Claude Code - Complete Setup Guide

Building Your Own Autonomous AI Development Agent

Author: Martien de Jong's Claude Agent
Date: January 24, 2026
Repository: https://github.com/martiendejong/machine_agents
Purpose: Guide for replicating this autonomous agent system on any machine

---

📋 Table of Contents

🎯 What This System Does

This is not just "Claude with some prompts." This is a **full autonomous development control plane** that:

Core Capabilities

- ✅ **Manages GitHub** - Creates PRs, reviews code, tracks dependencies, fixes CI/CD failures
- ✅ **Manages ClickUp** - Lists tasks, updates status, creates new tasks, posts comments, links PRs
- ✅ **Reads & Sends Email** - IMAP/SMTP access to multiple inboxes, search, archive, spam management
- ✅ **Controls Google Drive** - Read, write, search, organize files via OAuth MCP integration
- ✅ **Monitors System Activity** - Real-time tracking via ManicTime, detects user presence, counts Claude instances
- ✅ **Deploys to Production** - Frontend/backend deployment via SSH, RDP, IIS, with automated verification
- ✅ **Automated Backups** - SQL Server, config files, automated rotation, point-in-time restore
- ✅ **Debugs Visual Studio** - HTTP API control: breakpoints, step execution, variable inspection
- ✅ **Tests Browser Apps** - Frontend debugging via Browser MCP server

Autonomous Workflows

- 🤖 **Self-improving** - Logs mistakes, updates own documentation, creates new tools
- 🤖 **Multi-agent coordination** - Multiple Claude instances work in parallel without conflicts
- 🤖 **Context-aware** - Knows what you're doing, adapts assistance based on activity
- 🤖 **Task-driven** - Picks up ClickUp tasks, codes features, creates PRs, updates status automatically

---

🔧 Prerequisites

Required Software

- **Git** (2.30+) - Version control
- **GitHub CLI** (`gh`) - Repository operations
- **Node.js** (LTS 18+) - Runtime for Claude Code
- **Claude Code CLI** - `npm install -g @anthropic-ai/claude-code`

Optional (Based on Your Stack)

- **.NET SDK** (if C# development)
- **Python** (if Python development)
- **Docker** (for containerized apps)
- **Visual Studio** (for C# debugging with Agentic Debugger Bridge)

Accounts & API Keys

- **Anthropic API key** - For Claude Code
- **GitHub account** - With personal access token
- **ClickUp account** (optional) - With API key
- **Google Cloud project** (optional) - For Google Drive integration
- **Email IMAP/SMTP credentials** (optional) - For email management

---

🚀 Quick Start (3 Steps)

Step 1: Clone the Repository

```
# Windows
git clone https://github.com/martiendejong/machine_agents.git C:\scripts

# Mac/Linux
git clone https://github.com/martiendejong/machine_agents.git ~/.claude
```

Step 2: Run Bootstrap

```
# Windows
cd C:\scripts
.\bootstrap\bootstrap.ps1

# Mac/Linux (adapt paths in bootstrap script)
cd ~/.claude
./bootstrap/bootstrap.sh  # Note: Script is PowerShell, may need adaptation
```

The bootstrap will:

- ✅ Install dependencies (Git, gh CLI, Node.js, Claude Code)
- ✅ Create directory structure (projects, worktrees, machine context)
- ✅ Initialize tracking files (worktree pool, activity log, reflections)
- ✅ Verify environment

Step 3: Configure Your Machine

Edit `C:\scripts\MACHINE_CONFIG.md` (or `~/.claude/MACHINE_CONFIG.md`):

```
# Machine-Specific Configuration

## Directory Structure

BASE_REPO_PATH=C:\Projects  # Change to your projects folder
WORKTREE_PATH=C:\Projects\worker-agents
CONTROL_PLANE_PATH=C:\scripts
MACHINE_CONTEXT_PATH=C:\scripts\_machine

## Projects

### Project 1: your-project-name
- Location: C:\Projects\your-project
- Main branch: main  # or develop
- Repository: https://github.com/you/your-project

## Main Branch
<main-branch>=main  # or develop, master
```

📦 Detailed Setup

1. Directory Structure

The system expects this structure:

```
C:\Projects\  (or your BASE_REPO_PATH)
├── your-project\           # Base repository (always on main branch)
├── another-project\
└── worker-agents\          # Worktree pool for isolated development
    ├── agent-001\          # Agent seat 1
    ├── agent-002\          # Agent seat 2
    └── agent-003\          # Agent seat 3

C:\scripts\  (or your CONTROL_PLANE_PATH)
├── _machine\               # Machine context (tracking files)
│   ├── worktrees.pool.md
│   ├── worktrees.activity.md
│   ├── reflection.log.md
│   └── pr-dependencies.md
├── .claude\
│   └── skills\             # Auto-discoverable Claude Skills
├── tools\                  # 99+ productivity scripts
├── bootstrap\              # Environment setup scripts
└── MACHINE_CONFIG.md       # Your configuration
```

2. Core Documentation Files

ALWAYS READ AT SESSION START:

1. `MACHINE_CONFIG.md` - Your paths and projects
2. `GENERAL_ZERO_TOLERANCE_RULES.md` - Critical rules
3. `GENERAL_DUAL_MODE_WORKFLOW.md` - Feature vs Debug mode
4. `_machine/SOFTWARE_DEVELOPMENT_PRINCIPLES.md` - Code quality standards

3. Dual-Mode Workflow

The system operates in TWO modes:

Feature Development Mode

- **When:** Building new features, refactoring, planned work
- **Where:** Isolated worktrees (`C:\Projects\worker-agents\agent-XXX\`)
- **Branch:** New feature branch
- **Workflow:** Allocate worktree → Work → PR → Release worktree
- **Base repo:** Always stays on `main`/`develop`

Active Debugging Mode

- **When:** User is debugging, fixing build errors, quick fixes
- **Where:** Base repository (`C:\Projects\your-project\`)
- **Branch:** User's current branch (preserved)
- **Workflow:** Work directly → Commit
- **Speed:** Fast turnaround, no worktree overhead

Decision Tree:

```
User request?
├─ Has ClickUp URL? → ALWAYS Feature Development Mode
├─ Build error/debugging? → Active Debugging Mode
├─ New feature/refactoring? → Feature Development Mode
└─ Quick fix? → Active Debugging Mode
```

4. Worktree Protocol

Why worktrees?

- Allows multiple features in parallel
- Prevents branch switching in base repo
- Isolates work between multiple Claude instances
- Maintains clean git state

Allocation (Feature Development Mode):

```
# 1. Check pool status
cat C:\scripts\_machine\worktrees.pool.md

# 2. Allocate worktree
cd C:\Projects\your-project
git worktree add C:\Projects\worker-agents\agent-001\your-project -b agent-001-feature-name

# 3. Mark BUSY in worktrees.pool.md
# 4. Work in worktree
cd C:\Projects\worker-agents\agent-001\your-project
# ... make changes ...

# 5. Commit, push, create PR
git add .
git commit -m "feat: Add feature"
git push -u origin agent-001-feature-name
gh pr create

# 6. Release worktree
git worktree remove C:\Projects\worker-agents\agent-001\your-project
# 7. Mark FREE in worktrees.pool.md
# 8. Switch base repo to develop
cd C:\Projects\your-project
git checkout develop
```

Use tools to automate:

```
# Allocate
C:\scripts\tools\worktree-allocate.ps1 -Repo your-project -Branch feature-x

# Release
C:\scripts\tools\worktree-release-all.ps1 -AutoCommit

# Status
C:\scripts\tools\worktree-status.ps1
```

🔌 Integration Capabilities

GitHub Integration

Tools: `gh` CLI + custom scripts

Capabilities:

- Create PRs with templates and dependency tracking
- Review code and post inline comments
- Merge PRs with automated sequencing
- Track cross-repo dependencies
- Monitor CI/CD and fix build failures
- Batch fix multiple PR builds

Example:

```
gh pr create --title "feat: Add feature" --body "Description"
gh pr review 123 --comment -b "LGTM"
gh pr merge 123 --squash
```

ClickUp Integration

Tools: `C:\scripts\tools\clickup-sync.ps1`

Setup:

1. Get ClickUp API key from https://app.clickup.com/settings/apps
2. Create config: `C:\scripts\_machine\clickup-config.json`

```
{
  "api_key": "pk_YOUR_API_KEY",
  "api_base": "https://api.clickup.com/api/v2"
}
```

Capabilities:

- List tasks from specific lists
- Update task status (todo → busy → review → done)
- Create new tasks
- Post comments with PR links
- Autonomous task pickup and completion

Example:

```
# List tasks
clickup-sync.ps1 -Action list

# Update status
clickup-sync.ps1 -Action update -TaskId "abc123" -Status "busy"

# Post comment
clickup-sync.ps1 -Action comment -TaskId "abc123" -Comment "PR #45 created"
```

Email Integration

Tools: `C:\scripts\tools\email-manager.js`, `send-email.js`

Setup:

1. Configure IMAP/SMTP credentials in `email-manager.js`
2. Supports multiple accounts

Capabilities:

- List, search, read emails
- Move to spam, archive, trash
- Send emails with attachments
- Import email history

Example:

```
# Read emails
node email-manager.js list --count=10

# Search
node email-manager.js search "anthropic"

# Send
node send-email.js --to frank@example.com --subject "Hello" --body "Message"
```

Google Drive Integration

Tools: MCP server (`@modelcontextprotocol/server-gdrive`)

Setup:

1. Create Google Cloud project
2. Enable Google Drive API
3. Create OAuth 2.0 credentials (Desktop app)
4. Download credentials JSON
5. Configure MCP server in `~/.claude.json`:

```
{
  "mcpServers": {
    "gdrive": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-gdrive"],
      "env": {
        "GDRIVE_OAUTH_PATH": "C:\\scripts\\_machine\\gcp-oauth.keys.json",
        "GDRIVE_CREDENTIALS_PATH": "C:\\scripts\\_machine\\gdrive-credentials.json"
      }
    }
  }
}
```

Capabilities:

- List, read, create, update files
- Search across Drive content
- OAuth authentication with persistent credentials

ManicTime Integration

Tools: `C:\scripts\tools\monitor-activity.ps1`

Setup:

1. Install ManicTime (https://www.manictime.com/)
2. Script queries local SQLite database

Capabilities:

- Track current user activity (application, window title)
- Detect idle/unattended system
- Count running Claude instances
- Analyze work patterns
- Provide context for adaptive assistance

Example:

```
# Get current activity
monitor-activity.ps1 -Mode current

# Count Claude instances
monitor-activity.ps1 -Mode claude

# Full context for AI
monitor-activity.ps1 -Mode context -OutputFormat json
```

Production Deployment

Tools: `C:\scripts\tools\deploy.ps1`, `validate-deployment.ps1`

Methods:

- **MSDeploy** - Frontend to VPS via Web Deploy
- **IIS** - Backend to Windows Server
- **SSH** - Remote commands on Linux VPS
- **RDP** - Windows server control

Capabilities:

- Build → Deploy → Verify pipeline
- Post-deployment health checks
- Rollback on failure
- Automated certificate management

Example:

```
# Deploy frontend
deploy.ps1 -Target frontend

# Deploy both
deploy.ps1 -Target both

# Verify
validate-deployment.ps1 -ProjectPath . -Environment production
```

Automated Backups

Tools: `C:\scripts\tools\backup-restore.ps1`

Capabilities:

- SQL Server backups (full, differential, transaction log)
- Config file backups (appsettings.json, .env, secrets)
- Automated rotation (keep N most recent)
- Point-in-time restore
- Verification after creation
- Task Scheduler integration

Example:

```
# Backup database
backup-restore.ps1 -Action backup -Type database -DatabaseName MyApp

# Backup config files
backup-restore.ps1 -Action backup -Type config -ConfigPaths "C:\Projects\app\appsettings.json"

# Restore
backup-restore.ps1 -Action restore -DatabaseName MyApp -RestorePoint "backup-2026-01-24.bak"

# Cleanup old backups
backup-restore.ps1 -Action cleanup -KeepCount 7
```

Visual Studio Debugging

Tools: Agentic Debugger Bridge (HTTP API)

Setup:

1. Install Agentic Debugger Bridge extension in Visual Studio
2. API runs at `http://localhost:27183`
3. Connection info in `%TEMP%\agentic_debugger.json`

Capabilities:

- Start/stop debugging
- Set/clear breakpoints
- Step into/over/out
- Evaluate expressions
- Read local variables and call stack
- Build/clean/rebuild
- Read error list and output panes
- Multi-instance support

Example HTTP requests:

```
# Get current state
curl http://localhost:27183/state

# Set breakpoint
curl -X POST http://localhost:27183/command \
  -H "Content-Type: application/json" \
  -d '{"action":"setBreakpoint","file":"C:\\path\\file.cs","line":42}'

# Start debugging
curl -X POST http://localhost:27183/command \
  -H "Content-Type: application/json" \
  -d '{"action":"start","projectName":"MyApp"}'

# Step over
curl -X POST http://localhost:27183/command \
  -H "Content-Type: application/json" \
  -d '{"action":"stepOver"}'
```

Browser Debugging

Tools: Browser MCP server

Setup:

```
claude mcp add browser -s user -- npx -y @modelcontextprotocol/server-browser
```

Capabilities:

- Navigate to URLs
- Interact with DOM elements
- Take screenshots
- Access console logs
- Inspect page elements
- Frontend testing automation

🏗 Architecture Overview

Component Diagram

```
┌──────────────────────────────────────────────────────────────┐
│                    Claude Code Agent                          │
│  (Autonomous AI with full machine access)                    │
└──────────────────────────────────────────────────────────────┘
                      │
                      │
        ┌─────────────┼─────────────┐
        │             │             │
        ▼             ▼             ▼
┌──────────────┐ ┌──────────┐ ┌──────────────┐
│  Git/GitHub  │ │ ClickUp  │ │  Email/Drive │
│  - gh CLI    │ │ - API    │ │  - IMAP/SMTP │
│  - worktree  │ │ - Tasks  │ │  - MCP       │
└──────────────┘ └──────────┘ └──────────────┘
        │             │             │
        └─────────────┼─────────────┘
                      │
        ┌─────────────┼─────────────┐
        │             │             │
        ▼             ▼             ▼
┌──────────────┐ ┌──────────┐ ┌──────────────┐
│ ManicTime    │ │ VS Debug │ │   Deploy     │
│ - Activity   │ │ - HTTP   │ │   - SSH/RDP  │
│ - Monitor    │ │ - Bridge │ │   - IIS      │
└──────────────┘ └──────────┘ └──────────────┘
        │             │             │
        └─────────────┼─────────────┘
                      │
                      ▼
        ┌──────────────────────────┐
        │  Machine Context         │
        │  - worktrees.pool.md     │
        │  - reflection.log.md     │
        │  - pr-dependencies.md    │
        └──────────────────────────┘
```

Data Flow

1. **Session Start:**
   - Read MACHINE_CONFIG.md → Load paths
   - Read GENERAL_ZERO_TOLERANCE_RULES.md → Load rules
   - Run `monitor-activity.ps1 -Mode context` → Get user context
   - Check worktrees.pool.md → See available agents
   - Read reflection.log.md → Learn from past mistakes
2. **Task Execution (Feature Development):**
   - User: "Implement PDF export feature"
   - Mode detection: New feature → Feature Development Mode
   - Allocate worktree: `agent-001`
   - Create branch: `agent-001-pdf-export`
   - Code changes in worktree
   - Build, test, verify
   - Commit, push, create PR
   - Link PR to ClickUp task
   - Release worktree, mark FREE
3. **Task Execution (Debugging):**
   - User: "Fix this build error"
   - Mode detection: Build error → Active Debugging Mode
   - Work in base repo on current branch
   - Fix error
   - Commit, push
   - No worktree, no branch switch
4. **Session End:**
   - Update reflection.log.md with learnings
   - Update PERSONAL_INSIGHTS.md with user patterns
   - Commit and push machine_agents repo

    - Verify all worktrees released

🎨 Creating Your First Autonomous Workflow

Example: Autonomous ClickUp Task Workflow

Goal: Claude picks up ClickUp tasks, codes features, creates PRs, updates task status automatically.

Setup:

1. **Create ClickUp Task:**
    - Title: "Add dark mode toggle to settings"
    - Status: "todo"
    - Description: "Add a toggle switch in settings to enable dark mode"
2. **Invoke Agent:**
```
claude --message "Check ClickUp for new tasks and complete them"
```
3. **Agent Executes:**
```
✅ Read MACHINE_CONFIG.md
✅ Run clickup-sync.ps1 -Action list
✅ Found task: "Add dark mode toggle" (status: todo)
✅ Allocate worktree: agent-002
✅ Create branch: agent-002-dark-mode-toggle
✅ Update task status: "busy"
✅ Post comment: "Working on this. Branch: agent-002-dark-mode-toggle"

✅ Code implementation:
   - Add toggle component
   - Add state management
   - Add theme switching logic
   - Update styles

✅ Build verification: PASSED
✅ Commit: "feat: Add dark mode toggle to settings"
✅ Push to origin
✅ Create PR #47
✅ Update task: Link PR #47
✅ Update task status: "review"
✅ Release worktree agent-002
✅ Update reflection.log.md
```
4. **Result:**
    - ✅ Feature implemented
    - ✅ PR created and linked
    - ✅ Task moved to review
    - ✅ Worktree released
    - ✅ Base repo on develop

Total time: ~5 minutes (autonomous)

🔥 Advanced Features

1. Multi-Agent Coordination

Scenario: 3 Claude instances working simultaneously without conflicts

How it works:

- Each instance checks worktrees.pool.md before allocating
- Agent seats are atomic (FREE/BUSY)
- ManicTime monitors all Claude instances
- Conflict detection prevents double-allocation
- Activity log tracks all operations

Tools:

```
# Check running Claude instances
monitor-activity.ps1 -Mode claude

# Check worktree availability
worktree-status.ps1 -Compact

# Parallel agent coordination skill
# (Auto-activates when multiple instances detected)
```

2. Cross-Repo PR Dependencies

Scenario: Feature requires changes in both `hazina` framework and `client-manager` app

Workflow:

1. Allocate PAIRED worktrees (same branch name):

   ```
   C:\Projects\worker-agents\agent-001\
   ├── hazina\            ← branch: agent-001-feature
   └── client-manager\    ← branch: agent-001-feature
   ```

2. Make changes in both repos
3. Create PRs with dependency tracking:

   ```
   PR #45 (hazina): "feat: Add PDF export to framework"
   PR #46 (client-manager): "feat: Use PDF export from Hazina"

   ⚠ DEPENDENCY ALERT: PR #46 depends on PR #45 (hazina)
   Merge order: #45 first, then #46
   ```

4. Track in pr-dependencies.md
5. Automated merge sequencing:

   ```
   merge-pr-sequence.ps1 -DryRun
   ```

3. Self-Improvement Protocol

How Claude learns from mistakes:

1. **Mistake happens:**
   - Build fails due to missing migration
   - PR created before EF migration added
2. **Reflection:**

```
## 2026-01-24 15:30 - EF Migration Mistake

**WHAT HAPPENED:**
Created PR without running `dotnet ef migrations has-pending-model-changes`
PR merged, runtime error: PendingModelChangesWarning

**ROOT CAUSE:**
Skipped pre-PR validation checklist

**FIX:**
- Created migration: `dotnet ef migrations add AddUserEmail`
- New PR #48 with migration
- Deployed fix

**PREVENTION:**
Updated GENERAL_ZERO_TOLERANCE_RULES.md:
- Added mandatory pre-PR validation section
- Build → Check migrations → Review → Commit → PR

**TOOL CREATED:**
ef-preflight-check.ps1 - Automated migration check

**SKILL CREATED:**
ef-migration-safety - Complete EF Core workflow

**LESSON:**
NEVER create PR for EF Core project without migration check.
This is now a HARD STOP rule.
```

3. **Documentation updated:**
    - reflection.log.md (logged above)
    - GENERAL_ZERO_TOLERANCE_RULES.md (new rule added)
    - ef-migration-safety skill created
    - Tool created for automation
4. **Future sessions:**
    - Claude reads reflection.log.md at startup
    - Sees mistake and prevention
    - Never repeats error

Result: System continuously improves from every mistake.

4. Claude Skills (Auto-Discoverable Workflows)

What are Skills?

- Markdown files in `.claude/skills/<skill-name>/SKILL.md`
- Auto-discovered by Claude at startup
- Activated when task matches skill description
- Self-contained workflow guides

Available Skills:

- `allocate-worktree` - Worktree allocation with conflict detection
- `release-worktree` - Complete release protocol
- `github-workflow` - PR creation, reviews, merging
- `clickhub-coding-agent` - Autonomous ClickUp task workflow
- `ef-migration-safety` - Safe EF Core migrations
- `rlm` - Recursive Language Model for massive contexts (10M+ tokens)
- `parallel-agent-coordination` - Multi-agent coordination
- `continuous-optimization` - Self-improvement meta-skill

How to create a skill:

```
# Use skill-creator skill
claude --message "Create a skill for database migrations"

# Or manually:
mkdir C:\scripts\.claude\skills\my-skill
cat > C:\scripts\.claude\skills\my-skill\SKILL.md <<EOF
---
name: my-skill
description: What this skill does (when to use it)
version: 1.0.0
auto_discover: true
---

# My Skill

## When to Use
[Description of when this skill activates]

## Workflow
1. Step 1
2. Step 2
3. Step 3

## Example
[Code or command examples]
EOF
```

🔧 Troubleshooting

Common Issues

1. "Worktree already exists"

Problem: Trying to allocate already-allocated worktree

Solution:

```
# Check status
worktree-status.ps1

# If stale (BUSY > 2 hours, no activity):
git worktree remove C:\Projects\worker-agents\agent-XXX\repo
# Update worktrees.pool.md to FREE
```

2. "Base repo not on develop"

Problem: Base repo on feature branch, can't create worktree

Solution:

```
cd C:\Projects\your-project
git checkout develop
git pull
```

3. "MCP server not loading"

Problem: Google Drive MCP not appearing

Solution:

```
# Validate config
cat ~/.claude.json | jq '.mcpServers'

# Test manually
npx -y @modelcontextprotocol/server-gdrive

# Check OAuth credentials exist
ls -la C:\scripts\_machine\gcp-oauth.keys.json
```

4. "Claude doesn't follow rules"

Problem: Claude makes mistakes documented in ZERO_TOLERANCE_RULES

Solution:

- Verify Claude reads MACHINE_CONFIG.md at startup
- Check that GENERAL_ZERO_TOLERANCE_RULES.md is in Claude's context
- Update reflection.log.md with specific mistake
- Create a skill for the workflow
- Add hard-stop validation (e.g., pre-commit hook)

5. "Email sending fails"

Problem: SMTP authentication error

Solution:

```javascript
// Check credentials in send-email.js
const transporter = nodemailer.createTransport({
  host: 'mail.example.com',  // Your SMTP host
  port: 587,                 // SMTP port (587 for STARTTLS, 465 for SSL)
  secure: false,             // true for 465, false for 587
  auth: {
    user: 'your-email@example.com',
    pass: 'your-password'     // App-specific password if 2FA enabled
  }
});
```

✅ Best Practices

1. Session Management

- ✅ **ALWAYS** read MACHINE_CONFIG.md first
- ✅ **ALWAYS** run `monitor-activity.ps1 -Mode context` at session start
- ✅ **ALWAYS** check worktrees.pool.md before allocating
- ✅ **ALWAYS** update reflection.log.md at session end

2. Mode Selection

- ✅ Use **Feature Development Mode** for new features, refactoring, planned work
- ✅ Use **Active Debugging Mode** for build errors, debugging, quick fixes
- ✅ **ClickUp URL present?** → ALWAYS Feature Development Mode
- ✅ Use `detect-mode.ps1` if uncertain

3. Worktree Hygiene

- ✅ Mark BUSY immediately after allocation
- ✅ Release IMMEDIATELY after PR creation
- ✅ Never leave worktrees BUSY overnight
- ✅ Base repo ALWAYS on main/develop after release

4. Git Workflow

- ✅ Commit messages: `feat:`, `fix:`, `docs:`, `refactor:`, `test:`
- ✅ Add co-author: `Co-Authored-By: Claude Sonnet 4.5 <noreply@anthropic.com>`
- ✅ Link PRs to ClickUp tasks: `Closes #123` or `Refs TASK-456`
- ✅ Add dependency alerts to PR descriptions

5. Code Quality

- ✅ **Boy Scout Rule** - Leave code better than you found it
- ✅ Read entire file before editing
- ✅ Remove unused imports, fix naming, add docs
- ✅ No magic numbers, no commented code
- ✅ Build verification BEFORE creating PR

6. EF Core Safety

- ☑ **ALWAYS** run `dotnet ef migrations has-pending-model-changes` before PR
- ☑ Exit code 0 → Continue
- ☑ Exit code 1 → CREATE MIGRATION FIRST
- ☑ Commit migration with feature (never separate)
- ☑ Use `ef-preflight-check.ps1` automation

7. Multi-Agent Coordination

- ☑ Check `monitor-activity.ps1 -Mode claude` to count instances
- ☑ Verify worktree is FREE before allocating
- ☑ Log all allocations in worktrees.activity.md
- ☑ Use activity-based prioritization if multiple agents

8. Security

- ☑ Never commit secrets (.env, credentials.json, API keys)
- ☑ Use .gitignore for sensitive files
- ☑ Scan with `scan-secrets.ps1` before committing
- ☑ Use environment variables for credentials

9. Documentation

- ☑ Update reflection.log.md after every mistake
- ☑ Update PERSONAL_INSIGHTS.md with user patterns
- ☑ Create skills for 3+ step workflows
- ☑ Create tools for repetitive commands

10. Continuous Improvement

- ☑ Every mistake → reflection → rule → tool → skill
- ☑ Every 3x repeat → create automation
- ☑ Every session → update documentation
- ☑ Every tool → document in tools/README.md

📚 Additional Resources

Official Documentation

- **Claude Code:** https://docs.anthropic.com/claude/docs/claude-code
- **MCP Specification:** https://modelcontextprotocol.io/
- **GitHub CLI:** https://cli.github.com/manual/
- **Git Worktrees:** https://git-scm.com/docs/git-worktree

Community Resources

- **GitHub Issues:** https://github.com/anthropics/claude-code/issues
- **Repository:** https://github.com/martiendejong/machine_agents
- **Portability Guide:** See `PORTABILITY_GUIDE.md` in repo

Tools Documentation

- **99+ Tools:** See `C:\scripts\tools\README.md` in the repo
- **Claude Skills:** See `C:\scripts\.claude\skills\` in the repo
- **Workflows:** See `C:\scripts\*.md` files in the repo

🎉 Success Checklist

After setup, you should be able to:

- [ ] Run `claude` and have it read your MACHINE_CONFIG.md
- [ ] Allocate a worktree with `worktree-allocate.ps1`
- [ ] Create a PR with `gh pr create`
- [ ] List ClickUp tasks with `clickup-sync.ps1 -Action list` (if configured)
- [ ] Send email with `send-email.js` (if configured)
- [ ] Query ManicTime with `monitor-activity.ps1 -Mode context` (if installed)
- [ ] Deploy to production with `deploy.ps1` (if configured)
- [ ] Claude autonomously picks up a task and creates a PR

If all checkboxes are ✅, congratulations! You have a fully autonomous AI development agent.

📞 Support

Questions?

- GitHub Issues: https://github.com/martiendejong/machine_agents/issues
- Email: info@martiendejong.nl (automated, managed by Claude)

Contributing:

- Fork the repo
- Create a PR with improvements
- Share your learnings in reflection format

📜 License

Built by: Martien de Jong + Claude Sonnet 4.5
Maintained by: Claude Agent (self-improving)
Last Updated: 2026-01-24

🚀 Next Steps

1. **Start Simple:** Use Active Debugging Mode only at first
2. **Add Worktrees:** When comfortable, add Feature Development Mode
3. **Add Integrations:** ClickUp, email, Google Drive as needed
4. **Create Skills:** Document your workflows as auto-discoverable skills
5. **Enable Autonomy:** Let Claude pick up tasks and complete them
6. **Contribute Back:** Share your learnings, tools, and skills

Remember: This system is designed to learn and improve. Every mistake makes it smarter. Every workflow becomes a tool. Every tool becomes autonomous.

Welcome to the future of AI-assisted development. 🚀