Claude Code

Résumé détaillé — 26 slides — 4h de formation

4h | 26 slides | 60+ commandes | Agents & Teams

Anthropic 2026

docs.claude.com · Formation Claude Code

INTRO

Comprendre l'IA & les Tokens

Qu'est-ce qu'un LLM ?

Un Large Language Model **prédit** — il ne comprend pas. Il a été exposé à des milliards de textes et apprend à prédire le token le plus probable à chaque étape. Trois piliers :

`Données`	Milliards de textes, codes, livres, conversations humaines
`Modèle`	Réseau de neurones profond, milliards de paramètres ajustés par gradient descent
`Inférence`	À chaque token généré, prédit le suivant selon le contexte complet

■ Une IA ne *sait* rien — elle a appris à prédire ce qui est utile de dire.

Tokens & Plans

1 token ≈ ¾ de mot. 1 000 tokens ≈ 750 mots ≈ 3 pages A4. Ce qui consomme des tokens : prompt + historique + CLAUDE.md + résultats outils + réponse de Claude.

Plan	Prix	Tokens/5h	Contexte
Free	0 $	~8 800	200k
Pro	20 $/m	~44 000	200k
Max 5x	100 $/m	~88 000	1M*
Max 20x	200 $/m	~220 000	1M*
Team Std	25 $/u	~55 000	200k
Team Premium	125 $/u	~275 000	1M*
Enterprise	devis	illimité	500k

* 1M tokens dans Claude Code uniquement (Max/Team/Enterprise, mars 2026). Valeurs estimées — Anthropic ne publie pas les chiffres exacts.

MODULE 00

La Boucle Agentive

Le mécanisme fondamental

À chaque message, Claude reçoit **tout le contexte disponible**, réfléchit, décide d'une action, l'exécute via un outil, reçoit le résultat, et recommence. Ce n'est pas un échange question-réponse — c'est une boucle agentive.

USER	Votre prompt — le déclencheur de chaque tour
CONTEXT	CLAUDE.md + historique + résultats outils précédents
LLM	Raisonnement — choisit l'action la plus pertinente
ACTION	Outil appelé : Bash, Edit, Read, MCP...
RESULT	Résultat renvoyé dans le contexte — et on recommence
HOOKS	Scripts déterministes qui s'intercalent : Pre/PostToolUse, Stop

Insight clé : Claude n'a aucune mémoire entre les sessions. Tout vient du contexte injecté à chaque tour. La gestion du contexte est la compétence n°1.

MODULE 01

Installation & Premiers pas

Installation

```
# Windows PowerShell
irm https://claude.ai/install.ps1 | iex

# Linux
curl -fsSL https://claude.ai/install.sh | bash

# Via npm (toutes plateformes)
npm install -g @anthropic-ai/claude-code

# Vérifier
claude /doctor
claude update
```

Les 3 modes

`claude`	Mode REPL interactif — le mode quotidien
`claude -p '…'`	Print mode — one-shot, parfait pour scripting et CI/CD
`claude -c`	Reprendre la dernière session (contexte + historique conservés)

`claude -w`	Démarrer dans un worktree isolé

Les 3 modèles

`Opus 4.6`	Le plus puissant — architecture, problèmes complexes, raisonnement
`Sonnet 4.5`	Meilleur ratio qualité/coût — défaut sur plan Pro, usage quotidien
`Haiku 4.5`	Rapide et économique — tâches simples, itérations rapides

MODULE 02

Commandes & Raccourcis essentiels

Raccourcis clavier

ESC	Interrompre la génération en cours — économise des tokens
ESC × 2	Rewind — restaurer code seul, conversation seule, ou les deux
Shift+Tab	Cycler entre les modes de permission (default/acceptEdits/plan)
↑	Historique des prompts, même inter-sessions
ALT+T	Extended Thinking — Claude réfléchit plus longtemps
! cmd	Shell direct, bypass Claude — moins de tokens

Slash commands essentiels

/btw	Contexte silencieux — Claude l'intègre sans répondre. ex: /btw utilise pnpm
/clear	Vider le contexte — à utiliser à chaque nouvelle tâche, systématiquement
/compact	Compresser l'historique en résumé — quand la session dure
/context	Voir la consommation de tokens en temps réel
/plan	Mode planification — Claude propose un plan sans toucher au code
/effort high	Forcer Opus en mode intensif pour le prochain tour
/model	Changer de modèle en cours de session
/memory	Voir et gérer les mémoires auto-sauvegardées

MODULE 03

Configuration & Mémoire

CLAUDE.md — La mémoire projet

Chargé automatiquement à chaque session. C'est le briefing de Claude sur votre projet. **Moins de 2 000 tokens / 200 lignes max.** Pour les grosses docs, référencez avec @chemin/fichier.md.

`~/.claude/CLAUDE.md`	Global — tous vos projets
`.claude/CLAUDE.md`	Projet courant — prioritaire sur le global
`sous-dossier/CLAUDE.md`	Scope local — pour les monorepos

■ Contenu recommandé : stack, commandes build/test/lint, conventions de code, patterns à éviter, références architecture. Mettez à jour en cours de dev — la session d'aujourd'hui améliore toutes les suivantes.

settings.json

Trois niveaux de configuration, du plus global au plus spécifique.

`~/.claude/settings.json`	Global — tous vos projets
`.claude/settings.json`	Projet — prioritaire sur le global
`/etc/claude/settings.json`	Entreprise — lecture seule, imposé par l'organisation

```
{
"model": "claude-opus-4-6",
"permissions": {
"allow": ["Bash(git *)", "Edit(*.ts)"],
"deny": ["Bash(rm -rf *)"]
},
"hooks": {
"Stop": [{"hooks": [{"type": "command", "command": "npm test --silent"}]}]
},
"env": {
"CLAUDE_CODE_EFFORT_LEVEL": "medium",
"CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
}
}
```

Permissions & Sessions

`default`	Demande confirmation avant chaque action risquée
`acceptEdits`	Auto-accepte les éditions de fichiers — mode dev actif
`plan`	Propose un plan, n'exécute rien — idéal pour valider avant
`bypassPermissions`	Bypass tout — sandboxes et CI/CD uniquement

```
claude --permission-mode acceptEdits # au lancement

# Shift+Tab pour cycler en session


claude -c # reprendre dernière session

claude -r # reprendre par ID

claude --from-pr URL # depuis une pull request
```

MODULE 04

Hooks & Automatisation

Hooks — checkpoints déterministes

Les hooks s'exécutent à chaque fois, sans exception. **exit 2** = blocage dur. **exit 0** = succès. stdout → revient dans le contexte.

`PreToolUse`	Avant l'exécution d'un outil — bloquer, modifier, valider
`PostToolUse`	Après l'exécution — formatter, linter, type-checker
`Stop`	Quand Claude finit de répondre — lancer les tests, cleanup

`command`	Script shell — le type le plus courant
`http`	POST JSON vers une URL — webhooks, Slack...
`prompt`	Délègue la décision à un LLM — règles complexes en langage naturel
`agent`	Vérificateur complet avec accès aux outils

```
"hooks": {
"PreToolUse": [{"matcher": "Bash(rm -rf *)",
"hooks": [{"type": "command", "command": "exit 2"}]}],
"PostToolUse": [{"matcher": "Edit(*.ts)",
"hooks": [{"type": "command", "command": "npx tsc --noEmit"}]}],
"Stop": [{"hooks": [{"type": "command", "command": "npm test --silent"}]}]
}
```

output-format · /loop · /batch

`--output-format json`	Sortie parseable — session ID, résultat, métadonnées pour scripting
`--output-format stream-json`	Streaming temps réel — pour intégrations CI/CD avancées
`/loop 5m`	Polling à intervalles — chaque itération a son propre contexte
`/batch`	Changements massifs en parallèle sur worktrees — crée des PRs auto
`/simplify`	Review qualité pre-PR via un pipeline 3 agents

MODULE 05

Skills, Agents & Teams

Skills vs Slash Commands

`Skills`	Invoqués automatiquement par Claude quand le contexte le justifie
`Slash Commands`	Invoqués manuellement — vous tapez /cmd explicitement
`user-invocable`	Un skill avec user-invocable: true est disponible en slash command aussi

```
# SKILL.md — frontmatter minimal

---

name: code-review

description: Revue securite approfondie du code modifie. # <-- Claude lit ca pour decider

model: opus

allowed-tools: Read, Bash

user-invocable: true

context: fork # contexte isole

isolation: worktree # branche git dediee

---
```

> Skills bundlés disponibles par défaut : /debug · /loop · /simplify · /batch · /claude-api

Qu'est-ce qu'un Agent ?

Un agent = un LLM + une boucle d'action + des outils + un contexte isolé. Il agit, reçoit des résultats, décide, agit à nouveau — sans intervention humaine.

`context: fork`	Contexte isolé — l'agent ne voit pas le contexte parent
`isolation: worktree`	Branche git dédiée — zéro conflit avec les autres agents
`model: opus`	Choisir le modèle selon la complexité de la tâche de l'agent
`/memory`	Mémoires auto-sauvegardées par Claude entre les sessions

```
# Parler à un sous-agent dans un prompt :

Use the Task tool to spawn a subagent that reads all *.test.ts files

and generates a coverage report. Work independently,

return a JSON summary when done.
```

Orchestration multi-agents

Un orchestrateur décompose, délègue à des sous-agents spécialisés dans des worktrees isolés, et synthétise les résultats. Ne pas partager l'implémentation avec l'agent de tests.

`TaskCreated`	Hook déclenché quand une tâche est déléguée à un sous-agent
`TaskCompleted`	Déclenche automatiquement l'étape suivante
`TeammateIdle`	Un agent est en attente — lui donner du travail
`CLAUDE_CODE_SIMPLE=true`	Désactiver les agents pour débugger

Exemple — Équipe Dev complète

Scénario : implémenter un reset de mot de passe, de la spec à la PR, en autonomie.

1	2	3	4	5
Orchestrateur Reçoit spec	Agent Dev worktree dédié	Agent Test Sans voir impl	Agent Security OWASP check	Orchestrateur Ouvre la PR

```
# Prompt de l'orchestrateur

Implemente le reset de mot de passe. Specs : @docs/reset-password.md


Utilise Task tool :

Agent Dev → POST /auth/reset-password (isolation: worktree)

Agent Test → TU sans voir l'impl (isolation: worktree)

Agent Sec → checklist OWASP (isolation: worktree)

Quand tous done → PR avec rapport complet.
```

Combo Ultime — /plan + agents + worktrees

/plan → Claude propose un plan, valider avant d'agir (zéro code touché)

Agents → Orchestrateur spawne des agents via Task tool (context: fork)

Worktrees → Chaque agent sur sa branche dédiée (isolation: worktree)

Résultat : feature complète, testée, auditée, en une seule instruction.

```
# Étape 1 — Mode plan

/plan Implemente le login Google OAuth

# → Claude propose : Routes, Service, Tests, Securite → vous validez


# Étape 2 — Lancement de l'equipe

Utilise Task tool :

Agent Dev → OAuthService + routes (isolation: worktree)
```

```
Agent Test → TU + integration tests (isolation: worktree)

Agent Sec → checklist OWASP tokens/scopes (isolation: worktree)


# Étape 3 — Résultat

# 3 branches isolees → revue → PR fusionnee avec rapport complet
```

Git Worktrees

`claude -w`	Démarrer dans un worktree isolé (alias --worktree)
`worktree.sparsePaths`	Dans settings.json — checkout partiel pour monorepos
`WorktreeCreate/Remove hooks`	Automatiser setup/teardown de chaque worktree

3-4 worktrees simultanés — plusieurs Claude Code en parallèle, zéro conflit.

MCP — Connexion au monde extérieur

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

claude mcp add postgres npx @modelcontextprotocol/server-postgres

claude mcp add playwright npx @playwright/mcp@latest

/mcp # gérer en session
```

■ Limite : moins de 20k tokens MCP actifs. Désactivez les serveurs inutilisés.

■ Tool Search (2026) : lazy loading — les outils chargent à la demande (–95% tokens).

MODULE 06

Best Practices & Anti-patterns

À FAIRE

- /clear entre chaque tâche distincte — le réflexe numéro un
- CLAUDE.md sous 2 000 tokens — dense, pas exhaustif
- /plan avant d'implémenter — le nombre d'itérations chute
- Hooks de qualité : tests au Stop, type-check au PostToolUse
- Worktrees pour le travail parallèle — plusieurs Claude sans conflit
- CLAUDE.md vivant — mis à jour en cours de développement

À ÉVITER

- Hooks de formatage auto (Prettier → jusqu'à 160k tokens en 3 rounds)
- > 20k tokens de contexte MCP actifs — dégrade sérieusement les performances
- Multi-agents trop complexes trop tôt — commencer par 1 orchestrateur + 1 agent
- Instructions vagues ('améliore ce code') — soyez précis sur le contexte et l'attendu
- Sauter l'étape /plan sur les features complexes — particulièrement coûteux à défaire
- Historique > 100k tokens sans /compact — ralentissement, hallucinations, coûts inutiles

Ressources

docs.claude.com	Documentation officielle Claude Code
code.claude.com/docs/en/hooks	Référence hooks — 21 événements, 4 types
awesomeclaude.ai/code-cheatsheet	Toutes les commandes CLI en un coup d'œil
github.com/hesreallyhim/awesome-claude-code	Catalogue communautaire : skills, hooks, plugins
rosmur.github.io/claudecode-best-practices	Synthèse pratiques des équipes Anthropic
github.com/anthropics/claude-code/CHANGELOG	Suivi des nouveautés — Claude Code évolue vite