/learning-opportunity

Source: `~/.claude/skills/learning-opportunity/SKILL.md`

---

name: learning-opportunity description: Self-improving feedback loop. When something goes wrong, analyze root cause, patch the system, and ensure it never happens again. argument-hint: "[describe what went wrong]"

Learning Opportunity

Turn every mistake into a permanent system improvement.

Instructions

You are analyzing a mistake or failure and patching the system so it never recurs.

Principle: AI bez enforcement-a ne radi. Markdown rules = suggestions. Hooks/scripts = enforcement. Always prefer deterministic fixes over documentation fixes.

Step 1: Identify the Failure

If argument provided, use it. Otherwise, ask:

- What went wrong?
- When did it happen?
- What was the expected vs actual outcome?

Classify the failure type:

- **HALLUCINATION** — AI invented something that doesn't exist (tool, path, port, import)
- **PROCESS_SKIP** — AI skipped a required step (no boot, no backup, no task)
- **WRONG_OUTPUT** — AI produced incorrect content (wrong data, bad code, broken logic)
- **KNOWLEDGE_GAP** — AI didn't know something it should have known
- **REPEAT_MISTAKE** — Same error as a previous session (worst category)

Step 2: Root Cause Analysis

Trace the failure through GOTCHA layers:

1. **Goals** — Was there a spec/rule that should have prevented this?

```
# Check existing rules
ls ~/system/rules/
grep -r "relevant keyword" ~/system/rules/
```

2. **Tools** — Did a tool fail or was a phantom tool used?

```
# Check manifest
grep "relevant tool" ~/system/tools/manifest.md
```

3. **Context** — Was the context missing or wrong?

```
# Check HiveMind for prior knowledge
node ~/system/agents/hivemind/hivemind.js query "relevant keyword"
```

4. **Hooks** — Should an enforcement hook have caught this?

```
# Check existing hooks
ls ~/.claude/hooks/
```

5. **Memory** — Was this a known issue that was forgotten?

```
# Check memory files
grep "relevant keyword" ~/.claude/projects/-Users-makinja/memory/MEMORY.md
```

Document: Which layer failed? Why?

Step 3: Determine Fix Type

Choose the STRONGEST fix available (top = strongest):

Priority	Fix Type	When to Use
1	**Hook** (Python enforcement)	Hallucinations, phantom tools, security violations
2	**Tool update** (deterministic code)	Missing validation, wrong behavior
3	**Rule addition** (~/system/rules/)	New process requirement, agent behavior
4	**CLAUDE.md update**	Missing instruction, wrong priority
5	**Memory update**	Lesson learned, context for future

NEVER use only option 5 alone. Memory without enforcement = ZAKON #1 violation.

Step 4: Apply the Patch

Based on fix type, apply changes:

If HALLUCINATION ? Update hallucination-detector.py

```
# Read current blocklist
grep -A 50 "PHANTOM_TOOLS" ~/.claude/hooks/hallucination-detector.py
```

Add the hallucinated item to the appropriate blocklist (PHANTOM_TOOLS, KNOWN_PORTS, etc.)

If PROCESS_SKIP ? Update/create enforcement hook

Check if gotcha-enforcer.py can be extended, or create new hook.

If WRONG_OUTPUT ? Update tool or add validation

Fix the tool that produced wrong output. Add input validation.

If KNOWLEDGE_GAP ? Add to context + memory

```
# Add to HiveMind
node ~/system/agents/hivemind/hivemind.js post john lesson "description"
```

If REPEAT_MISTAKE ? Escalate enforcement

If this mistake happened before, the previous fix was too weak. Go UP the priority list (e.g., if rule exists but wasn't followed → add hook).

Step 5: Verify the Fix

Test that the fix actually works:

- If hook: test with a simulated bad input
- If tool: run the tool and verify output
- If rule: check it's in the right location and formatted correctly

Step 6: Log Everything

```
# 1. Log to CHANGELOG
bash ~/system/tools/syslog.sh add "LEARNING: [description] — fix: [what was changed]"

# 2. Log to HiveMind
node ~/system/agents/hivemind/hivemind.js post john lesson "[failure type]: [what happened] → [what was fixed]"

# 3. Update lessons-learned if exists
# ~/system/rules/lessons-learned.md
```

Step 7: Report

Show the user:

```
## Learning Opportunity Report

### Failure
- **Type:** [HALLUCINATION|PROCESS_SKIP|WRONG_OUTPUT|KNOWLEDGE_GAP|REPEAT_MISTAKE]
- **Description:** [what went wrong]
- **Root Cause:** [which GOTCHA layer failed and why]

### Fix Applied
- **Fix Type:** [Hook|Tool|Rule|CLAUDE.md|Memory]
- **File Changed:** [path]
- **What Changed:** [description]

### Enforcement Level
- [ ] Deterministic (hook/script blocks bad behavior)
- [ ] Documented (rule/instruction guides good behavior)
```

```
- [ ] Remembered (memory/HiveMind for context)

### Verification
- [ ] Fix tested and working
- [ ] Logged to CHANGELOG
- [ ] Logged to HiveMind
```

Rules

1. **Deterministic > Documented** — A hook that blocks is worth 100 markdown rules
2. **ZAKON #1 applies** — If the fix is "write more markdown", it's NOT a fix
3. **Escalate repeats** — Same mistake twice = previous fix was too weak
4. **Always log** — CHANGELOG + HiveMind, no exceptions
5. **Backup first** — `setup-backup.sh` before any hook/tool changes

$ARGUMENTS

Revision #5
Created 2026-02-18 08:39:48 UTC by John
Updated 2026-06-21 20:00:42 UTC by John