# Agent-ready Repository


- 1. Cos'è un Agent-ready Repository
  - Definizione di Agent-ready Repository
- 2. Struttura del repository
  - Struttura di cartelle e file
- 3. AGENTS.md e CLAUDE.md
  - Scrivere AGENTS.md e CLAUDE.md
- 4. Configurazione Claude Code
  - Permessi, comandi, skill e hook
- 5. Configurazione ChatGPT / Codex
  - Codex e ChatGPT
- 6. Procedura di adozione
  - Procedura di adozione passo per passo
- 7. Esempio: SoluzionaEvento
  - Creazione della struttura di SoluzionaEvento

# 1. Cos'è un Agent-ready Repository

Definizione, principi e componenti di un repository pronto per gli agenti AI.

# Definizione di Agent-ready Repository

## ? Definizione

Un **Agent-ready Repository** è un repository strutturato in modo che un agente AI di coding (Claude Code, Codex/ChatGPT, Cursor e simili) possa entrarci, capire il progetto ed eseguire task utili con pochissima guida umana.

Non dipende da un singolo file o strumento: è una proprietà strutturale dell'intero progetto.

## ? Idea di fondo

Un collega umano può chiedere «come lo avvio in locale?». Un agente no: la risposta deve esistere **nel repository stesso**. Il repo deve quindi essere comprensibile, eseguibile, verificabile e sicuro.




## ? I sei livelli

| Livello | Domanda a cui risponde | Artefatti tipici |
|---|---|---|

| | | |
|---|---|---|
| □□ Contesto | Come capisce il progetto e i suoi confini? | AGENTS.md, CLAUDE.md, docs/ |
| □□ Intento | Cosa deve fare il task, cosa è fuori scope, cosa significa «fatto»? | specifiche, definition of done |
| ⚙ Esecuzione | Come si installa e si avvia? | scripts/setup.ps1, versioni dichiarate |
| □ Verifica | Come controlla il proprio lavoro? | test, linter, pre-commit, CI |
| □□ Governance | Cosa può o non può toccare? | permessi, cartelle vietate, revisione umana |
| □□ Feedback | Come migliora nel tempo? | errori ricorrenti trasformati in regole |

**ℹ** *Non è uno standard formale: è un concetto emergente, con definizioni leggermente diverse a seconda della fonte.*

# 2. Struttura del repository

Albero di cartelle e file consigliato.

# Struttura di cartelle e file

## ? Struttura consigliata

```
SoluzionaEvento/
    AGENTS.md  — istruzioni (fonte unica)
    CLAUDE.md  — contiene: @AGENTS.md
    README.md · .gitignore · .env.example
    docs/  — architecture.md, conventions.md
    .claude/  — settings.json, commands/, skills/
    scripts/  — setup.ps1, test.ps1
    .github/workflows/  — ci.yml
    src/  e  tests/
```

## ?? A cosa serve ogni elemento

- **AGENTS.md**: fonte unica delle istruzioni, letta da Codex e dagli altri agenti.
- **CLAUDE.md**: letto da Claude Code; importa AGENTS.md con la riga `@AGENTS.md`.
- **docs/**: architettura e convenzioni, per non gonfiare AGENTS.md.
- **.claude/**: permessi, comandi personalizzati e skill (solo Claude Code).
- ⚙ **scripts/**: setup e test con un solo comando.
- **.github/workflows/**: la stessa verifica che usa l'agente, eseguita in CI.

## ? Quattro requisiti che contano

1. **Setup riproducibile**: un solo comando installa e avvia tutto, con versioni dichiarate.
2. **Verifica automatica**: test, linter e build veloci e deterministici.
3. **Confini chiari**: cartelle vietate e azioni che richiedono conferma.
4. **Segreti fuori dal repo**: usare `.env.example`, mai credenziali committate.

# 3. AGENTS.md e CLAUDE.md

Come scrivere le istruzioni e condividerle tra Claude e ChatGPT/Codex.

# Scrivere AGENTS.md e CLAUDE.md

## ? Principio: una sola fonte di verità

Si scrive tutto in `AGENTS.md` e si fa puntare `CLAUDE.md` a quel file.




## ? CLAUDE.md

Contenuto del file (una sola riga):

```
@AGENTS.md
```

In alternativa, un symlink: `ln -s AGENTS.md CLAUDE.md` (su Windows richiede privilegi o modalità sviluppatore: l'import è più semplice).

## ? Esempio di AGENTS.md

```
# Progetto: SoluzionaEvento


## Stack
- (da compilare: linguaggio, framework, versioni)
```

```
## Comandi
- Setup: `scripts/setup.ps1`
- Test: `scripts/test.ps1`
- Prima di ogni commit build e test devono passare.

## Struttura
- src/   codice applicativo
- tests/ specchio di src/
- docs/  architettura e convenzioni

## Convenzioni
- Modifiche mirate, niente riscritture complete di file
- Query SQL sempre parametrizzate
- Nessuna logica di business nei componenti UI

## Da non toccare
- migrations/ (solo su richiesta esplicita)
- file di configurazione di produzione, segreti, connection string

## Definition of done
- Build e test verdi, nessun warning nuovo, documentazione aggiornata
```

# ?? Regole di scrittura

- □ Breve, concreto e verificabile (sotto le 100–200 righe).
- □ Scrivere solo ciò che l'agente non può dedurre dal codice.
- ➡ Spostare i dettagli in `docs/`.
- □□ Ogni errore ricorrente dell'agente diventa una riga nuova.

# 4. Configurazione Claude Code

Permessi, comandi, skill e hook.

# Permessi, comandi, skill e hook

## ?? Permessi in `.claude/settings.json`

Definisce cosa Claude Code può fare senza chiedere e cosa non deve mai toccare.

```
{
  "permissions": {
    "allow": [
      "Bash(dotnet build:*)",
      "Bash(dotnet test:*)",
      "Bash(dotnet format:*)"
    ],
    "deny": [
      "Read(./.env)",
      "Edit(./migrations/**)",
      "Edit(./appsettings.Production.json)"
    ]
  },
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [ { "type": "command", "command": "dotnet format --verify-no-changes" } ]
      }
    ]
  }
}
```

⚠ Gli esempi sono indicativi: adattare comandi e percorsi al progetto e verificare la sintassi aggiornata nella documentazione ufficiale di Claude Code.

# ?? Comandi personalizzati — `.claude/commands/`

Un file markdown per ogni prompt ripetuto. Esempio `review.md`, richiamabile con `/review`:

```
Esegui una code review delle modifiche non ancora committate.
Controlla: convenzioni in docs/conventions.md, query SQL parametrizzate,
test mancanti. Elenca i problemi per gravità, senza modificare file.
```

# ? Skill — `.claude/skills/`

Procedure a più passaggi. Esempio `.claude/skills/nuovo-modulo/SKILL.md`:

```
---
name: nuovo-modulo
description: Aggiunge un nuovo modulo seguendo la struttura e le convenzioni del progetto.
---
1. Crea la cartella in src/ seguendo docs/architecture.md
2. Aggiungi i test in tests/
3. Esegui scripts/test.ps1 e correggi finché è verde
```

# ? Hook

Eseguono comandi in automatico (formattazione, test) dopo le modifiche: vedi la sezione `hooks` nell'esempio sopra.

# ? CLAUDE.md nelle sottocartelle

Un `CLAUDE.md` in una sottocartella viene caricato solo quando l'agente lavora lì: utile nei monorepo.

# 5. Configurazione ChatGPT / Codex

Uso di AGENTS.md e contesto in chat.

# Codex e ChatGPT

## ? Codex e agenti OpenAI

- □□ Leggono `AGENTS.md` alla radice del repository.
- □□ Un `AGENTS.md` può stare anche nelle sottocartelle: vale quello più vicino ai file modificati.
- □□ Sandbox e livello di approvazione si impostano nelle opzioni dello strumento (non nel repo).

## ? ChatGPT in chat (senza agente)

Non legge il repository da solo: va fornito il contesto.

1. Caricare `AGENTS.md` e i file rilevanti.
2. Tenere `docs/architecture.md` compatto, così entra nel contesto.
3. Indicare i comandi di build/test e la definition of done.

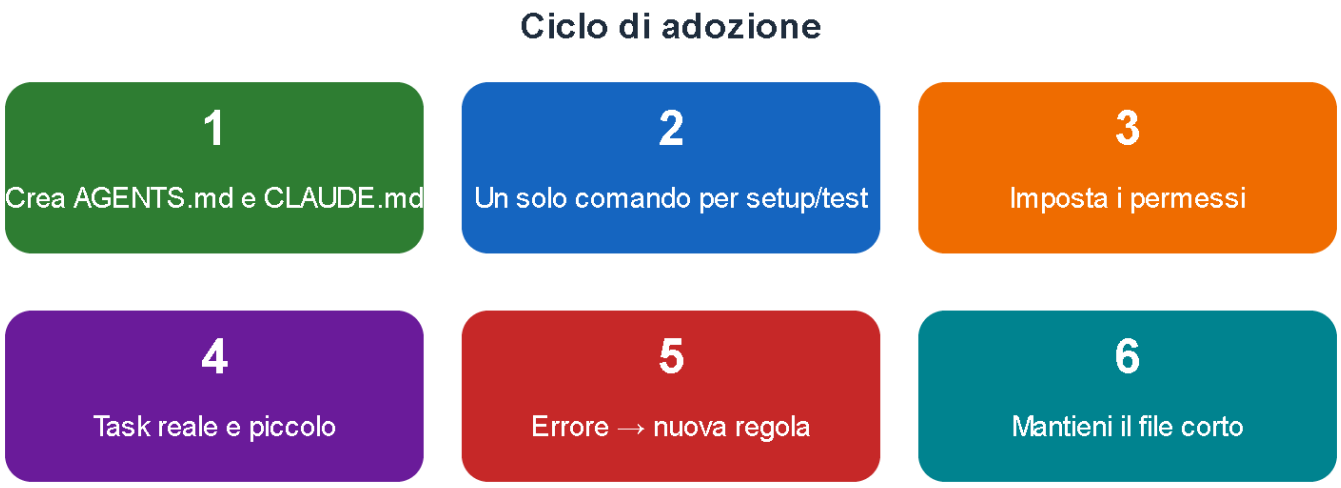
**ℹ** Le funzioni degli strumenti evolvono in fretta: confermare i dettagli sulla documentazione ufficiale.

# 6. Procedura di adozione

Passi operativi e miglioramento continuo.

# Procedura di adozione passo per passo

## ? Passi operativi




1. ◫ Creare `AGENTS.md` e `CLAUDE.md` (con `@AGENTS.md`).
2. ⚙ Verificare che build e test partano con un solo comando.
3. ◫ Aggiungere i permessi in `.claude/settings.json`.
4. ◫ Assegnare all'agente un task reale e piccolo e osservare dove sbaglia.
5. ◫ **Ogni errore ricorrente diventa una riga in AGENTS.md.**
6. ✂ Tenere il file sotto le 100–200 righe e spostare i dettagli in `docs/`.

# 7. Esempio: SoluzionaEvento

Struttura creata in C:\Repos\Devops\SoluzionaEvento.

# Creazione della struttura di SoluzionaEvento

## ? Caso reale

La struttura è stata creata in `C:\Repos\Devops\SoluzionaEvento`, con file vuoti da compilare. Sono stati usati script `.ps1` perché l'ambiente è Windows.

## ?? Script PowerShell equivalente

```
$root = 'C:\Repos\Devops\SoluzionaEvento'
$dirs  =
'docs','.claude\commands','.claude\skills','scripts','.github\workflows','src','tests'
$files = 'AGENTS.md','CLAUDE.md','README.md','.gitignore','.env.example',
         'docs\architecture.md','docs\conventions.md','.claude\settings.json',
         'scripts\setup.ps1','scripts\test.ps1','.github\workflows\ci.yml'
$dirs  | ForEach-Object { New-Item -ItemType Directory -Force -Path "$root\$_" | Out-Null }
$files | ForEach-Object { New-Item -ItemType File -Force -Path "$root\$_" | Out-Null }
```

**ℹ** Nessun `git init` è stato eseguito; i file `.gitkeep` servono solo a tracciare le cartelle vuote.

## ?? Prossimi passi

- Compilare `AGENTS.md` con stack e comandi reali.
- Scrivere `CLAUDE.md` con `@AGENTS.md`.
- Impostare `.claude/settings.json`.