

# Plan de Formation

Claude Code 2.1.45 — Programme détaillé (2 jours)

The Bearded Bear

Février 2026



## Table des matières

<b>Plan de Formation : Claude Code 2.1.45 — Maîtriser l'Agent de Développement</b>	<b>2</b>
Contexte de Formation . . . . .	2
Version couverte . . . . .	2
Format : 2 jours (14h) . . . . .	2
Programme Détaillé . . . . .	2
JOUR 1 : Fondamentaux Claude Code (7h) . . . . .	2
JOUR 2 : Pratiques Avancées et Autonomie (7h) . . . . .	12
Options Additionnelles . . . . .	23
Option A : Demi-journée de suivi (3h) — J+15 . . . . .	23
Option B : Coaching continu (1 mois) . . . . .	23
Prérequis Participants . . . . .	23
Techniques . . . . .	23
Comptes . . . . .	23
Matériel . . . . .	24
Livrables Inclus . . . . .	24
Évaluation . . . . .	24
Quiz de validation des acquis (15min) . . . . .	24
Auto-évaluation des compétences . . . . .	25
Critères de réussite . . . . .	25
Adaptation . . . . .	26
Matrice de Couverture . . . . .	26
Features Claude Code 2.1.45 et modules correspondants . . . . .	26
Couverture par catégorie . . . . .	30

# Plan de Formation : Claude Code 2.1.45 — Maîtriser l’Agent de Développement

## Contexte de Formation

- **Public** : Équipe de développeurs
- **Objectif** : Maîtrise de Claude Code 2.1.45 comme agent de développement
- **Cas d’usage** : Tout projet logiciel, toute stack technologique
- **Format** : 2 jours (14h/groupe), max 5 stagiaires/groupe
- **Approche** : Progressive — du guidé vers l’autonome

## Version couverte

Composant	Version	Nouveautés clés
Claude Code	2.1.45	Extended Thinking, MCP, Sub-agents, Permissions 3-tier, Sonnet 4.6/Opus 4.6, Agent Teams, Fast Mode, Hooks, Headless Mode
Formation	1.0.0	Version initiale

## Format : 2 jours (14h)

Journée	Focus	Durée
Jour 1	Fondamentaux Claude Code	7h
Jour 2	Pratiques Avancées et Autonomie	7h

## Programme Détaillé

### JOUR 1 : Fondamentaux Claude Code (7h)

**Module 1 : Introduction à Claude Code (1h30)** **Objectifs pédagogiques :** - Comprendre ce qu'est Claude Code et son positionnement agentique - Installer Claude Code (CLI, Desktop, Web, IDE extensions) - Maîtriser les commandes de base et les modèles disponibles

**Contenu :**

**1. L'outil agentique (20min)**

- Définition : Claude Code est l'agent de développement officiel d'Anthropic
- Positionnement : 4% des commits GitHub, SDK -> Agent SDK
- Différence fondamentale avec ChatGPT, GitHub Copilot, Cursor :
  - ChatGPT : conversationnel, pas d'accès au système de fichiers
  - GitHub Copilot : auto-complétion inline, pas de raisonnement agentique
  - Cursor : IDE-first, fork de VS Code
  - Claude Code : terminal-first, agentique, accès système complet
- Les 7 interfaces d'accès :
  - **Terminal** (CLI) : interface principale, pleine puissance
  - **VS Code** : extension officielle avec inline edit, diff view
  - **JetBrains** : plugin officiel avec tool window
  - **Desktop app** : application native Anthropic
  - **Web** : [claude.ai/code](https://claude.ai/code)
  - **Slack** : intégration conversationnelle
  - **Chrome** : extension pour QA et navigation

**2. Installation (20min)**

- Installation native (recommandé) : Homebrew, curl, WinGet
- CLI via npm (fallback, déprécié) : `npm install -g @anthropic-ai/claude-code`
- Vérification : `claude --version (2.1.45+)`
- Desktop app : téléchargement et configuration
- Web interface : connexion sur [claude.ai/code](https://claude.ai/code)
- VS Code extension : installation depuis le marketplace
- JetBrains plugin : installation depuis le marketplace
- Configuration de la clé API Anthropic
- Premier lancement : `claude` dans un répertoire de projet
- `/doctor` pour diagnostiquer l'installation

**3. Interface et commandes de base (20min)**

- Commandes essentielles :
  - `/help` : aide et liste des commandes
  - `/status` : état de la session (tokens, coût, modèle)
  - `/clear` : vider le contexte
  - `/compact` : résumer le contexte sans le perdre
  - `/exit` ou `/quit` : quitter Claude Code
  - `/cost` : suivi détaillé de la consommation
  - `/doctor` : diagnostic de l'installation
- Commandes avancées :
  - `/model` : changer de modèle en cours de session

- `/fast` : basculer en mode rapide (même Opus 4.6, 2.5x plus rapide)
- `/permissions` : gérer les autorisations
- `/mcp` : configuration des serveurs MCP
- `/skills` : lister les skills disponibles
- `/keybindings` : personnaliser les raccourcis
- `/tasks` : gestion multi-tâches
- `/plan` : basculer en mode planification
- `/sandbox` : activer l'isolation OS
- `/rename` : renommer la session courante
- `/rewind` : revenir en arrière

#### 4. Modèles et coût (20min)

- Modèles disponibles et positionnement :
  - **Sonnet 4.6** (défaut) : équilibre vitesse/intelligence, usage quotidien, 200K tokens (1M en beta)
  - **Opus 4.6** (flagship) : modèle le plus puissant, 1M tokens, raisonnement complexe
  - **Haiku 4.5** : léger et rapide, tâches simples, prompt-based hooks, 200K tokens
- Changer de modèle :
  - `/model` en session interactive
  - `/fast` pour basculer Opus 4.6 en mode rapide
  - `--model` en ligne de commande
- Extended Thinking :
  - Automatique avec Opus 4.6 (adaptive thinking)
  - Mots-clés héritage (`think`, `think hard`, etc.) : dépréciés mais fonctionnels
  - `/effort` (low/medium/high) pour contrôler la profondeur
  - `/think` pour forcer la réflexion
- Monitoring et pricing :
  - `/cost` pour le suivi en temps réel
  - Context window : 200K (Sonnet 4.6/Haiku 4.5) à 1M tokens (Opus 4.6)
  - Statusline affiche le % de contexte utilisé
  - Stratégies d'optimisation des coûts

#### 5. Exercice pratique (10min)

- Installer Claude Code CLI
- Lancer une session dans un projet
- Exécuter un premier prompt
- Tester Extended Thinking : `think hard about the architecture of this project`
- Vérifier le coût avec `/cost`

**Support:** `modules/jour1/01-introduction-claude-code.md` **Exercice:** `exercices/exercice-01-premier-projet.md`

**Module 2 : CLAUDE.md et Configuration (1h30)** **Objectifs pédagogiques :** - Maîtriser le système de mémoire à 3 niveaux - Configurer les permissions et la sécurité - Utiliser les références contextuelles et .claudeignore

**Contenu :**

**1. Mémoire 3 niveaux (25min)**

- **Niveau 1 : Global** ~/ .claude/CLAUDE.md
  - Préférences personnelles du développeur
  - Conventions globales (langue, style, outils préférés)
  - S'applique à TOUS les projets
  - Exemple : Toujours répondre en français, utiliser des noms de variables en anglais
- **Niveau 2 : Projet (racine)** projet/CLAUDE.md
  - Instructions d'équipe, commitées dans Git
  - Stack technologique, conventions de commit
  - Architecture du projet
  - Exemple : Stack: Symfony 8.0 / PHP 8.5, Conventional Commits obligatoires
- **Niveau 3 : Projet (détaillé)** .claude/CLAUDE.md
  - Instructions détaillées, règles, références
  - Modularisation via .claude/rules/ et .claude/references/
  - Hiérarchie et priorité : global < projet racine < projet détaillé
- Bonnes pratiques :
  - CLAUDE.md principal < 200 lignes
  - Modulariser dans .claude/rules/ (un fichier par sujet)
  - Utiliser .claude/references/ pour la documentation technique
  - Chaque instruction supplémentaire dilue l'attention du modèle

**2. CLAUDE.local.md (10min)**

- Instructions privées, NON commitées dans Git
- Cas d'usage :
  - Préférences IDE personnelles
  - Chemins locaux spécifiques à la machine
  - Tokens et credentials de développement
  - Overrides temporaires
- Ajout automatique au .gitignore

**3. settings.json 3 niveaux (15min)**

- **Global** : ~/ .claude/settings.json
  - Préférences utilisateur par défaut
  - Permissions globales
- **Projet** : .claude/settings.json
  - Configuration spécifique au projet
  - Hooks, MCP servers, permissions
- **Enterprise** : /etc/claude/settings.json

- Politiques d'entreprise (IT/sécurité)
- Ne peut pas être override par les niveaux inférieurs
- Priorité : entreprise > projet > global
- Contenu configurable :
  - permissions : Allow, Deny, Ask
  - hooks : 13 événements automatiques
  - mcpServers : outils externes MCP
  - env : variables d'environnement

#### 4. **Permissions 3-tier** (15min)

- Les 3 niveaux de permissions :
  - **Ask** (défaut) : demande confirmation à chaque utilisation
  - **Allow** (auto-accept) : autorise automatiquement
  - **Deny** : interdit l'utilisation
- Syntaxe wildcards pour contrôle fin :
  - Bash(docker compose \*) : autorise toutes les commandes docker compose
  - Write(src/\*\*) : autorise l'écriture dans src/ et sous-répertoires
  - Bash(rm -rf \*) : deny pour bloquer les suppressions récursives
- Allowlist patterns :
  - Par outil : Bash, Write, Read, Glob, Grep, Edit
  - Par pattern de commande : regex supportées
- Deny rules de sécurité :
  - Bloquer les opérations destructives
  - Protéger les fichiers sensibles (.env, credentials)
- Configuration via settings.json ou /permissions

#### 5. **.claudeignore** (5min)

- Exclure fichiers et dossiers du contexte Claude
- Syntaxe identique à .gitignore
- Cas d'usage :
  - Fichiers binaires volumineux
  - Dossiers de build (node\_modules, vendor, dist)
  - Fichiers sensibles (.env, secrets)
  - Documentation externe non pertinente
- Placement : racine du projet

#### 6. **@ references** (10min)

- Injecter du contexte supplémentaire dans les prompts :
  - @fichier : référence un fichier spécifique
  - @dossier/ : référence un répertoire entier
  - @URL : référence une page web (lecture automatique)
- Cas d'usage :
  - @docs/architecture.md : référencer la documentation d'architecture
  - @src/Entity/User.php : référencer une entité pour contexte
  - @https://api-docs.example.com : référencer une documentation externe
- Combinaison avec les prompts :

- “En te basant sur @docs/specs.md, implémente la feature X”
- “Refactore @src/Service/ en suivant les patterns de @docs/patterns.md”

#### 7. Exercice pratique (10min)

- Créer un CLAUDE.md optimal pour un projet réel
- Configurer des permissions adaptées
- Tester les @ références
- Créer un .claudeignore

**Support :** modules/jour1/02-claudemd-configuration.md **Exercice :** exercices/exercice-02-configuration.md

---

**Module 3 : Patterns de Travail (2h) Objectifs pédagogiques :** - Formuler des prompts efficaces pour un agent IA - Maîtriser Plan Mode et la gestion du contexte - Utiliser les sub-agents, sessions et modes de travail

#### Contenu :

##### 1. Prompt engineering pour agents IA (20min)

- Principes fondamentaux :
  - Être spécifique et précis
  - Donner du contexte (fichiers, contraintes, architecture)
  - Définir le format de sortie attendu
  - Itérer et affiner
- Patterns efficaces :
  - “Analyse @src/Service/OrderService.php et identifie les violations SOLID”
  - “Génère un endpoint REST pour la création d'utilisateurs en suivant l'architecture existante”
  - “Écris les tests unitaires pour cette classe en couvrant les edge cases”
- Anti-patterns :
  - Prompts trop vagues : “corrige le bug” (lequel?)
  - Prompts trop longs sans structure
  - Ignorer le contexte existant du projet
- Techniques avancées :
  - Décomposer les tâches complexes en étapes
  - Fournir des exemples (few-shot)
  - Spécifier les contraintes négatives (“ne pas utiliser X”)

##### 2. Plan Mode (15min)

- Activation : /plan ou Shift+Tab pour basculer
- Principe : Explorer et planifier AVANT d'agir
- Workflow :
  1. Activer Plan Mode
  2. Claude explore le codebase, identifie les fichiers impactés



3. Claude propose un plan détaillé
  4. Le développeur valide ou ajuste
  5. Désactiver Plan Mode pour exécuter
- Avantages :
    - Sécurité : pas de modification accidentelle
    - Compréhension : vision d'ensemble avant action
    - Collaboration : validation humaine du plan
    - Économie : évite le travail à refaire
  - Quand l'utiliser :
    - Feature complexe (> 3 fichiers)
    - Refactoring architectural
    - Choix technologique
    - Impact incertain
3. **Gestion du contexte** (20min)
- Fenêtre de contexte : ~200K tokens
  - **Statusline** : affiche le % de contexte utilisé en temps réel
  - Seuils d'action :
    - < 30% : normal, continuer
    - 30-60% : surveiller, éviter les lectures inutiles
    - 60-80% : déléguer aux sub-agents, envisager /c l e a r
    - 80% : compaction imminente
  - /c l e a r : vider le contexte
    - Utiliser entre deux tâches NON liées
    - Après une longue investigation
    - Avant de commencer une nouvelle feature
    - NE PAS utiliser au milieu d'une tâche en cours
  - /compact : résumer le contexte sans le perdre
    - Préserve les informations essentielles
    - Libère de l'espace pour continuer
    - Moins radical que /c l e a r
  - Context compaction automatique :
    - Déclenchée quand le contexte approche la limite
    - Messages anciens sont résumés automatiquement
    - Hooks SessionStart : compact pour réinjecter le contexte critique
  - Signes de pollution du contexte :
    - Claude répète des informations déjà données
    - Réponses moins précises
    - Confusion entre tâches différentes
4. **Sub-agents** (15min)
- Principe : déléguer les recherches pour garder le contexte principal propre
  - Types de sub-agents :
    - **Explore** : recherche et exploration du codebase

- **Plan** : planification d'implémentation
  - **General-purpose** : exécution de tâches indépendantes
  - Task tool :
    - Chaque sub-agent a sa propre fenêtre de contexte
    - Le contexte principal reste propre
    - `run_in_background` pour parallélisme
  - Quand utiliser un sub-agent :
    - Investigation multi-fichiers (> 3 fichiers)
    - Explorer une architecture inconnue
    - Tâche indépendante en parallèle
    - Recherche d'un pattern dans le codebase
  - Exemple :
    - "Explore comment fonctionne l'authentification dans ce projet"
    - Le sub-agent explore et retourne un résumé
    - Le contexte principal reste propre
5. **Modes de travail** (15min)
- **Fast Mode** :
    - `/fast` pour basculer
    - Même modèle Opus 4.6, sortie 2.5x plus rapide, coût 6x (\$30/\$150 par M tokens)
    - Idéal pour tâches urgentes, à utiliser avec parcimonie (coût élevé)
  - **Headless Mode** :
    - `claude -p "prompt"` : exécution non-interactive
    - `--output-format text` : sortie texte brut
    - `--output-format json` : sortie structurée JSON
    - `--output-format stream-json` : streaming JSON
    - Piping: `echo "analyse ce fichier" | claude -p`
    - Intégration dans scripts bash, CI/CD, automation
  - **Mode interactif** (défaut) :
    - Session conversationnelle
    - Historique de la conversation
    - Itération et raffinement
6. **Session management** (10min)
- `--continue` : reprendre la dernière session
    - Retrouve le contexte où on l'a laissé
    - Idéal pour couper/reprendre le travail
  - `--resume` : reprendre une session spécifique par ID
    - Lister les sessions disponibles
    - Reprendre une session ancienne
  - `/rename` : renommer la session courante
    - Facilite l'identification des sessions
    - Convention : "feature-auth", "bugfix-login"
7. **Checkpointing et rewind** (10min)
- `Esc+Esc` : annuler la dernière action

- Interruption immédiate
- Claude s'arrête et attend
- `/rewind` : revenir en arrière dans la conversation
  - Annule les derniers échanges
  - Revient à un état précédent
  - Utile quand Claude part dans une mauvaise direction

#### 8. **Support images** (5min)

- Drag & drop d'images dans le terminal
- Coller des screenshots depuis le presse-papier
- Claude analyse visuellement l'image
- Cas d'usage :
  - Screenshots de bugs UI
  - Maquettes à implémenter
  - Diagrammes d'architecture
  - Messages d'erreur

#### 9. **Sandboxing** (5min)

- `/sandbox` pour activer l'isolation OS
- Protection du système de fichiers :
  - Claude ne peut modifier que les fichiers autorisés
  - Prévention des opérations destructives accidentelles
- Cas d'usage :
  - Expérimentation sur du code inconnu
  - Exécution de scripts non vérifiés
  - Environnement de formation sécurisé

#### 10. **Keybindings et statusline** (5min)

- `/keybindings` : personnaliser les raccourcis clavier
- Statusline : informations temps réel
  - Contexte utilisé (%)
  - Coût cumulé de la session
  - Modèle actif
  - Mode (plan/normal/fast)

**Support :** `modules/jour1/03-patterns-travail.md` **Exercice :** `exercices/exercice-03-patterns-travail.md`

---

**Module 4 : Pratique Guidée (2h) Objectifs pédagogiques :** - Appliquer Claude Code sur un projet existant (onboarding, refactoring, debugging) - Créer un projet depuis zéro avec Claude Code (scaffolding, features, tests)

#### **Contenu :**

##### 1. **Projet existant — Onboarding et Compréhension (30min)**

- Exploration d'un codebase inconnu :
    - "Explore l'architecture de ce projet et donne-moi une vue d'ensemble"
    - "Quels sont les principaux modules et leurs responsabilités?"
    - "Identifie les patterns architecturaux utilisés"
  - Documentation automatique :
    - Génération de diagrammes d'architecture
    - Inventaire des endpoints API
    - Cartographie des dépendances
  - Utilisation de Plan Mode pour comprendre avant d'agir
- 2. Projet existant — Refactoring Guidé (30min)**
- Identification des problèmes :
    - Violations SOLID
    - Code dupliqué (DRY)
    - Complexité excessive (KISS)
    - Dead code (YAGNI)
  - Refactoring assisté :
    - Plan Mode : établir la stratégie
    - Décomposition en étapes
    - Vérification par tests à chaque étape
  - Pattern : Investigation puis Implémentation
    1. Session 1 : explorer, comprendre, documenter
    2. /clear
    3. Session 2 : implémenter avec un contexte propre
- 3. Projet existant — Debugging Assisté (20min)**
- Analyse de logs et stack traces
  - Reproduction de bugs
  - Identification de la cause racine
  - Correction avec test de non-régression
  - Code review : relecture et suggestions d'amélioration
- 4. Projet vierge — Scaffolding (20min)**
- Création de l'architecture complète :
    - Structure des répertoires
    - Configuration du framework
    - Setup Docker / docker-compose
  - Claude génère la base :
    - "Crée un projet API REST avec cette architecture..."
    - "Configure Docker Compose avec PHP 8.5, PostgreSQL, Redis"
- 5. Projet vierge — Développement de Features (20min)**
- Génération de code guidé :
    - Entités et modèles
    - Services et use cases
    - Contrôleurs et endpoints API

- Tests unitaires et d'intégration
- Cycle TDD avec Claude :
  1. Décrire le comportement attendu
  2. Claude écrit les tests (RED)
  3. Claude écrit le code (GREEN)
  4. Claude refactorise (REFACTOR)

#### 6. **Projet vierge — CI/CD et Documentation (20min)**

- Mise en place CI/CD basique :
  - GitHub Actions ou GitLab CI
  - Pipeline : lint, tests, build
- Documentation automatique :
  - README génération
  - API documentation (OpenAPI)
  - Changelog initialisation

**Support :** modules/jour1/04-pratique-guidee.md **Exercice :** exercices/exercice-04-pratique-guidee.md

---

## JOUR 2 : Pratiques Avancées et Autonomie (7h)

---

**Module 5 : Hooks et Automatisation (1h30)** **Objectifs pédagogiques :** - Configurer et utiliser les 13 événements hooks - Créer des slash commands custom - Automatiser les workflows de développement

### Contenu :

1. **Les 13 événements hooks** (30min)
  - Liste complète des événements :

Événement	Déclencheur	Usage principal
PreToolUse	Avant l'utilisation d'un outil	Bloquer des patterns dangereux
PostToolUse	Après l'utilisation d'un outil	Lint automatique, formatage
PostToolUseFailure	Après un échec d'outil	Logging, notification d'erreurs
PermissionRequest	Demande de permission	Validation custom

Événement	Déclencheur	Usage principal
UserPromptSubmit	Soumission d'un prompt	Pré-traitement, validation
Stop	Arrêt de l'agent	Actions de cleanup
SubagentStop	Arrêt d'un sub-agent	Agrégation de résultats
SubagentStart	Démarrage d'un sub-agent	Initialisation de contexte
Notification	Notification système	Alertes externes
PreCompact	Avant la compaction	Sauvegarde du contexte
SessionStart	Début de session	Initialisation, réinjection
SessionEnd	Fin de session	Cleanup, métriques
Setup	Premier lancement	Configuration initiale

- Configuration dans settings.json :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "command": "php-cs-fixer fix $FILE_PATH"
      }
    ]
  }
}
```

- Matchers :

- Nom d'outil : Write, Bash, Edit
- Matchers spécialisés par événement (ex : compact pour SessionStart)

- Règles d'exécution :

- Timeout : 10 minutes max
- stdout/stderr capturés
- Exit codes : 0 = succès, 2 = bloquant (empêche l'action)

- Types de hooks :

- **Command-based** : exécute un script/commande
- **Prompt-based** (type : "prompt") : utilise Haiku 4.5 pour évaluer

## 2. Cas d'usage pratiques (20min)

- **Lint automatique** : PostToolUse : Write
  - Formater automatiquement chaque fichier modifié
  - PHP-CS-Fixer, Prettier, ESLint -fix
- **Blocage patterns dangereux** : PreToolUse : Bash
  - Bloquer `rm -rf /`, `DROP TABLE`, etc.
  - Regex de détection dans le hook
- **Notifications externes** :

- Envoyer une notification Slack/Teams quand une tâche est terminée
- Hook Stop pour notifier la complétion
- **Pre-compact backup :**
  - Sauvegarder le contexte essentiel avant compaction
  - Réinjection via SessionStart :compact
- **Validation de commit :**
  - Vérifier les Conventional Commits
  - Bloquer si format invalide

### 3. Hooks de session (10min)

- SessionStart avec matcher compact :
  - Se déclenche après une compaction automatique
  - Réinjecter le contexte critique
  - Configuration :

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "command": "cat .claude/context-essentials.md"
      }
    ]
  }
}
```
- SessionStart:startup pour initialisation :
  - Charger l'état du projet au démarrage
  - Vérifier les prérequis
  - Afficher un rappel des tâches en cours

### 4. Slash commands custom (20min)

- Emplacement : `.claude/commands/*.md`
- Variables disponibles :
  - `$ARGUMENTS` : arguments passés à la commande
  - `$SELECTION` : texte sélectionné (IDE)
- Création d'une commande d'équipe :
  - Fichier : `.claude/commands/review.md`
  - Contenu : instructions de review spécifiques au projet
  - Usage : `/review src/Service/OrderService.php`
- Exemples de commandes utiles :
  - `/audit` : audit qualité du code
  - `/review` : code review assistée
  - `/deploy` : checklist de déploiement
  - `/onboard` : guide d'onboarding nouveau développeur
- Partage en équipe : commitées dans Git

### 5. Exercice pratique (10min)

- Créer un hook PreToolUse qui bloque les commandes dangereuses

- Créer un hook PostToolUse :Write pour lint automatique
- Créer une commande custom /audit
- Tester les hooks en action

**Support :** modules/jour2/05-hooks-automatisation.md **Exercice :** exercices/exercice-05-hooks-commands.md

**Module 6 : MCP et Intégrations (1h15) Objectifs pédagogiques :** - Comprendre et configurer le Model Context Protocol (MCP) - Intégrer Claude Code dans l'écosystème de développement (IDE, CI/CD) - Maîtriser les plugins et outils d'extension

### Contenu :

#### 1. MCP — Model Context Protocol (25min)

- Concept : protocole ouvert pour connecter des outils externes à Claude
- Architecture client-serveur :
  - Claude Code = client MCP
  - Serveurs MCP = fournisseurs d'outils
- Types de serveurs MCP :
  - Base de données (PostgreSQL, MySQL, SQLite)
  - APIs externes (GitHub, Jira, Confluence)
  - Navigateur web (Puppeteer, Playwright)
  - Système de fichiers avancé
  - Outils métier custom
- Configuration dans settings.json :

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "DATABASE_URL": "postgresql://..."
      }
    }
  }
}
```

- CLI: `claude mcp add <name> <command> [args...]`
- MCP\_TIMEOUT : configuration du timeout
- OAuth pour serveurs authentifiés :
  - `--client-id` et `--client-secret`
  - Flux d'authentification OAuth2

#### 2. Plugins (10min)

- Installation: `/plugin install <name>`



- Plugins LSP (Language Server Protocol) :
  - `php-lsp` : analyse PHP en temps réel
  - `pyright-lsp` : analyse Python avec types
  - `typescript-lsp` : analyse TypeScript
- Différence plugins vs MCP :
  - Plugins : intégrés dans Claude Code, fonctionnalités prédéfinies
  - MCP : protocole ouvert, outils externes illimités
- Gestion : `/plugin list`, `/plugin remove`

### 3. IDE intégrations (15min)

- **VS Code** :
  - Extension officielle Claude Code
  - Inline edit : modifier du code directement dans l'éditeur
  - Diff view : visualiser les changements proposés
  - Session picker : gérer plusieurs sessions
  - `$SELECTION` : passer la sélection au prompt
- **JetBrains** :
  - Plugin officiel
  - Tool window dédiée
  - Intégration avec les outils JetBrains (refactoring, tests)
- Fonctionnalités communes :
  - Lancer des commandes depuis l'IDE
  - Voir les résultats inline
  - Navigation fichiers assistée

### 4. CI/CD (15min)

- **GitHub Actions** :
  - Utiliser `claude -p "prompt"` en headless mode
  - Revue automatique de PR
  - Génération de changelogs
  - Vérification de qualité de code
- **GitLab CI** :
  - Même principe en headless mode
  - Pipeline de validation
- Configuration :
  - Variables d'environnement pour la clé API
  - `--output-format json` pour parsing automatique
  - Timeout et gestion d'erreurs

### 5. `--from-pr` (5min)

- Review automatisée de Pull Requests :
  - `claude --from-pr 123` : analyse la PR #123
  - Commentaires inline sur le code
- Auto-link PR : Claude référence automatiquement la PR
- Indicateurs de statut : `check pass/fail`

### 6. Sécurité MCP (5min)

- Risques des serveurs MCP tiers :
  - Injection de commandes
  - Exfiltration de données
  - Exécution de code arbitraire
- Bonnes pratiques :
  - Auditer le code source avant installation
  - Préférer écrire ses propres serveurs MCP
  - Limiter les permissions (tools allowlist)
  - Version pinée (pas de latest)
  - Pas d'accès réseau non justifié

**Support :** modules/jour2/06-mcp-integrations.md **Exercice :** exercices/exercice-06-mcp-setup.md

---

**Module 7 : Multi-Agent et Coordination (1h15) Objectifs pédagogiques :** - Maîtriser Agent Teams pour le travail parallèle - Utiliser les patterns de coordination avancés (fan-out, writer/reviewer) - Exploiter Git worktrees pour les sessions concurrentes

**Contenu :**

1. **Agent Teams** (25min)

- Concept : coordination multi-agents pour tâches complexes
- Outils de coordination :
  - TeamCreate : créer une équipe d'agents
  - SendMessage : communication inter-agents
  - TaskCreate / TaskUpdate : gestion de tâches partagées
- Architecture Leader-Teammates :
  - Un agent leader coordonne
  - Des agents teammates exécutent en parallèle
  - Communication via messages structurés
- Spawn teammates via Task tool avec team\_name :
  - Chaque teammate a son propre contexte
  - Résultats agrégés par le leader
- Idle state et message delivery :
  - Agents en attente quand pas de tâche
  - Messages délivrés de manière asynchrone

2. **Git worktrees** (15min)

- Principe : sessions parallèles sur branches séparées
- Flag -w pour lancer Claude dans un worktree :
  - Chaque worktree = une session Claude indépendante
  - Travail parallèle sans conflit
- Setup :

```
git worktree add ../feature-auth feature/auth  
cd ../feature-auth && claude
```

- Pattern Writer/Reviewer :
  - Terminal 1 (Writer) : implémente la feature
  - Terminal 2 (Reviewer) : revoit le code
  - Contexte frais, pas de biais d’auteur
- Recommandations :
  - 3-5 worktrees maximum simultanément
  - Un worktree = une tâche
  - Cleanup après complétion

### 3. **Fan-out patterns** (15min)

- Opérations batch parallèles :
  - Refactoring multi-fichiers simultané
  - Migrations de code en parallèle
  - Analyses de codebase distribuées
- Task tool avec `run_in_background` :
  - Lancer plusieurs sub-agents en parallèle
  - Chacun travaille sur une partie du problème
- Agrégation des résultats :
  - Le leader collecte les résultats
  - Synthèse et résolution des conflits
- Cas d’usage :
  - “Refactore tous les services de ce dossier en parallèle”
  - “Analyse les 10 fichiers les plus complexes simultanément”

### 4. **Interview pattern** (10min)

- AskUserQuestion pour collecte structurée :
  - Claude pose des questions ciblées
  - Collecte progressive d’informations
- Cas d’usage :
  - Onboarding d’un nouveau projet
  - Configuration interactive
  - Clarification de requirements
- Exemple :
  - “Quel framework utilisez-vous ?”
  - “Quelle base de données ?”
  - “Quels sont vos patterns architecturaux ?”

### 5. **Writer/Reviewer pattern** (10min)

- 2 agents en parallèle :
  - Agent 1 écrit le code
  - Agent 2 revoit le code
- Avantages :
  - Relecture croisée sans biais d’auteur
  - Détection précoce des problèmes

- Qualité supérieure du code produit
- Implémentation avec worktrees :
  - Branches séparées, merge après validation

**Support :** modules/jour2/07-multi-agent.md **Exercice :** exercices/exercice-07-agent-teams.md

---

**Module 8 : Qualité et Sécurité (1h) Objectifs pédagogiques :** - Intégrer TDD/BDD dans le workflow Claude Code - Appliquer les bonnes pratiques de sécurité - Maîtriser le workflow Git avec Claude Code

**Contenu :**

1. **TDD/BDD avec Claude Code** (20min)

- Cycle Red-Green-Refactor avec Claude :
  1. **RED** : décrire le comportement attendu, Claude écrit le test
  2. **GREEN** : Claude écrit le code minimal pour passer
  3. **REFACTOR** : Claude améliore le code, les tests restent verts
- Génération de tests :
  - Tests unitaires : logique métier isolée
  - Tests d'intégration : connexions entre composants
  - Tests E2E : parcours utilisateur complet
- BDD avec Gherkin :
  - Scénarios Given-When-Then
  - Claude génère les step definitions
- Couverture :
  - Objectif :  $\geq 80\%$  de couverture
  - /cost pour surveiller l'investissement
- Bonnes pratiques :
  - Un assert par test (préférence)
  - Nommage explicite des tests
  - Tests indépendants et reproductibles
  - Factories et fixtures

2. **Audit code** (15min)

- Analyse d'architecture :
  - Détection des violations de couches
  - Identification des dépendances circulaires
  - Vérification des patterns (SOLID, DRY, KISS, YAGNI)
- Détection d'anti-patterns :
  - God classes
  - Feature envy

- Shotgun surgery
- Dead code
- Code review assistée :
  - Claude analyse le diff
  - Suggestions d'amélioration
  - Identification des risques
- Refactoring guidé :
  - Plan Mode pour la stratégie
  - Étapes incrémentales
  - Tests à chaque étape

### 3. **Sécurité** (10min)

- OWASP Top 10 : détection avec Claude
  - Injection SQL, XSS, CSRF
  - Broken authentication
  - Security misconfiguration
- Audit de dépendances :
  - Détection de vulnérabilités connues (CVE)
  - Mise à jour guidée
- Détection de secrets :
  - Clés API, tokens, mots de passe en dur
  - Recommandations de remédiation
- Bonnes pratiques :
  - Validation des entrées
  - Requêtes paramétrées
  - Headers de sécurité
  - Chiffrement des données sensibles

### 4. **Git workflow avec Claude Code** (10min)

- Conventional Commits :
  - `feat:`, `fix:`, `refactor:`, `test:`, `docs:`, `chore:`
  - Scope entre parenthèses: `feat(auth): add JWT validation`
  - Claude génère des messages de commit conformes
- Pull Request :
  - Création assistée : description, checklist, labels
  - Code review automatique avec `--from-pr`
- Branches :
  - Nomenclature : `feature/`, `fix/`, `refactor/`
  - Durée de vie < 3 jours
  - Squash merge pour historique propre

### 5. **Cost management** (5min)

- Suivi de la consommation :
  - `/cost` pour le détail de la session
  - Statusline pour le suivi temps réel
- Optimisation des tokens :

- /clear entre tâches non liées
- Sub-agents pour les investigations
- CLAUDE.md concis (< 200 lignes)
- Choix du modèle selon la tâche :
  - Haiku 4.5 pour tâches simples et répétitives (\$1/\$5 par M tokens)
  - Sonnet 4.6 pour usage quotidien (\$3/\$15 par M tokens)
  - Opus 4.6 pour raisonnement complexe (\$5/\$25 par M tokens)
- Budget d'équipe et monitoring

**Support :** modules/jour2/08-qualite-securite.md **Exercice :** exercices/exercice-08-qualite-securite.md

---

**Module 9 : Bonus — Claude Craft (30min)** **Objectifs pédagogiques :** - Découvrir Claude-Craft comme extension de Claude Code - Comprendre les possibilités d'extension du framework - Savoir où trouver plus d'informations

**Contenu :**

**1. Présentation condensée (10min)**

- Claude-Craft : framework multi-technologie pour Claude Code
- Installation : `npx @the-bearded-bear/claude-craft install . --tech=symfony --lang=fr`
- 18 stacks technologiques supportées
- 63 agents spécialisés
- 204 commandes réparties en 26 namespaces

**2. Points forts (10min)**

- BMAD v6 : framework de gestion de projet intégré
- Ralph Wiggum : boucle autonome jusqu'à complétion
- QA Recette : tests d'acceptance automatisés via Chrome
- TCL (Tiered Context Loading) : économie de ~95% de tokens
- Agents spécialisés par technologie

**3. Illustration des possibilités d'extension (10min)**

- Comment Claude-Craft étend Claude Code :
  - CLAUDE.md modulaire
  - Commands custom avancées
  - Agents avec personas
  - Références techniques chargées à la demande
- Où trouver plus d'informations :
  - Documentation officielle Claude-Craft
  - Formation dédiée Claude-Craft (2 jours supplémentaires)
  - GitHub : `@the-bearded-bear/claude-craft`

**Support :** modules/jour2/09-bonus-claude-craft.md

---

**Module 10 : Atelier Final (1h30) Objectifs pédagogiques :** - Appliquer toutes les connaissances en situation réelle - Établir les bonnes pratiques d'équipe - Créer un plan d'action concret

**Contenu :**

1. **Challenge mixte** (45min)

— **Exercice 1 : Projet existant** (20min)

- Prendre un projet de l'équipe ou un projet préparé
- Audit complet : architecture, qualité, sécurité
- Identifier les 3 améliorations prioritaires
- Implémenter la plus impactante avec Claude Code
- Utiliser Plan Mode, sub-agents, hooks

— **Exercice 2 : Projet vierge** (25min)

- Créer un micro-projet de A à Z
- Scaffolding avec Claude Code
- Implémenter une feature complète avec TDD
- Configurer les hooks et permissions
- Générer la documentation

2. **Restitution** (15min)

- Chaque binôme/participant présente sa solution
- Partage des apprentissages et découvertes
- Discussion sur les choix techniques
- Retour d'expérience sur les difficultés rencontrées

3. **Q&A** (15min)

- Questions ouvertes sur tous les modules
- Cas spécifiques de l'équipe
- Scénarios avancés non couverts
- Bonnes pratiques adaptées au contexte de l'équipe

4. **Plan d'action équipe** (15min)

- Bonnes pratiques à adopter immédiatement :
  - CLAUDE.md d'équipe à définir et commiter
  - Conventions de travail avec Claude Code
  - Permissions et hooks à configurer
  - Workflow Git adapté
- Roadmap d'adoption :
  - Semaine 1 : installation et configuration
  - Semaine 2 : utilisation guidée sur tâches simples
  - Semaine 3 : autonomie progressive
  - Semaine 4 : revue et optimisation
- Indicateurs de succès :

- Temps de développement réduit
- Qualité de code améliorée
- Couverture de tests augmentée
- Satisfaction équipe

**Support :** modules/jour2/10-atelier-final.md **Exercice :** exercices/exercice-09-challenge-final.md

---

## Options Additionnelles

### Option A : Demi-journée de suivi (3h) — J+15

**Contenu :** - Retour d'expérience de l'équipe (1h) - Quelles fonctionnalités sont les plus utilisées? - Quelles difficultés ont été rencontrées? - Quels patterns se sont révélés les plus utiles? - Résolution des problèmes rencontrés (1h) - Debugging des configurations - Optimisation des CLAUDE.md - Ajustement des permissions et hooks - Approfondissement sur demande (30min) - MCP avancé, Agent Teams, Headless CI/CD - Cas d'usage spécifiques à l'équipe - Optimisation des workflows établis (30min) - Revue des hooks et commandes custom - Partage des meilleures configurations - Plan d'amélioration continue

### Option B : Coaching continu (1 mois)

**Contenu :** - 2h de support par semaine - Revue de code assistée par Claude Code - Accompagnement sur projets spécifiques - Accès prioritaire au formateur - Suivi des indicateurs de performance - Ajustement continu des configurations

---

## Prérequis Participants

### Techniques

- Connaissance du développement logiciel (tout langage)
- Familiarité avec Git (commit, branch, merge, PR)
- IDE installé (VS Code recommandé)
- Node.js installé (pour l'installation CLI via npm)
- Terminal accessible (bash, zsh, PowerShell)

### Comptes

- Compte Anthropic actif avec crédits suffisants
- Accès aux dépôts de code de l'équipe (si exercices sur projet réel)



## Matériel

- Ordinateur portable avec accès Internet
- Droits d'installation de logiciels (npm global)
- Au moins 8 Go de RAM recommandés

## Livrables Inclus

Livrable	Description
<b>Supports de formation</b>	Support de présentation complet (PDF)
<b>Cheat sheet Claude Code</b>	Aide-mémoire avec toutes les commandes, raccourcis et patterns
<b>Exercices et solutions</b>	Tous les ateliers pratiques avec corrections détaillées
<b>Templates de configuration</b>	CLAUDE.md, settings.json, hooks, commands custom prêts à l'emploi
<b>Certificat</b>	Attestation de participation à la formation
<b>Accès support</b>	Support post-formation par email/Slack (1 mois)

## Évaluation

### Quiz de validation des acquis (15min)

#### Questions types :

Thème	Exemple de question
Installation	Quelle commande installe Claude Code en global ?
Commandes	Quelle commande vide le contexte de la session ?
CLAUDE.md	Quels sont les 3 niveaux de mémoire CLAUDE.md ?
Permissions	Quels sont les 3 niveaux de permissions ?
Extended Thinking	Quels sont les 4 niveaux de réflexion ?
Modèles	Quel est le modèle par défaut de Claude Code ?
Contexte	À partir de quel % de contexte faut-il déléguer aux sub-agents ?

Thème	Exemple de question
Hooks	Combien d'événements hooks existent ?
MCP	À quoi sert le Model Context Protocol ?
Plan Mode	Quand utiliser Plan Mode ?
Sub-agents	Quels sont les 3 types de sub-agents ?
Git	Quel format de commit est recommandé ?
Fast Mode	Que fait / fast exactement ?
Sessions	Quelle option reprend la dernière session ?
Agent Teams	Quel est le pattern de coordination principal ?

### Auto-évaluation des compétences

Compétence	Débutant	Intermédiaire	Avancé
Installation et configuration			
Prompts efficaces			
Plan Mode et contexte			
Hooks et automatisation			
MCP et intégrations			
Multi-agent et coordination			
TDD/BDD avec Claude			
Git workflow			

### Critères de réussite

Critère	Objectif
Installation Claude Code	Autonome, fonctionnel
Configuration CLAUDE.md	Optimal pour un projet
Prompt engineering	Prompts spécifiques et efficaces
Plan Mode	Utilisé pour les tâches complexes
Hooks	Au moins 2 hooks configurés
Sub-agents	Utilisés pour les investigations

Critère	Objectif
TDD	Cycle Red-Green-Refactor maîtrisé
Git workflow	Conventional Commits automatisés

## Adaptation

Ce plan est adaptable selon :

- **Durée** : Condensable sur 1 jour (essentiel) ou extensible sur 3 jours (approfondi)
- **Stack** : Exemples adaptables à toute stack (PHP, Python, JS, C#, Dart...)
- **Niveau** : Junior (plus de guidage), confirmé (équilibre), senior (focus avancé)
- **Focus** : Projets existants uniquement, greenfield uniquement, ou mixte
- **Spécialisation** : Module MCP étendu, Module Agent Teams étendu, Module CI/CD étendu

## Matrice de Couverture

### Features Claude Code 2.1.45 et modules correspondants

Feature	Module	Couverture
<b>Installation CLI</b>	M1	Complète
<b>Desktop App</b>	M1	Mentionnée
<b>Web Interface</b>	M1	Mentionnée
<b>VS Code Extension</b>	M6	Complète
<b>JetBrains Plugin</b>	M6	Complète
<b>Slack Integration</b>	M1	Mentionnée
<b>Chrome Extension</b>	M1	Mentionnée
<b>Commandes de base</b> (/help, /clear, /exit)	M1	Complète
<b>Extended Thinking</b> (automatique Opus 4.6, /effort)	M1	Complète
<b>/think pour forcer réflexion</b>	M1	Complète

Feature	Module	Couverture
<b>Modèle Sonnet 4.6</b>	M1	Complète
<b>Modèle Opus 4.6</b>	M1	Complète
<b>Modèle Haiku 4.5</b>	M1, M5	Complète
<b>/model changement modèle</b>	M1	Complète
<b>/fast Fast Mode</b>	M3	Complète
<b>/cost suivi consommation</b>	M1, M8	Complète
<b>/compact résumé contexte</b>	M3	Complète
<b>/doctor diagnostic</b>	M1	Mentionnée
<b>/plan Plan Mode</b>	M3	Complète
<b>/sandbox isolation</b>	M3	Complète
<b>/permissions gestion droits</b>	M2	Complète
<b>/mcp configuration MCP</b>	M6	Complète
<b>/skills skills disponibles</b>	M5	Mentionnée
<b>/keybindings raccourcis</b>	M3	Complète
<b>/tasks multi-tâches</b>	M7	Mentionnée
<b>/rename renommer session</b>	M3	Complète
<b>/rewind retour arrière</b>	M3	Complète
<b>CLAUDE.md 3 niveaux</b> (global, projet, détaillé)	M2	Complète
<b>CLAUDE.local.md</b>	M2	Complète
<b>settings.json 3 niveaux</b> (global, projet, entreprise)	M2	Complète
<b>Permissions 3-tier</b> (Ask, Allow, Deny)	M2	Complète
<b>Wildcards permissions</b>	M2	Complète
<b>.claudeignore</b>	M2	Complète
<b>@ references</b> (fichier, dossier, URL)	M2	Complète
<b>Prompt engineering agents</b>	M3	Complète

Feature	Module	Couverture
<b>Gestion contexte / fenêtre tokens</b>	M3	Complète
<b>Statusline</b>	M3	Complète
<b>Sub-agents</b> (Explore, Plan, General)	M3	Complète
<b>Task tool</b>	M3, M7	Complète
<b>run_in_background</b>	M3, M7	Complète
<b>Headless mode</b> (claude -p)	M3	Complète
<b>Output formats</b> (text, json, stream-json)	M3	Complète
<b>Piping stdin</b>	M3	Complète
<b>--continue reprendre session</b>	M3	Complète
<b>--resume reprendre session spécifique</b>	M3	Complète
<b>Esc+Esc interruption</b>	M3	Complète
<b>Support images</b> (drag & drop, paste)	M3	Complète
<b>13 événements hooks</b>	M5	Complète
<b>PreToolUse</b>	M5	Complète
<b>PostToolUse</b>	M5	Complète
<b>PostToolUseFailure</b>	M5	Complète
<b>PermissionRequest</b>	M5	Complète
<b>UserPromptSubmit</b>	M5	Complète
<b>Stop</b>	M5	Complète
<b>SubagentStop</b>	M5	Complète
<b>SubagentStart</b>	M5	Complète
<b>Notification</b>	M5	Complète
<b>PreCompact</b>	M5	Complète
<b>SessionStart</b>	M5	Complète
<b>SessionEnd</b>	M5	Complète
<b>Setup</b>	M5	Complète

Feature	Module	Couverture
<b>Prompt-based hooks</b> (Haiku 4.5)	M5	Complète
<b>Matchers hooks</b>	M5	Complète
<b>Exit codes hooks</b> (0, 2)	M5	Complète
<b>Slash commands custom</b> (.claude/commands/)	M5	Complète
<b>Variables</b> (\$ARGUMENTS, \$SELECTION)	M5	Complète
<b>MCP — Model Context Protocol</b>	M6	Complète
<b>claude mcp add</b>	M6	Complète
<b>MCP_TIMEOUT</b>	M6	Complète
<b>MCP OAuth</b> (--client-id, --client-secret)	M6	Complète
<b>Plugins</b> (/plugin install)	M6	Complète
<b>LSP plugins</b> (php, pyright, typescript)	M6	Complète
<b>CI/CD integration</b> (GitHub Actions, GitLab CI)	M6	Complète
<b>--from-pr review PR</b>	M6	Complète
<b>Sécurité MCP</b> (vetting, risques)	M6	Complète
<b>Agent Teams</b> (TeamCreate, SendMessage)	M7	Complète
<b>Leader-Teammates coordination</b>	M7	Complète
<b>team_name spawn</b>	M7	Complète
<b>Git worktrees</b> (-w flag)	M7	Complète
<b>Fan-out patterns</b>	M7	Complète
<b>Interview pattern</b> (AskUserQuestion)	M7	Complète
<b>Writer/Reviewer pattern</b>	M7	Complète
<b>TDD Red-Green-Refactor</b>	M8	Complète

Feature	Module	Couverture
<b>BDD Gherkin</b>	M8	Complète
<b>Audit code</b> (architecture, anti-patterns)	M8	Complète
<b>OWASP Top 10 détection</b>	M8	Complète
<b>Conventional Commits</b>	M8	Complète
<b>PR création assistée</b>	M8	Complète
<b>Cost management</b>	M8	Complète
<b>Context compaction automatique</b>	M3	Complète
<b>Hooks</b>	M5	Complète
<b>SessionStart :compact</b>		
<b>Inline edit (VS Code)</b>	M6	Complète
<b>Diff view (VS Code)</b>	M6	Complète
<b>Session picker (VS Code)</b>	M6	Complète
<b>Tool window (JetBrains)</b>	M6	Complète

### Couverture par catégorie

Catégorie	Features	Couvertes	%
Installation & Interfaces	7	7	100%
Commandes de base	15	15	100%
Configuration & Mémoire	8	8	100%
Modèles & Extended Thinking	7	7	100%
Gestion du contexte	8	8	100%
Sub-agents & Multi-agent	9	9	100%
Hooks (13 événements)	15	15	100%
MCP & Plugins	7	7	100%
IDE Intégrations	6	6	100%
CI/CD & Git	5	5	100%
Sécurité	3	3	100%
Modes de travail	6	6	100%
<b>TOTAL</b>	<b>96</b>	<b>96</b>	<b>100%</b>

---

Catégorie	Features	Couvertes	%
-----------	----------	-----------	---

---

---

**Version** : 1.0.0 **Date** : Février 2026 **Auteur** : The Bearded CTO **Claude Code** : 2.1.45