
  research/           # Project-level ecosystem research
  codebase/           # /gsd:map-codebase and /gsd:scan output
  intel/              # /gsd:intel JSON files (stack, api-map, etc.)
  seeds/
    SEED-NNN-slug.md  # /gsd:plant-seed artifacts
  threads/
```

```

{thread-name}.md      # /gsd:thread cross-session knowledge stores
todos/                # Captured ideas from /gsd:add-todo and /gsd:note
quick/
  NNN-slug/           # /gsd:quick ad-hoc task directory
  PLAN.md
  SUMMARY.md
forensics/
  report-{timestamp}.md # /gsd:forensics output
learnings/
  {phase}-LEARNINGS.md # /gsd:extract-learnings output
reports/
  SESSION_REPORT.md    # /gsd:session-report
  MILESTONE_SUMMARY-v{v}.md # /gsd:milestone-summary
ui-reviews/           # Screenshots from /gsd:ui-review
phases/
  NN-slug/            # One directory per phase
  CONTEXT.md          # From /gsd:discuss-phase
  DISCUSSION-LOG.md   # Audit trail for discuss-phase
  RESEARCH.md         # From /gsd:plan-phase (or /gsd:research-phase)
  PLAN.md             # Aggregate phase plan
  NN-PLAN.md          # Per-wave plans (NN = wave number)
  VALIDATION.md       # Nyquist validation data
  NN-SUMMARY.md       # Per-plan execution summary
  VERIFICATION.md     # From /gsd:verify-work and executor wrap-up
  UAT.md              # User acceptance test results
  UI-SPEC.md          # From /gsd:ui-phase
  UI-REVIEW.md        # From /gsd:ui-review
  REVIEW.md           # From /gsd:code-review
  REVIEW-FIX.md       # From /gsd:code-review-fix
  REVIEWS.md          # From /gsd:review (cross-AI)
  SECURITY.md         # From /gsd:secure-phase
  AI-SPEC.md          # From /gsd:ai-integration-phase
  EVAL-REVIEW.md      # From /gsd:eval-review

```

Workspace Layout

Workspaces created by `/gsd:new-workspace` live under `~/gsd-workspaces/<name>/` by default, with their own `.planning/` directory, a `WORKSPACE.md` manifest, and one or more repo copies (as worktrees or clones).

Global Layout

```

~/gsd/
  defaults.json      # Global defaults merged into new projects
~/claude/           # Runtime config dir (overridable via CLAUDE_CONFIG_DIR)

```

End of reference. GSD v1.35.0. This document is CC BY-SA 4.0 — The Tibsfox Team.