ChatGPT

실행 아키텍처 개요

이 시스템은 Python의 **Invoke** 기반 태스크 프레임워크와 PowerShell/Batch 스크립트를 결합한 형태로 동작합니다. 주요 통제 인터페이스는 `tasks.py` / `tasks_unix.txt`에 정의된 Invoke 태스크들로, 예를 들어 `invoke start`, `invoke end`, `invoke test` 등입니다. 각 태스크는 내부적으로 `scripts/runner.py`가 제공하는 `run_command` 함수(`subprocess.run(shell=False)` 래퍼)를 호출하여 실제 작업(예: Git 명령, Python 스크립트 실행 등)을 수행하고, 실패 시 에러 로그를 남기도록 설계되었습니다 [1]. 이렇게 실행된 모든 태스크 호출은 SQLite 기반 `usage.db`에 사용 이력이 기록되며 [2] [3], 이는 `scripts/usage_tracker.py`를 통해 수행됩니다.

PowerShell/Batch 실행기는 각 AI 에이전트(Codex, Gemini, 향후 Claude 등)를 구분하여 구동합니다. 예를 들어 `codex.ps1` 스크립트는 환경변수 `ACTIVE_AGENT`를 `codex`로 설정하고, 자동 녹화를 위한 `AI_REC_AUTO`도 설정한 뒤 실제 Codex CLI를 호출합니다 [4]. Gemin 에이전트용 `gemini.ps1`도 동일하게 `ACTIVE_AGENT='gemini'`를 설정해 실행됩니다. 또한, `codex-session.ps1` · `gemini-session.ps1` 같은 런처 스크립트를 사용하면 워크스페이스 루트로 이동, UTF-8 설정, `AI_REC_AUTO=1` 설정 등 세션 초기화가 자동으로 이루어집니다 [5]. 이 과정에서 `PYTHONIOENCODING=utf-8` 환경변수 고정을 권장하여(모든 Python 입출력 UTF-8) 인코딩 문제를 방지합니다 [6].

이처럼 Invoke 태스크와 PowerShell 런처가 결합된 아키텍처는 **윈도우 환경 친화적**으로 설계되어 있습니다. 전체 작업 흐름은 레포지토리 루트 디렉터리에서 실행되며, 각 실행기(Launch 스크립트)는 프로세스 단위로 `ACTIVE_AGENT` 환경변수를 분리해 여러 터미널에서 병행 실행할 수 있도록 지원합니다 [7].

태스크 정의 및 실행 흐름

`tasks.py` / `tasks_unix.txt`에는 수많은 사용자 명령어(Invoke 태스크)가 정의되어 있습니다. 예를 들어 **에이전트 관리 태스크**로 `invoke agent.status`, `invoke agent.set`가 있으며, 현재 활성 에이전트를 조회·설정합니다 [8]. 메시징과 허브 관련해서는 `invoke agent.msg`, `agent.inbox`, `hub.send`, `hub.claim`, `hub.complete` 등이 있어 에이전트 간 간단한 메시지 큐 워크플로우를 제공합니다.

편집 제안 워크플로우도 구축되어 있습니다. `invoke edits.capture --file <file>`은 대상 파일의 초기 스냅샷을 저장하여 수정용 초안을 생성하고, `invoke edits.propose --file <file>`로 변경된 초안을 제안하며, `invoke edits.apply`로 제안된 편집을 파일에 적용하는 흐름입니다. 실제로 이들 태스크는 `scripts/edits_manager.py`를 호출하여 작업을 처리하며, `invoke edits.diff`로 변경 내용을 미리볼 수도 있습니다 [9] [10].

세션 관리를 위해 `invoke start`와 `invoke end` 태스크가 있습니다. `invoke start`는 세션 시작 시 시스템 상태 점검과 준비 작업을 담당합니다. 예를 들어 Git 상태와 태스크(HUB.md) 현황을 요약 출력하고, 필요시 Doctor 검사와 컨텍스트 인덱스 생성을 수행하며, 세션 시작 로그를 기록합니다 [11]. 반면 `invoke end`는 세션 종료 시점에 **WIP 커밋**(`invoke wip`)과 함께 `docs/HUB.md`의 `__lastSession__` 블록을 업데이트하고 변경사항을 커밋합니다. 종료 시 `agent_messages.mark_read()`로 인박스를 처리하고, 비동기적으로 세션종료 사용 이력을 기록합니다 [12]. 이외에도 `invoke status`, `invoke wip`, `invoke test`, `invoke clean_cli`, `invoke doctor` 등의 기본 운영 보조 태스크가 구현되어 있습니다.

헬스체크는 `invoke doctor`가 담당하며, 시스템 의존성 및 환경 설정을 검사해 출력합니다(내부적으로 `scripts/doctor.py` 호출). 또한 세션 도중 오류 발생 시 3회 재시도 제한 같은 정책을 적용하여 안정성을 높입니다(Plan_O3 참조). **비상 래퍼** 기능도 도입되어 Codex 호출을 안전하게 제어합니다. `.agents/emergency.json`에 `enabled: true` 설정 시, PowerShell 프로필을 통해 `codex` 호출이 `tools/codex_emergency.ps1`으로 래핑됩니다 [13]. 이는 최대 토큰수, 속도(`rps`) 등을 환경변수(`CODEX_*`)로 주입하여 제어 가능하며, 비활성화 시 해당 래퍼를 우회합니다.

에이전트 연동

에이전트 연동은 크게 **CLI 실행기**와 **환경 변수**를 통한 구분으로 이루어집니다. 위에서 언급한 `codex.ps1`/`gemini.ps1` 등이 실질적인 에이전트 CLI 호출을 담당하며, 각각 실행 시 `ACTIVE_AGENT`를 설정하여 어떤 에이전트로 동작할지 구분합니다 [4]. `claude.cmd` 스크립트도 레포 내부에서 Claude UI(툴)를 실행하도록 래핑되어 있는데, 개발 단계에서는 아직 유료 API 불가로 활용이 제한된 상태입니다. 환경 변수 `ACTIVE_AGENT`는 Invoke 태스크 내부에서도 참조되어(예: `agent_manager.get_active_agent()`) 기본 동작을 결정합니다.

또한 녹화 기능(`ai-rec`)이 에이전트 세션과 연결됩니다. `AI_REC_AUTO=1` 환경변수가 설정된 상태로 세션을 시작하면 자동으로 터미널 전체 출력을 텍스트로 기록하는 Start-Transcript가 작동합니다. 녹화된 로그는 `<repo>/terminal_logs/YYYY-MM-DD/session_HHMMSS__agent-<이름>__term-<터미널유형>__pid-<PID>.txt` 형태로 저장됩니다 [14]. 수동으로도 동일 세션에서 `.\ai-rec-start.ps1`/`ai-rec-stop.ps1`로 녹화를 토글할 수 있습니다.

환경 변수는 위의 `ACTIVE_AGENT` 외에도 동작에 영향을 줍니다. 예를 들어 `PYTHONIOENCODING=utf-8`을 항상 유지하여 Python CLI 호출 시 UTF-8 인코딩을 강제합니다 [6]. `AI_REC_FLAT=1`은 날짜별 디렉터리 없이 단일 폴더에 로그를 저장하게 합니다. 다중 터미널에서 병행 실행 시에도 각 프로세스별로 `ACTIVE_AGENT`를 다르게 설정하여 중복을 방지할 수 있으며, 이때 데이터베이스도 WAL 모드로 안정적 동시 기록이 보장됩니다 [7].

로그 및 기록 관리

실행 로그와 기록 관리는 `scripts/runner.py`와 `scripts/usage_tracker.py`를 중심으로 처리됩니다. `runner.py`는 Invoke 태스크 호출 시 subprocess 실행 결과(성공, 실패 코드 등)를 캡처하여 **명령어별 로그**를 남기며, 오류 발생 시 별도의 `command_error` 이벤트를 `usage.db`에 기록합니다 [1] [3]. 예를 들어, `invoke start`가 호출되면 즉시 `log_usage('session','start',...)`로 사용 이력을 남기고 [11], `invoke end`에서는 세션 종료를 비동기 기록합니다 [12]. 요약하자면, 모든 태스크 실행은 SQLite `usage.db`로 축적되어 분석 및 감사용 로그베이스로 활용됩니다 [2].

세션별 대화 기록(terminal transcript)은 위에서 언급한대로 `terminal_logs/` 하위에 저장되며, 터미널 종류에 따라 파일명이 달라집니다(예: Windows Terminal의 `WT_SESSION`, VSCode 터미널은 `VSCODE_PID`, 나머지는 `ConsoleHost`) [14]. 또한 Git 커밋 관련 유틸(예: `invoke review`, `invoke git.push`)을 통해 소스 변경 이력 관리를 지원하며, 사전 편집(diff) 작업도 모두 Git 트리 상태로 유지됩니다. 전역적으로 삭제 금지 대상 파일(`.no_delete_list`)을 지정하여 중요한 문서 보호도 병행합니다.

장점과 한계

이런 실행기/태스크 기반 아키텍처의 **장점**은 일관된 워크플로우와 자동화입니다. 사용자 명령을 Invoke 태스크로 추상화하여 일관된 명령어 인터페이스를 제공하고, 각 단계(시작/종료/검사 등)마다 표준화된 절차가 적용됩니다. PowerShell 세션 자동 설정(`ps7_utf8_profile_sample.ps1`, `codex-session.ps1` 등)으로 Windows

환경 최적화가 이뤄졌으며, 스크립트 단위 격리 덕분에 오류가 다른 세션에 전파되지 않습니다. 사용 이력과 로그가 중앙 DB에 쌓이기 때문에 분석·감사가 가능하고, 허브 메시징이나 편집 제안 같은 협업 기능도 텍스트 파일 기반으로 지원됩니다.

한계로는 복잡도가 증가한다는 점을 들 수 있습니다. 여러 스크립트와 태스크가 겹쳐 있어 개별 구성요소가 중복되거나 관리 포인트가 많습니다. 특히 PowerShell/Unix용 스크립트가 병행되면서 파일이 두 벌로 유지되고, Windows에 종속적인 설정(인코딩, 세션 프로필 등)이 산재해 있습니다. 향후 새로운 에이전트(Claude)나 타 OS 지원이 추가되면, 이러한 플랫폼 의존성을 제거하고 일관성 있게 유지보수하는 데 추가 노력이 필요합니다. 또한 모든 오류를 기록하고 3회 재시도 등 제어 로직이 있지만, 복합 작업에서의 실패 복구 등은 아직 수동 개입에 의존할 수 있습니다.

요약 및 사용 맥락

이 분석은 **Claude Code** 등 새로운 에이전트가 실제 워크스페이스를 가동하기 전에, 현재 세션이 어떻게 실행되고 관리되는지 빠르게 파악하기 위한 것입니다. 멀티 에이전트 워크플로우는 Invoke 태스크(`tasks.py`)와 PowerShell 런처(`codex.ps1`, `gemini.ps1` 등), 그리고 실행기 스크립트(`runner.py`, `usage_tracker.py` 등)로 구성됩니다. 사용자는 `invoke <task>` 명령과 환경변수 설정으로 에이전트를 전환하고 작업할 수 있으며, 모든 세션 출력을 `terminal_logs/`에 기록하여 추후 검토가 가능합니다 [15] [16]. 이런 구조를 이해하면, 각 세션 실행 흐름과 책임 범위를 명확히 파악하고 문제 발생 시 대응할 수 있습니다.

출처: 레포지토리 내 `scripts/` 디렉터리 및 `tasks.py` 정의, AGENTS.md(운영 지침) 등 [4] [8] [17] [16].

---

[1] [3] [16] [P0]Plan_O3.md
https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/scratchpad/2_proposals_and_plans/[P0]Plan_O3.md

[2] [5] [6] [7] [13] [14] [15] [17] AGENTS.md
https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/AGENTS.md

[4] codex.ps1
https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/codex.ps1

[8] [9] [10] [11] [12] tasks_unix.txt
https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/tasks_unix.txt