ІНСТРУМЕНТИ · CLAUDE

Claude Code: setup і реальні use-cases 2026

Як налаштувати Claude Code за 15 хвилин, які MCP-сервери ставити одразу, 8 практичних use-case'ів від рефакторингу до side-projects.

~10 ХВ

GOSHA.LIVE

ЩО ВСЕРЕДИНІ

- → Setup за 15 хвилин з нуля
- → 10 must-have MCP-серверів
- → 8 use-cases з реальними prompt-патернами
- → Як не злити бюджет на токенах

Claude Code — це CLI-агент який живе у твоїй терміналі. Не autocomplete у редакторі, а **повноцінний пара-розробник**: читає твій код, робить зміни, запускає тести, дебажить. Я використовую щодня замість Cursor і вважаю — для серйозних проєктів цей вибір кращий.

Тут — setup за 15 хвилин і use-cases які ти повинен спробувати першими.

Setup

```
# Install (потрібен Node 18+)
npm install -g @anthropic-ai/claude-code

# Запуск у проєкті
cd ~/your-project
claude
```

Перший раз попросить API key — з console.anthropic.com. Підписка Pro/Max включає API кредити.

Створи `.claude/settings.json` для проєкту:

```
{
  "permissions": {
    "allow": ["Bash(npm:*)", "Bash(pnpm:*)", "Bash(git status)", "Bash(git log)"],
    "deny": ["Bash(rm -rf:*)"]
  },
  "env": {
    "EDITOR": "code"
  }
}
```

> **Перші команди**
>
> `/init` — створює CLAUDE.md з документацією твого проєкту (auto-generated). `/config` — налаштовує модель, theme, MCP. `/permissions` — переглядає що Claude може робити без підтвердження.

MCP-сервери які поставити одразу

MCP (Model Context Protocol) — спосіб давати Claude Code додаткові tools. Топ-10:

1. **github** — interaction з PR/issues/repos
2. **filesystem** — extended file operations
3. **postgres / sqlite** — query БД напряму з чату
4. **playwright** — automate browser для testing
5. **fetch** — HTTP requests до API
6. **context7** — актуальна документація бібліотек
7. **slack** — повідомлення в Slack
8. **linear** — task management
9. **memory** — persistent memory між сесіями
10. **sequential-thinking** — для складних задач

Install через:

```
claude mcp add github --command "npx @modelcontextprotocol/server-github"
```

Use-case 1: Refactor великого файлу

```
Подивись на src/components/CheckoutFlow.tsx. Він на 800 рядків.
Розбий на логічні підкомпоненти зберігаючи функціональність:

1. Знайди natural boundaries (UI sections, hooks, helpers)
2. Витягни кожну частину у окремий файл
3. Перевір що типи передаються правильно
4. Запусти tests після — переконайся що нічого не зламано

План спочатку, потім підтвердження від мене, потім виконання.
```

Use-case 2: Debug складного бага

```
Я бачу що при логіні юзер іноді отримує session token але не залогінений
у UI. Stack: Next.js 15 + Lucia + cookies.

Знайди root cause:
1. Прочитай auth-related код
2. Знайди де відбувається mismatch між server і client state
3. Reprодьюся локально (інструкції які я виконаю)
4. Запропонуй fix з поясненням
```

Чому Claude Code краще для debug

Cursor дає suggestions inline. Claude Code читає **весь контекст проєкту**, бачить як файли пов'язані, може запускати тести і дивитись output. Це інший рівень debugging-партнерства.

Use-case 3: Setup нового проєкту

```
Я хочу запустити новий side-project: SaaS для [опис].
Stack: Next.js 15, Postgres, Stripe.

Створи:
1. Init проєкту з Tailwind, Drizzle, ESLint, Prettier
2. Структуру папок (app, components, lib, db, server)
3. Базові routes (landing, /app, /api/auth)
4. Setup Stripe з test mode
5. Docker compose для local Postgres
6. README з instructions

Не вкладай у все — тільки скелет. Деталі допишу сам.
```

Use-case 4: Generate tests

```
Покрий unit-tests все що в src/lib/billing/. Vitest, AAA-стиль.

Для кожної функції:
- Happy path
- Edge cases (empty input, нульові значення)
- Error states (throws)
- Boundary conditions

Запусти tests після генерації — переконайся що проходять.
```

Use-case 5: PR review

```
Я зробив гілку feature/new-checkout. Зроби PR review:

1. git diff main..HEAD — переглянь зміни
2. Знайди potential bugs (off-by-one, null handling, race conditions)
3. Security review (auth, input validation, secrets)
4. Performance issues
5. Code style consistency з рештою проєкту

Виведи у форматі GitHub comment'ів — буду копіювати в PR.
```

Use-case 6: Migration / refactor (великий)

```
Я хочу мігрувати з Drizzle на Prisma. Зроби план:

1. Аналіз поточних schemas / queries
2. Mapping Drizzle → Prisma
3. Sequence migration (які файли в якому порядку)
4. Risk зон (де можуть бути runtime issues)
5. Testing strategy (як перевірити що нічого не зламалось)

План спочатку, виконання — крок за кроком з моїм підтвердженням.
```

Use-case 7: Documentation

```
Згенеруй README.md для цього проєкту:

- Опис проєкту (1 параграф)
- Tech stack
- Setup instructions (з якого моменту нового developer'а почати)
- Project structure (tree з поясненнями)
- Available scripts
- Deployment

Стиль як у Vercel/Next.js README — професійно, без води.
```

Use-case 8: Side-project automation

```
Я хочу запустити простий side-project: AI-боту для translation [конкретні мови].

Створи MVP за 1 годину:
- CLI tool який бере текст і повертає переклад
- Використовує Claude API з prompt caching
- Зберігає історію перекладів у sqlite
- Має basic config через env vars

Просто і робоче. Деталізую пізніше.
```

Як не злити бюджет на токенах

- Використовуй `claude --resume` щоб не починати з нуля - Налаштуй `model: "claude-haiku-4-5"` для дешевших задач, opus — тільки для складних - Великі рефактори — спочатку `/compact` старий контекст - Перевіряй cost у `claude config` періодично

Що далі

У повному PDF — детальні setup-чек-листи для 5 stack'ів (Next.js, Python, Go, Rust, mobile), 20+ готових prompt-патернів, гайд по MCP-серверах з прикладами використання. Завантажуй і прокачай свій dev-workflow.
