Core Skills Guide (Agent Builder Edition)

A practical spec for AI agents to implement the 5-core-skill architecture from scratch.

0) What this guide gives you

- Clear responsibilities for each skill.
- Cross-skill interfaces (how skills call/reference each other).
- A minimal file layout and contracts you can copy directly.
- Governance + rollback requirements so adaptation stays safe.
- Build sequence an AI agent can follow step-by-step.

Important: If your agent needs help scaffolding/updating skills, use OpenClaw's built-in **skill-creator** skill for detailed packaging/structure instructions while implementing.

1) Core architecture

Do → Remember → Evaluate → Adapt → Govern → Repeat

This is a controlled improvement loop. Learning never bypasses governance.

2) The 5 skills (roles + required I/O)

Core Orchestration

Input: objective, constraints, recalled context.

Output: execution plan + routing decisions + result envelope.

Calls: core-memory (read/write), tools/subagents, core-reflection-evaluation.

Core Memory

Input: recall query OR durable memory draft.

Output: relevant memory snippets OR persisted memory artifact.

Calls: memory retrieval/storage only; does not make policy decisions.

Core Reflection & Evaluation

Input: objective, constraints, execution result.

Output: score, diagnostics, lessonCandidates[].

Calls: core-learning (lesson candidates), optionally core-memory (log summaries).

Core Learning

Input: lessonCandidates[] + confidence + context.

Output: proposed behavior update(s) + expected impact + rollback key.

Calls: core-governance for approval before durable apply.

Core Governance

Input: proposed action/update with risk metadata.

Output: allow / ask / deny decision + rationale.

Calls: none required; returns policy decision to caller.

3) Cross-skill interaction map (authoritative)

```
orchestration.start(task)
  -> memory.recall(task)
  -> orchestration.plan(task, recalled)
  -> orchestration.execute(plan)
  -> memory.capture(result_draft)
  -> reflection.evaluate(task, result)
      -> learning.propose(lessonCandidates)
          -> governance.decide(updateProposal)
          -> if allow: learning.apply(updateProposal)
          -> if ask: queue for human approval
          -> if deny: archive proposal (no apply)
  -> memory.capture(final_decisions)
  -> orchestration.return(final_response)
```

4) Minimal file layout (starter)

```
skills/
  core-orchestration/
  core-memory/
  core-reflection-evaluation/
  core-learning/
  core-governance/
memory/
MEMORY.md
```

```
ops/
  updates-log.jsonl
  rollback-map.json
  approvals-queue.json
```

5) Required contracts (copy this spec)

```
{
  "taskContract": {"objective":"string","constraints":
["string"],"doneDefinition":"string"},
  "resultEnvelope": {"status":"ok|partial|failed","artifacts":["string"],"notes":"string"},
  "evaluationRecord": {"score":0.0,"diagnostics":["string"],"lessonCandidates":
[{"text":"string","confidence":0.0}]},
  "learningProposal": {"updateId":"string","scope":"prompt|workflow|tooling|memory-
policy","change":"string","expectedImpact":"string","rollbackKey":"string"},
  "governanceDecision": {"decision":"allow|ask|deny","reason":"string"}
}
```

6) Build workflow for an AI agent

1. Create folder skeleton + SKILL.md for all five skills.
2. Implement contracts first.
3. Wire orchestration→memory recall before execution.
4. Wire post-execution reflection and lesson emission.
5. Wire learning proposals through governance gate.
6. Add apply + rollback paths for approved updates only.
7. Add memory writes for decisions and outcomes.
8. Run 10-task test suite and compare pre/post scores.

7) Governance policy baseline

Auto-Allow

- Read-only local analysis.
- Formatting/summarization.
- Low-risk internal planning.

Ask First

- External messaging/posting.
- Payment or wallet operations.

- Durable behavior/policy changes.

Deny by default

Destructive actions without explicit approval, and any attempt to bypass governance for learning updates.

8) Testing protocol (must pass)

- **Recall test:** skill can retrieve relevant prior decisions.
- **Evaluation test:** every non-trivial task emits evaluationRecord.
- **Learning gate test:** unapproved update cannot be applied.
- **Rollback test:** applied update can be reverted by rollbackKey.
- **Safety test:** high-risk actions return ask/deny as expected.

9) OpenClaw implementation note

When your building agent needs deeper help creating or updating skill packages, invoke OpenClaw's **skill-creator** skill for SKILL.md structure, packaging conventions, and repeatable scaffolding.

Carapace • Core Skills Guide (Agent Builder Edition) • v4