PDD
PROMPT-DRIVEN DEVELOPMENT

PARTICIPANT FIELD GUIDE / 2026

Build with PDD at the Hackathon

Go from a well-scoped idea to a tested pull request with the PDD CLI or GitHub App.

DEFINE GENERATE VERIFY SHIP

Made for
Hackathon participants

Keep nearby
Setup, commands, labels, review, and day-of cheat sheet

PDD / PARTICIPANT FIELD GUIDE

Start Here

PDD is a prompt-native development system. You preserve the intent, constraints, examples, and tests that define your software, then generate conventional code as a reviewable artifact. This field guide is designed for a hackathon participant who needs to start building now and still finish with a credible, tested pull request.

Use the first three sections to get running. Return to the issue and review checklists while you build. Print or detach the final cheat sheet for the event floor.

> **The operating rule:** prompts, acceptance criteria, and tests are the durable source. Generated code is an artifact. When behavior changes, update the source and regenerate; do not let an important fix exist only as a manual code patch.

Contents

What You Should Be Able To Do

After a short setup, you should be able to:

- Turn a GitHub issue into a working branch or pull request.
- Use the hosted GitHub App without installing a local toolchain.
- Run feature, bug, test, sync, and review workflows from the CLI.
- Resume, steer, stop, or clean-restart a workflow safely.
- Review generated output using tests and observable acceptance criteria.

PDD / PARTICIPANT FIELD GUIDE

Choose Your Route

Choose one of these two participant routes. Both use GitHub issues as the durable task record and produce reviewable pull requests.

Choose	Best when	You need	First action
GitHub App	You want the fastest hosted experience and shared issue-based progress	GitHub access, an installed App, and available PDD Cloud credits	Apply `pdd-issue` to a well-written issue
CLI	You want local control and terminal visibility	PDD, GitHub CLI, and one supported agentic CLI	Run `pdd change ISSUE_URL`

Sixty-Second Start

GitHub App

1. Install the PDD GitHub App on the hackathon repository.
2. Create an issue with a clear goal and acceptance criteria.
3. Apply `pdd-issue`.
4. Follow the live-progress link and reply to questions on the issue.
5. Review the linked PR and require green tests.

CLI

```
uv tool install pdd-cli
pdd setup
pdd auth login
gh auth login

pdd change https://github.com/OWNER/REPO/issues/123
```

PDD / PARTICIPANT FIELD GUIDE

How PDD Works

PDD moves the source of truth up one level. Instead of preserving every manual implementation edit, it preserves the specification and the constraints that make regeneration reliable.

The Development Loop

1. **Define:** state the intended behavior in an issue or module prompt.
2. **Constrain:** add interfaces, MUST/MUST NOT rules, examples, and tests.
3. **Generate:** let PDD create or update the conventional code artifact.

4. **Verify:** run executable tests, negative cases, and the normal project checks.
5. **Review:** inspect the diff and confirm that prompt, code, tests, and docs agree.
6. **Back-propagate:** when implementation work reveals new intent, record it in the prompt or acceptance criteria before completion.

The Three Capitals

Capital	What it contains	Why it matters
Prompt capital	Intent, responsibilities, interfaces, constraints, and vocabulary	Directs what should be generated and why
Test capital	Behavioral, negative, integration, and regression tests	Creates the mold walls that future generations cannot cross
Grounding capital	Successful examples and carefully selected context	Teaches the implementation patterns that fit the project

Verification is the spine of the system. A plausible diff is not a completed change. The useful unit is **verified behavioral change per unit cost**, not lines of generated code.

When PDD Fits Best

PDD is a strong fit for work with clear behavioral outcomes, repeatable tests, and modules that will evolve. It is less useful for disposable experiments, one-line mechanical changes, or tasks whose success cannot be observed or verified. During a hackathon, use PDD for the code you expect to demonstrate, review, and continue after the event.

PDD / PARTICIPANT FIELD GUIDE

Before You Build

Spend ten minutes establishing a clean starting point. This prevents access, credential, and baseline failures from consuming the time you meant to spend on the product.

Participant Preflight

1. Confirm you can clone, branch, push, and open a PR in the hackathon repository.
2. Identify the project's install, run, and test commands.
3. Run the baseline test command before making a change.
4. Confirm required environment variables are documented in `.env.example` or the event's approved secret manager; never commit real credentials.
5. Write one issue with one demonstrable outcome and observable acceptance criteria.
6. Choose either the GitHub App or CLI route.

Route-Specific Check

Using the GitHub App

- Confirm the PDD App is installed on your repository and the PDD command labels are visible.
- Confirm you can apply labels and post issue comments.
- Open the PDD setup flow if the App reports missing credentials or credits; resolve the exact preflight message before retrying.

Using the CLI

```
pdd --version
pdd auth status --verify
gh auth status
git status --short
```

Run your project's normal test command once. If the baseline already fails, record the failure in the issue before asking PDD to change anything.

> **Your first milestone:** one narrow issue, one working route, one clean baseline, and one testable demo path.

A Practical Hackathon Rhythm

When	Participant focus	Evidence to leave behind
Start	Pick one user-visible outcome, open the issue, and prove the baseline	Acceptance criteria and baseline test result
First build	Run `pdd-issue` or `pdd change`; answer questions quickly	Live progress, branch, and initial tests
Midpoint	Exercise the real UI/API path and reduce scope if needed	Working vertical slice and updated issue decisions
Final hour	Stop expanding, run the full verification ladder, and rehearse the demo	Green CI, checkup result, and repeatable demo path

Keep one person responsible for the issue and run controls while the rest of the team reviews behavior, tests the product, and prepares the demonstration.

PDD / PARTICIPANT FIELD GUIDE

GitHub App Workflow

The GitHub App is the simplest shared interface. The issue is the control surface, comments are the feedback channel, labels choose the workflow, and the resulting PR is the review artifact.

Install and Link

1. Open the PDD GitHub App.
2. Select **Configure** or **Install** and choose the personal account or organization that owns the repository.
3. Prefer **Only select repositories** for a hackathon unless organization-wide installation is intentional.
4. Sign in to PDD Cloud when redirected and link the installation.
5. In the setup wizard, wait until repository auto-configuration is complete.
6. If a repository is missing, add it in GitHub installation settings and recheck PDD Cloud repository health.

Recommended End-to-End Run

1. Create one issue for one demonstrable outcome.
2. Include acceptance criteria, forbidden outcomes, reproduction evidence, and the expected validation command.
3. Apply `pdd-issue`. No assignment is required.
4. Open the bot's live-progress link. Keep product decisions in the issue, where the team can see them.
5. If PDD asks for clarification, answer directly and concretely. Normal issue comments steer the active run.
6. Review the generated PR. Confirm it closes or clearly references the issue.
7. Run the application and project tests yourself.
8. Apply `pdd-checkup` to the linked PR for the final hosted gate.

The App performs preflight checks for repository access, credentials, and available credits before launching compute. A preflight failure should produce an actionable issue comment without starting the expensive run.

Choose the Right Label

Primary command labels

Label	Use it for	Expected outcome
`pdd-issue`	Autonomous solution of a well-scoped issue	Investigation, implementation, verification, and durable PR evidence
`pdd-change`	A feature or behavioral change	Prompt-owned changes and a linked PR
`pdd-bug`	Bug reproduction and analysis	Root cause plus failing regression tests
`pdd-fix`	Repair after reproduction	Code/prompt repair with test and CI validation
`pdd-test`	UI or end-to-end test generation	Tests derived from the issue

Label	Use it for	Expected outcome
`pdd-generate`	Architecture from a PRD issue	Architecture and prompt scaffolding
`pdd-sync`	Synchronize PDD source and artifacts	Prompt, code, example, and test synchronization
`pdd-connect`	Hosted interactive PDD session	Browser-accessible session

Review labels

Label	Behavior
`pdd-checkup`	Reviews an existing PR; with a linked issue, runs the canonical final PR gate and may fix findings
`pdd-checkup-nofix`	Reports PR findings without modifying the branch; not a final ready-to-merge signal
`pdd-checkup-prompt`	Checks changed prompt files and auto-applies fixable findings
`pdd-checkup-prompt-nofix`	Reports changed-prompt findings without writing or pushing

Modifier labels

Label	Behavior
`pdd-clean-restart`	One-shot restart modifier for `pdd-change`, `pdd-bug`, `pdd-fix`, or `pdd-test`; apply it alongside the command label
`pdd-decompose`	Forces decomposition of a large task when applied before `pdd-issue`

Model and effort labels are also available, but the default managed routing is the safest choice for a first run. Use a model override only when your team has confirmed provider access, cost, and availability.

Feature and Bug Patterns

Feature

```
Create issue -> apply pdd-change -> answer questions -> review PR -> apply pdd-checkup
```

Bug

```
Create issue -> apply pdd-bug -> review failing tests -> apply pdd-fix
             -> verify tests/CI -> apply pdd-checkup
```

The bug flow is deliberately two-stage. Do not start `pdd-fix` until the regression test fails for the correct reason. A passing, flaky, or over-broad test is not a useful mold wall.

Run Controls

Post one control per issue comment.

Comment	Effect
Normal human comment	Steers active work or answers clarification
`/pdd settings`	Shows effective budget, spend, run status, and explicit strength
`/pdd budget 30`	Sets a USD budget cap for a standard command run
`/strength 0.7`	Sets a combined model-strength and reasoning-effort dial for the next invocation
`/pdd stop`	Stops active work and removes the trigger label when appropriate
`/pdd continue codex`	When a `pdd-issue` run is waiting on a provider, switches and resumes with Codex; `claude` and `gemini` are also valid targets
`/pdd note text`	Leaves a human note that PDD ignores for steering and feedback

Controls that change a run require repository write/admin permission. `/pdd settings` is read-only. To retry after fixing a credential, credit, or permission problem, re-apply the trigger label. To restart from step 1, apply `pdd-clean-restart` alongside the supported command label.

Select an Exact PR for Checkup

On an issue, `pdd-checkup` first looks for an open PR that natively closes the issue, then considers issue-number/branch references. If no unique candidate is found, choose it explicitly in an issue comment:

```
/pdd checkup --pr https://github.com/OWNER/REPO/pull/123
```

Add `--no-fix` for report-only behavior. Comment this command on the issue, not the PR.

PDD / PARTICIPANT FIELD GUIDE

CLI Workflow

The CLI exposes the same core workflows while preserving terminal visibility and local project access.

One-Time Setup

```
# Install uv on macOS/Linux if needed.
curl -LsSf https://astral.sh/uv/install.sh | sh

# Install PDD. If already installed, use the upgrade command instead.
uv tool install pdd-cli
uv tool upgrade pdd-cli

pdd --version
pdd setup
pdd auth login

# Issue-driven workflows use GitHub CLI.
gh auth login
gh auth status
```

If `uv` is unavailable, use `pip install pdd-cli`. `pdd setup` detects supported agentic CLIs and credentials. Complete at least one offered Codex, Claude, Gemini/Antigravity, or OpenCode setup. Stored subscription/OAuth logins can be reused; direct local LiteLLM execution may instead need a provider API key.

Preflight

Run this from the cloned repository root before starting a costly workflow:

```
pdd --version
pdd auth status --verify
gh auth status
git status --short

# Replace with the repository's real baseline check.
pytest -q
```

Commit or stash unrelated edits. PDD agentic commands use isolated worktrees, but a clean baseline makes failures and resulting diffs easier to interpret.

Feature Workflow

```
ISSUE=https://github.com/OWNER/REPO/issues/123
pdd change "$ISSUE"
```

PDD posts progress to the issue, identifies affected development units, works in an isolated worktree, verifies the result, and opens a linked PR. If it pauses for clarification or an architecture decision:

1. Answer on the issue.
2. Run the same command again.
3. Let saved GitHub state resume the workflow from the last completed step.

Bug Workflow

```
ISSUE=https://github.com/OWNER/REPO/issues/456

# Analyze, reproduce, and create failing regression tests.
pdd bug "$ISSUE"

# After reviewing the tests, repair without weakening them.
pdd fix --protect-tests "$ISSUE"
```

Use `--protect-tests` only after confirming the tests encode the intended behavior. If the test is wrong, fix the test or issue specification before protecting it.

CLI Command Reference

Goal	Command
Feature from an issue	`pdd change ISSUE_URL`
Bug investigation and failing tests	`pdd bug ISSUE_URL`
Repair an issue with trusted tests	`pdd fix --protect-tests ISSUE_URL`
UI/end-to-end tests from an issue	`pdd test ISSUE_URL`
New architecture from a PRD issue	`pdd generate ISSUE_URL`
Multi-module issue synchronization	`pdd sync ISSUE_URL`
Launch the browser interface	`pdd connect`
Validate user stories	`pdd detect --stories`
Final gate for a linked PR	`pdd checkup --pr PR_URL --issue ISSUE_URL --final-gate`
Discover current options	`pdd COMMAND --help`

Resume or Restart

- **Normal interruption:** rerun the same issue command. GitHub comment state supports cross-machine resume.
- **Local-only state:** add `--no-github-state` when issue comment persistence is not appropriate.
- **Wrong or contaminated state:** add `--clean-restart` to `change`, `bug`, `fix`, or `test`.
- **Do not clean-restart by habit:** resuming saves completed work and budget.

PDD / PARTICIPANT FIELD GUIDE

Write an Effective Issue

PDD can research implementation details, but it should not invent a product decision. The issue should define observable behavior and enough evidence to reproduce the task.

Copyable Issue Template

```
## Goal
<One sentence describing the user-visible outcome.>

## Acceptance criteria
1. Given <starting state>, when <action>, then <observable result>.
2. <Required error, state change, output, API shape, or UI behavior.>
3. Existing <named behavior/test> still passes.

## Must not
- Must not expose secrets or personal data.
- Must not change <out-of-scope interface or behavior>.

## Evidence
- Reproduction steps, logs, screenshot, or failing test:
- Relevant files/routes/docs:
- Expected result:
- Actual result:

## Validation
- Install/build command:
- Targeted test command:
- Manual demo path:

## Done when
- Tests cover the positive case and important forbidden outcome.
- The issue, generated code, tests, and docs agree.
- The linked PR passes CI and PDD checkup.
```

Quality Test

Before applying a label or running the CLI, ask:

- Can a reviewer observe every acceptance criterion?
- Are important forbidden outcomes stated with **must not** language?
- Is the intended interface or output shape explicit?
- Does the issue distinguish expected from actual behavior?
- Are logs and screenshots redacted?
- Is the task narrow enough for one coherent PR?

Weak: "Build team registration."

Stronger: "A signed-in participant can create one team, invite up to three members, and see the same membership after refresh. Duplicate membership is rejected without creating a partial record. Anonymous requests return 401."

Scope Large Work Deliberately

Split a task when it contains multiple independently demonstrable outcomes, crosses unrelated subsystems, or cannot be verified with one coherent test plan. For the App, apply `pdd-decompose` before `pdd-issue`. For manual planning, create a parent issue with child issues and explicit dependency order.

PDD / PARTICIPANT FIELD GUIDE

Verify and Review

Treat PDD output like compiler output: inspect it, test it, and reject it when the source-to-artifact contract is not satisfied.

Verification Ladder

1. **Static sanity:** format, lint, type-check, and build.
2. **Targeted behavior:** run tests for the changed development unit.
3. **Negative behavior:** prove forbidden side effects and invalid input paths.
4. **Integration behavior:** exercise cross-module interfaces and persistence.
5. **Regression:** run the normal project suite and previously approved stories.
6. **Manual demo:** execute the exact user flow the team will present.
7. **Final gate:** run PDD checkup against the linked issue and PR.

Final CLI Gate

```
pdd checkup \
  --pr https://github.com/OWNER/REPO/pull/123 \
  --issue https://github.com/OWNER/REPO/issues/456 \
  --final-gate
```

The final gate is stricter than a general report. It requires issue alignment, valid current-head evidence, passing checks, and a clean reviewer verdict.

Pull Request Review Checklist

1. Confirm the PR closes or clearly links the correct issue.
2. Match every acceptance criterion to test or manual evidence.
3. Confirm important MUST NOT behavior has a negative test.
4. Review public interface changes and confirm callers were updated.
5. Inspect the diff and logs for secrets, tokens, private data, and credential files.

6. Run the normal test suite and confirm CI passes on the current PR head.
7. Exercise the application through the exact demo path.
8. Run `pdd-checkup` or the CLI final gate.

Do Not Accept These Signals Alone

- The code looks plausible.
- The agent says the work is complete.
- One happy-path test passes.
- A PR exists but is draft, conflicted, or behind failing CI.
- Generated code changed without corresponding issue or test intent.
- A test was weakened to make the implementation pass.

PDD / PARTICIPANT FIELD GUIDE

Security, Credentials, and Cost

Hackathon speed does not justify leaking credentials or giving automation unbounded access.

Credential Rules

- Never paste API keys, OAuth tokens, `.env` contents, JWTs, or private logs into issues, prompts, comments, screenshots, or commits.
- Use `pdd setup`, provider login flows, environment variables, or the event's approved secret manager.
- Install the GitHub App on only the repositories it needs.
- Redact secrets before attaching a PDD core dump or execution log.
- Revoke and replace any credential that may have entered Git history.

Cost Controls

- The App checks for sufficient PDD Cloud credits before launching a job.
- Use `/pdd settings` to inspect hosted spend and effective budget.
- Use `/pdd budget N` to bound a standard hosted command run.
- CLI workflows expose command-specific `--budget` options where supported; check `pdd COMMAND --help` for the installed version.
- Resume useful state rather than repeatedly clean-restarting.
- Scope issues tightly and provide evidence up front to reduce clarification and rework.

PDD / PARTICIPANT FIELD GUIDE

Troubleshooting

Symptom	First action
`pdd: command not found`	Open a new shell or add the uv tool bin directory to `PATH`; then run `uv tool upgrade pdd-cli`
CLI says unauthenticated	Run `pdd auth login`, then `pdd auth status --verify`
CLI cannot read the issue	Run `gh auth status` and confirm access to the repository
No supported agent is detected	Re-run `pdd setup` and configure one supported agentic CLI/login
App labels are missing	Confirm the App is installed on that repository; check PDD Cloud repository health
App run does not start	Remove/re-apply the compound label and read the latest credential, credit, or permission comment
Workflow asks a question	Answer on the issue, then rerun the same CLI command or follow the App's re-label instruction
Run is waiting on a provider	Use the posted queue choices; for `pdd-issue`, `/pdd continue <provider>` can switch and resume
Wrong state keeps resuming	Use CLI `--clean-restart` or the `pdd-clean-restart` App modifier
Correct tests were weakened	Restore/review them, then use `pdd fix --protect-tests ISSUE_URL`
The PR does not match the issue	Update the issue or tests, then rerun the App or CLI workflow
Several PRs match checkup	Comment `/pdd checkup --pr <exact-url>` on the issue
CI fails after local success	Inspect the current PR head, environment differences, and CI logs before retrying PDD
A command option is rejected	Run `pdd COMMAND --help`; installed CLI help is authoritative

When reporting a CLI failure, include `pdd --version`, the exact command, the final redacted error, and the redacted core dump path printed by PDD. Never attach tokens, `.env` files, credential caches, or unredacted logs.

PDD / PARTICIPANT FIELD GUIDE

Participant Cheat Sheet

Start

GitHub App: write issue -> apply `pdd-issue` -> follow progress -> review PR

CLI feature:

```
pdd change https://github.com/OWNER/REPO/issues/123
```

CLI bug:

```
pdd bug https://github.com/OWNER/REPO/issues/456
pdd fix --protect-tests https://github.com/OWNER/REPO/issues/456
```

Hosted Labels

Need	Label
End-to-end issue solving	`pdd-issue`
Feature	`pdd-change`
Reproduce bug	`pdd-bug`
Fix reproduced bug	`pdd-fix`
Generate tests	`pdd-test`
Synchronize PDD artifacts	`pdd-sync`
Final linked-PR gate	`pdd-checkup`
Report-only PR review	`pdd-checkup-nofix`

Hosted Controls

Need	Comment
Inspect run	`/pdd settings`
Set budget cap	`/pdd budget 30`
Steer work	Normal issue comment
Stop work	`/pdd stop`
Ignore a note	`/pdd note text`
Select exact PR	`/pdd checkup --pr PR_URL`

Definition of Done

1. Acceptance criteria are observable and satisfied.
2. Positive and important negative paths are tested.
3. The issue, code, tests, and docs agree.
4. No secrets or private data appear in the diff or logs.
5. Project tests and CI pass on the current PR head.
6. The exact demo flow works manually.
7. The linked PR passes PDD checkup.

> When in doubt: improve the issue, add a test, rerun the GitHub App or CLI workflow, and verify the result.

PDD / PARTICIPANT FIELD GUIDE

Source Material

This guide condenses the current behavior documented and implemented in these sources:

- PDD README
- PDD CLI documentation
- Issue-driven development guide
- Prompting guide
- Prompt-Driven Development doctrine
- The Last Programming Language
- PDD GitHub App

The repository README and `pdd COMMAND --help` are authoritative when an older web page shows a different step count or option name.