인공지능을 활용한 코딩 및 계산 자동화

황준오
PhD Candidate, Data Science Lab

Dept. of Physics Education, Seoul National University
wnsdh10@snu.ac.kr

차례

들어가며

맛보기

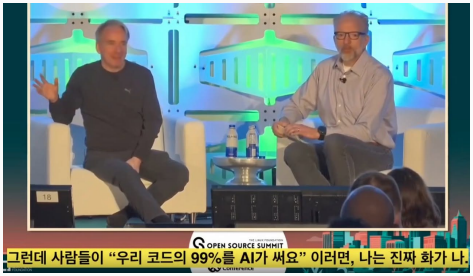
기본 실습

심화 실습과 업무 자동화

들어가며

리누스 토발즈의 Open Source Summit 기조강연

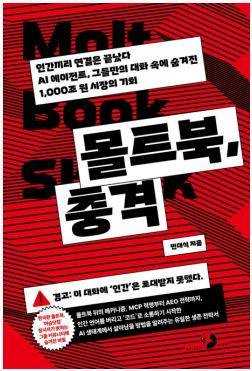
- AI-assisted coding에 호의적인 리누스 토발즈
- 어셈블러·컴파일러의 등장은 코딩 효율을 수천 ~ 수만 배 향상
- AI agentic coding은 기껏해야 10 ~ 100배 수준 — 상대적으로 미미한 혁신
- 누구도 “내 코드는 100% 컴파일러가 써줬어요”라고 말하지 않는다



Open Source Summit keynote (YouTube)

우후죽순 쏟아지는 새 기술들 — 상당수는 금방 deprecated 된다

- LangChain, OpenClaw, Moltbook, MCP, 프롬프트 엔지니어링, 하네스 엔지니어링…
- 실제로는 그렇게 중요하지 않은 것이 많지만, 써보기 전까지는 유용성을 판단하기 어렵다
- 금방 변하는 도구를 학습하느니, 오히려 직접 만들어서 쓰는 것이 더 유리
- 코딩이 돈과 provider/platform에 의존하게 되는 문제



정말 미니멀하게 필요한 것은?

코딩 에이전트

- Codex, Claude Code, Gemini CLI
- OpenCode, Aider …

에디터 / IDE

- VS Code, Cursor, Windsurf, Anigravity
- Jupyter Notebook, Vim, 메모장 …

오늘 실습은 어느 것을 써도 무방 — OpenCode를 접해보고 VS Code·Cursor 인터페이스에 친숙해지는 기회로.

AI 덕분에, 하면 좋지만 귀찮아서 미루던 것들을 할 수 있게 됨

- Git을 활용한 버전 관리와 문서화
- 추상화, 객체지향 프로그래밍 (OOP)
- logging / config 을 통한 체계적인 실험 관리 및 추적
- 클린 코드, 예외 처리
- Jupyter Notebook를 벗어나기

맛보기

OpenCode 는 무엇이고 왜 좋은가?

- 쉽고 빠른 설치와 사용, 가볍지만 깔끔한 인터페이스
- 계정조차 필요 없는 넉넉한 무료 사용량, 거의 무제한에 가까운 저렴한 구독제
- 완전한 오픈소스, provider-agnostic AI agent
- `opencode web`, `opencode run` 명령어 제공
- 공식 홈페이지: opencode.ai

```
# Linux / Mac
curl -fsSL https://opencode.ai/install | bash
# Windows (PowerShell, scoop 설치 후 / 터미널 재시작 권장)
scoop install opencode
```

OpenRouter — 모델 비교 및 provider-agnostic usage

- 하나의 인터페이스로 여러 provider(OpenAI, Anthropic, Google, Meta …)의 모델을 비교·전환
- 특정 업체에 종속되지 않는 provider-agnostic 사용
- 공식 홈페이지: openrouter.ai

Codex · Claude Code · Cursor 비교 시연

- 세 도구 모두 desktop 버전과 cli 버전을 제공
- 거의 표준화된 5시간 / 1주 / 1달 단위 quota
- 요금제는 비슷하나, pay-as-you-go보다 quota 구독제가 체감상 수배 저렴
- Cursor + OpenRouter / OpenCode 조합이면 거의 무료로 사용 가능
- rule · skill · command · plan · subagent · goal 작업 흐름도 표준화
- 세션이 길어지면 메모리 누수 가능 — 공용 서버에서는 잘 종료하기

VS Code 를 fork 한 개발환경의 특징

- 압도적인 Extension 생태계 (SSH, 마크다운, 구문 강조, Git 자동화 등)
- 한 윈도우를 벗어날 필요가 없음
- Python interactive window를 사용할 수 있다
- 하지만 조금 무겁다

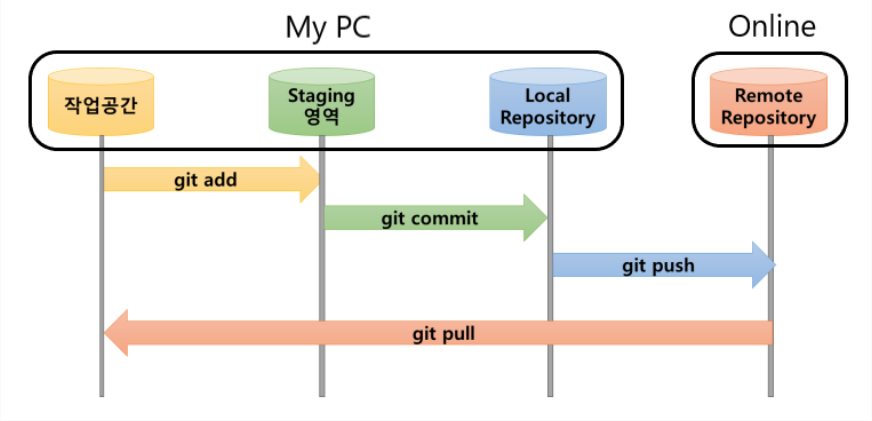
휴식 + 각자 원하는 개발환경 / 도구 설치 시간.

기본 실습

Agentic coding 시대의 새로운 표준: AGENTS.md

- 잠깐의 유행이 아닌 정말로 표준화되어 가고 있으며, 실효성도 높음
- README는 사람이, AGENTS는 기계가 읽는다 (CLAUDE.md는 클로드가 읽는다)
- README와 마찬가지로 폴더별로 설정 가능
- 여러 환경에서 작업할 경우, 환경 정보는 별도 파일에 넣고 `.gitignore` 처리 권장

Git 을 통한 버전 관리의 필수화



특정 지점 체크포인트로 되돌리려면 git 세팅 필요 (Cursor는 자동 체크포인트 제공).

Git 주요 명령어 — 초기 설정과 커밋

```
git init                          # 현재 폴더를 git 저장소로 초기화
git config --global user.name  "이름"
git config --global user.email "email"    # 작성자 정보 (최초 1회)
git config --list                 # 현재 설정 확인 (--global = 전역만)
.gitignore                        # 제외 파일 (__pycache__/, .env, *.pyc)
git add .                         # 변경 파일 스테이징 (특정: git add 파일명)
git commit -m "메시지"             # 스테이징 내용을 체크포인트로 저장
```

Git 주요 명령어 — 상태 · 되돌리기 · 원격

```
git status   /   git log --oneline       # 상태 · 커밋 이력 확인
git checkout -- 파일명  /  git restore 파일명  # 마지막 커밋으로 되돌리기
git remote add origin URL
git push -u origin main                  # 원격 연동 및 업로드

# 로그 옵션
git log --graph --oneline --all -n 20
#   --graph 분기/머지   --decorate 이름   --author 작성자 필터
```

자연어로 명령하는 시대이므로 안 외워도 됨.

Agent 권한 설정 & 간단한 실습

Agent 의 권한 설정

- 워크스페이스 바깥의 요소를 건드리지 않도록 주의
- permission 설정으로 접근 범위 제어 가능

간단한 실습

- 간단한 웹사이트, 파이썬 실험, LaTeX 빌드 연습

원격서버 SSH 연결과 HPC 클러스터 노드 주의

- Jupyter Hub 노드 $\neq$ SSH 노드 — SSH는 보통 고정된 진입점 사용
- `hostname` 또는 `echo $HOSTNAME` 으로 실제 계산에 쓸 노드 확인
- 기본 노드에 SSH 접속 후 `ssh {hostname}` 으로 실제 노드 접속 (막아둔 경우도 있음)
- Remote SSH 상태에서 AI agent를 실행하면 내 PC가 아닌 서버에서 실행됨 — 익스텐션 설치도 서버에 적용

Jupyter Notebook (*.ipynb) 을 어떻게 다룰 것인가?

- REPL(Read-Eval-Print Loop) 환경은 앞으로도 여전히 유용함
- 그러나 notebook은 거대한 JSON 파일 — 모든 게 직렬화되어 AI가 읽을 때 매우 비효율적
- 해결책: VS Code에서 사용 (Cursor는 Anthropic 모델 한정 제한적 지원)
- 차선책: `notebook-mcp` · Mito 익스텐션 / 그냥 지시 / jupytext 동기화
- 좋은 대안: Python interactive window 사용

```
pip install --user jupytext
jupytext --to notebook notebook.ipynb
jupytext --set-formats ipynb,py:percent notebook.ipynb
jupytext --sync notebook.ipynb
```

심화 실습과 업무 자동화

주기적 작업 자동화 & opencode run 활용

- Desktop agents가 제공하는 주기적인 작업 자동화 기능 활용
- 일반 코드 · 슈도 코드 · 자연어 코드를 결합하는 방법
- Python subprocess로 터미널 명령 실행 → `opencode run`을 파이썬 프로세스 안에 담기 (Windows는 `shell=True`)
- Semaphore로 병렬처리 (serve + `--attach`로 같은 서버에서 처리)

예) 메타분석 연구 자동화

```
질문 입력
연구문제 형식화 및 분석 메소드 선정
관련 논문 검색 & 리스트업 후 top N 선정

for 논문 in 논문 리스트 with Semaphore(k):
    PDF 다운로드
    주요 통계량 및 결과 저장

최종 메타분석
```

config (YAML · TOML · dataclass) 과 logging

```python
from dataclasses import dataclass
import yaml

@dataclass
class TrainConfig:
    seed: int = 42
    model: str = "resnet18"
    lr: float = 1e-3
    batch_size: int = 32
    epochs: int = 10
    data_dir: str = "data/"

with open("config.yaml") as f:
    data = yaml.safe_load(f)
cfg = TrainConfig(**data)
```

config.yaml

```yaml
seed: 42
model: resnet18
lr: 0.001
batch_size: 64
epochs: 20
data_dir: data/cifar10
```

실행 결과

```
TrainConfig(seed=42,
  model='resnet18',
  lr=0.001, ...)
```

핵심 포인트

- 코딩이 돈과 provider/platform에 의존하게 되는 문제를 경계
- 공용 서버에서는 세션을 잘 종료하기 (메모리 누수)
- Notebook을 벗어나 Python interactive window 사용
- Remote SSH 상태의 AI agent는 내 PC가 아닌 서버에서 실행됨
- 자연어로 명령하는 시대 — 명령어는 안 외워도 됨

황준오 · Data Science Lab, SNU · wnsdh10@snu.ac.kr