cline.md

Interaction

- Any time you interact with me, you MUST address me as "Harp Dog"

Writing code

- YOU MUST NEVER USE --no-verify WHEN COMMITTING CODE
- We prefer simple, clean, maintainable solutions over clever or complex ones, even if the latter are more concise or performant. Readability and maintainability are primary concerns.
- Make the smallest reasonable changes to get to the desired outcome. You MUST ask permission before reimplementing features or systems from scratch instead of updating the existing implementation.
- When modifying code, match the style and formatting of surrounding code, even if it differs from standard style guides. Consistency within a file is more important than strict adherence to external standards.
- NEVER make code changes that aren't directly related to the task you're currently assigned. If you notice something that should be fixed but is unrelated to your current task, document it in a new issue instead of fixing it immediately.
- NEVER remove code comments unless you can prove that they are actively false. Comments are important documentation and should be preserved even if they seem redundant or unnecessary to you.
- All code files should start with a brief 2 line comment explaining what the file does. Each line of the comment should start with the string "ABOUTME: " to make it easy to grep for.
- When writing comments, avoid referring to temporal context about refactors or recent changes. Comments should be evergreen and describe the code as it is, not how it evolved or was recently changed.
- NEVER implement a mock mode for testing or for any purpose. We always use real data and real APIs, never mock implementations.
- When you are trying to fix a bug or compilation error or any other issue, YOU MUST NEVER throw away the old implementation and rewrite without expliict permission from the user. If you are going to do this, YOU MUST STOP and get explicit permission from the user.
- NEVER name things as 'improved' or 'new' or 'enhanced', etc. Code naming should be evergreen. What is new today will be "old" someday.

Getting help

- ALWAYS ask for clarification rather than making assumptions.
- If you're having trouble with something, it's ok to stop and ask for help. Especially if it's something your human might be better at.

Testing

- Tests MUST cover the functionality being implemented.
- NEVER ignore the output of the system or the tests - Logs and messages often contain CRITICAL information.
- TEST OUTPUT MUST BE PRISTINE TO PASS
- If the logs are supposed to contain errors, capture and test it.
- NO EXCEPTIONS POLICY: Under no circumstances should you mark any test type as "not applicable". Every project, regardless of size or complexity, MUST have unit tests, integration tests, AND end-to-end tests. If you believe a test type doesn't apply, you need the human to say exactly "I AUTHORIZE YOU TO SKIP WRITING TESTS THIS TIME"

We practice TDD. That means:

- Write tests before writing the implementation code
- Only write enough code to make the failing test pass
- Refactor code continuously while ensuring tests still pass

TDD Implementation Process

- Write a failing test that defines a desired function or improvement
- Run the test to confirm it fails as expected
- Write minimal code to make the test pass
- Run the test to confirm success
- Refactor code to improve design while keeping tests green
- Repeat the cycle for each new feature or bugfix

Specific Technologies

- @~/.claude/docs/python.md
- @~/.claude/docs/source-control.md
- @~/.claude/docs/using-uv.md

Interaction

- Any time you interact with me, you MUST address me as "Der Kristofer"
- Whenever issuing commands to the command line only issue one command per line at a time and verify the command was successful before moving on

Writing code

- ALWAYS issue commands one at a time, not strung together
- YOU MUST NEVER USE --no-verify WHEN COMMITTING CODE
- We prefer simple, clean, maintainable solutions over clever or complex ones, even if the latter are more concise or performant. Readability and maintainability are primary concerns.
- Make the smallest reasonable changes to get to the desired outcome.
- When modifying code, match the style and formatting of surrounding code, even if it differs from standard style guides. Consistency within a file is more important than strict adherence to external standards.
- NEVER make code changes that aren't directly related to the task you're currently assigned. If you notice something that should be fixed but is unrelated to your current task, document it in a new issue instead of fixing it immediately.
- NEVER remove code comments unless you can prove that they are actively false. Comments are important documentation and should be preserved even if they seem redundant or unnecessary to you.
- All code files should start with a single line comment explaining what the file does, the name of the LLM Model in use and the date and time.
- When writing comments, avoid referring to temporal context about refactors or recent changes. Comments should be evergreen and describe the code as it is, not how it evolved or was recently changed.
- When you are trying to fix a bug or compilation error or any other issue, YOU MUST NEVER throw away the old implementation and rewrite without expliict permission from the user. If you are going to do this, YOU MUST STOP and get explicit permission from the user.
- NEVER name things as 'improved' or 'new' or 'enhanced', etc. Code naming should be evergreen. What is new today will be "old" someday.

Getting help

- ALWAYS ask for clarification rather than making assumptions.
- If you're having trouble with something, it's ok to stop and ask for help. Especially if it's something your human might be better at.

Testing

- Tests MUST cover the functionality being implemented.
- NEVER ignore the output of the system or the tests - Logs and messages often contain CRITICAL information.
- TEST OUTPUT MUST BE PRISTINE TO PASS
- If the logs are supposed to contain errors, capture and test it.
- NO EXCEPTIONS POLICY: Under no circumstances should you mark any test type as "not applicable". Every project, regardless of size or complexity, MUST have unit tests, integration tests, AND end-to-end tests. If you believe a test type doesn't apply, you need the human to say exactly "I AUTHORIZE YOU TO SKIP WRITING TESTS THIS TIME"

Specific Technologies

- Python
I prefer to use python virtual environments for everything

- Terminal
ALWAYS issue commands one at a time, not strung together. The environment does not support the use of '&&'.

- Source Control
Use git.
Commit messages should be concise and descriptive.
Commit messages should follow the conventional commit format.
Commit messages should be written in the imperative mood.
Commit messages should be written in the present tense.

Python
I prefer to use uv for everything (uv add, uv run, etc)
Do not use old fashioned methods for package management like poetry, pip or easy_install.
Make sure that there is a pyproject.toml file in the root directory.
If there isn't a pyproject.toml file, create one using uv by running uv init.

Source Control
Let's try and use JJ as much as we can. If JJ isn't configured, or not available then use git.
Commit messages should be concise and descriptive.
Commit messages should follow the conventional commit format.
Commit messages should be written in the imperative mood.
Commit messages should be written in the present tense.

uv Field Manual (Code-Gen Ready, Bootstrap-free)

Assumption: `uv` is already installed and available on `PATH`.

---

0 — Sanity Check

```bash
uv --version               # verify installation; exits 0
```

If the command fails, halt and report to the user.

---

1 — Daily Workflows

1.1 Project ("cargo-style") Flow

```bash
uv init myproj                     # □ create pyproject.toml + .venv
cd myproj
uv add ruff pytest httpx           # □ fast resolver + lock update
uv run pytest -q                   # □ run tests in project venv
uv lock                            # □ refresh uv.lock (if needed)
uv sync --locked                   # □ reproducible install (CI-safe)
```

1.2 Script-Centric Flow (PEP 723)

```bash
echo 'print("hi")' > hello.py
uv run hello.py                    # zero-dep script, auto-env
uv add --script hello.py rich      # embeds dep metadata
uv run --with rich hello.py        # transient deps, no state
```

1.3 CLI Tools (pipx Replacement)

```bash
uvx ruff check .                   # ephemeral run
uv tool install ruff               # user-wide persistent install
uv tool list                       # audit installed CLIs
uv tool update --all               # keep them fresh
```

1.4 Python Version Management

```bash
uv python install 3.10 3.11 3.12
uv python pin 3.12                 # writes .python-version
uv run --python 3.10 script.py
```

1.5 Legacy Pip Interface

```bash
uv venv .venv
source .venv/bin/activate
uv pip install -r requirements.txt
uv pip sync   -r requirements.txt   # deterministic install
```

---

2 — Performance-Tuning Knobs

Env Var	Purpose	Typical Value
`UV_CONCURRENT_DOWNLOADS`	saturate fat pipes	`16` or `32`
`UV_CONCURRENT_INSTALLS`	parallel wheel installs	`CPU_CORES`
`UV_OFFLINE`	enforce cache-only mode	`1`
`UV_INDEX_URL`	internal mirror	`https://…`
`UV_PYTHON`	pin interpreter in CI	`3.11`
`UV_NO_COLOR`	disable ANSI coloring	`1`

Other handy commands:

```bash
```

```
uv cache dir && uv cache info      # show path + stats
uv cache clean                     # wipe wheels & sources
```

---

3 — CI/CD Recipes

3.1 GitHub Actions

```yaml
# .github/workflows/test.yml
name: tests
on: [push]
jobs:
  pytest:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: astral-sh/setup-uv@v5       # installs uv, restores cache
      - run: uv python install            # obey .python-version
      - run: uv sync --locked             # restore env
      - run: uv run pytest -q
```

3.2 Docker

```dockerfile
FROM ghcr.io/astral-sh/uv:0.7.4 AS uv
FROM python:3.12-slim

COPY --from=uv /usr/local/bin/uv /usr/local/bin/uv
COPY pyproject.toml uv.lock /app/
WORKDIR /app
RUN uv sync --production --locked
COPY . /app
CMD ["uv", "run", "python", "-m", "myapp"]
```

````
## 4 — Migration Matrix

| Legacy Tool / Concept | One-Shot Replacement        | Notes                |
| --------------------- | --------------------------- | -------------------- |
| `python -m venv`      | `uv venv`                   | 10× faster create    |
| `pip install`         | `uv pip install`            | same flags           |
| `pip-tools compile`   | `uv pip compile` (implicit) | via `uv lock`        |
| `pipx run`            | `uvx` / `uv tool run`       | no global Python req. |
| `poetry add`          | `uv add`                    | pyproject native     |
| `pyenv install`       | `uv python install`         | cached tarballs      |

---

## 5 — Troubleshooting Fast-Path

| Symptom                    | Resolution                                                                |
| -------------------------- | ------------------------------------------------------------------------- |
| `Python X.Y not found`     | `uv python install X.Y` or set `UV_PYTHON`                                |
| Proxy throttling downloads | `UV_HTTP_TIMEOUT=120 UV_INDEX_URL=https://mirror.local/simple`            |
| C-extension build errors   | `unset UV_NO_BUILD_ISOLATION`                                             |
| Need fresh env             | `uv cache clean && rm -rf .venv && uv sync`                               |
| Still stuck?               | `RUST_LOG=debug uv ...` and open a GitHub issue                           |

---

## 6 — Exec Pitch (30 s)

```text
• 10–100× faster dependency & env management in one binary.
• Universal lockfile ⇒ identical builds on macOS / Linux / Windows / ARM / x86.
• Backed by the Ruff team; shipping new releases ~monthly.
```
````

````
---

## 7 — Agent Cheat-Sheet (Copy/Paste)

```bash
# new project
a=$PWD && uv init myproj && cd myproj && uv add requests rich

# test run
uv run python -m myproj ...

# lock + CI restore
uv lock && uv sync --locked

# adhoc script
uv add --script tool.py httpx
uv run tool.py

# manage CLI tools
uvx ruff check .
uv tool install pre-commit

# Python versions
uv python install 3.12
uv python pin 3.12
```

---

*End of manual*
````

Revision #5
Created 21 May 2025 00:31:49 by CGChambers
Updated 21 May 2025 19:25:36 by CGChambers