

Module 2 — CLAUDE.md et Configuration

Jour 1 — Memoire et permissions

The Bearded Bear

Février 2026



Table des matières

Module 2 : CLAUDE.md et Configuration	3
Objectifs	3
1. La memoire a 3 niveaux : CLAUDE.md	3
1.1 Concept	3
1.2 Niveau 1 : Global (~/.claude/CLAUDE.md)	4
1.3 Niveau 2 : Projet (./CLAUDE.md)	4
1.4 Niveau 3 : Detaille (./.claude/CLAUDE.md)	5
1.5 Priorite de chargement	5
1.6 Bonnes pratiques de taille	5
2. CLAUDE.local.md : Instructions privees	6
2.1 Concept	6
2.2 Emplacement	6
2.3 Cas d'usage	6
2.4 Ne pas oublier le .gitignore	6
3. Settings.json : Configuration avantee	7
3.1 Les 3 niveaux de settings	7
3.2 Priorite	7
3.3 Structure du fichier	7
3.4 Exemple complet : settings.json projet	7
3.5 Exemple : settings.json global	8
3.6 Exemple : settings.json entreprise	8
4. Permissions 3-tier	9
4.1 Les trois niveaux de permission	9
4.2 Syntaxe des regles de permission	9
4.3 Exemples de configuration	9
4.4 Wildcards	11
5. .claudeignore : Exclure des fichiers	11
5.1 Concept	11
5.2 Emplacement	11
5.3 Syntaxe	11
5.4 Pourquoi c'est important?	12
5.5 Recommandation	12
6. Les @ references	13
6.1 Concept	13
6.2 Syntaxe	13
6.3 Exemples pratiques	13
6.4 Utilisation dans CLAUDE.md	14
6.5 Cas d'usage avances	14
7. Structure recommandee d'un projet	14
7.1 Arborescence complete	14
7.2 Fichiers commites vs non commites	15

Exercice pratique : Configurer un CLAUDE.md optimal (30 min)	15
Etape 1 : Creer la structure (5 min)	15
Etape 2 : Rediger le CLAUDE.md projet (10 min)	15
Etape 3 : Configurer les permissions (5 min)	16
Etape 4 : Creer le .claudeignore (5 min)	16
Etape 5 : Tester avec Claude Code (5 min)	17
Points Cles a Retenir	17

Module 2 : CLAUDE.md et Configuration

Duree : 1h30 **Prerequis :** Module 1 - Introduction a Claude Code **Prochain module :** Module 3 - Patterns de Travail

Objectifs

A l'issue de ce module, vous serez capable de :

1. Comprendre le systeme de memoire a 3 niveaux de CLAUDE.md
2. Configurer des instructions privees avec CLAUDE.local.md
3. Maitriser les settings.json a 3 niveaux de priorite
4. Configurer les permissions avec le systeme 3-tier (Ask, Allow, Deny)
5. Utiliser .claudeignore pour exclure des fichiers du contexte
6. Injecter du contexte dynamiquement avec les @ references
7. Rediger un CLAUDE.md optimal pour un projet reel

1. La memoire a 3 niveaux : CLAUDE.md

1.1 Concept

Claude Code utilise un systeme hierarchique de fichiers CLAUDE . md pour recevoir des instructions persistantes. Ces fichiers sont lus automatiquement au demarrage de chaque session et injectes dans le contexte.

Niveau 1 : GLOBAL (~/.claude/CLAUDE.md) → Instructions personnelles, preferences → S'applique a TOUS vos projets
Niveau 2 : PROJET (./CLAUDE.md) → Instructions d'equipe, conventions → Commite dans le repo, partage avec l'equipe
Niveau 3 : DETAILLE (./.claude/CLAUDE.md) → Instructions techniques detaillees → Architecture, regles specifiques

1.2 Niveau 1 : Global (~/.claude/CLAUDE.md)

Ce fichier contient vos **preferences personnelles** qui s'appliquent a tous vos projets.

Preferences globales

- Repondre en francais par default
- Utiliser des commentaires en anglais dans le code
- Preferer les approches fonctionnelles quand possible
- Toujours proposer des tests unitaires
- Utiliser des noms de variables explicites

Cas d'usage : - Langue de communication preferee - Style de code personnel - Preferences d'outils (vim, emacs, etc.) - Conventions de nommage personnelles

1.3 Niveau 2 : Projet (./CLAUDE.md)

Ce fichier est a la **racine du projet** et est **commite dans le repository**. Il contient les instructions partagees par toute l'equipe.

Mon Projet - Instructions Claude Code

Stack technique

- Backend : Node.js 22 + TypeScript 5.7
- Frontend : React 19 + Vite
- Base de donnees : PostgreSQL 17
- Tests : Vitest + Playwright

Conventions

- Utiliser des fonctions fleches pour les composants React
- Nommer les fichiers en kebab-case
- Un composant par fichier
- Tests dans un dossier __tests__ adjacent

Architecture

- Clean Architecture avec 4 couches
- Voir docs/architecture.md pour les details

Commandes

- ``npm run dev`` : Lancer le serveur de developpement
- ``npm run test`` : Lancer les tests
- ``npm run lint`` : Verifier la qualite du code

Cas d'usage : - Stack technique du projet - Conventions d'equipe - Commandes utiles - Liens vers la documentation

1.4 Niveau 3 : Detaille (./.claude/CLAUDE.md)

Ce fichier est dans le dossier `./.claude/` du projet. Il contient des instructions techniques detaillees, souvent plus longues.

Instructions detaillees

Architecture

- Voir `./.claude/rules/` pour les regles detaillees
- Voir `./.claude/references/` pour la documentation technique

Technologies supportees

Stack	Version	Patterns
-----	-----	-----
React	19.x	Hooks, Zustand

Commandes disponibles

- `/workflow:init` - Initialiser le workflow
- `/team:audit` - Lancer un audit

1.5 Priorite de chargement

Les trois niveaux sont charges dans l'ordre et se **completent** (pas de remplacement) :

- | | | |
|-------------------------------------|------------|---------------------|
| 1. <code>~/.claude/CLAUDE.md</code> | (global) | → Charge en premier |
| 2. <code>./CLAUDE.md</code> | (projet) | → Complete |
| 3. <code>./.claude/CLAUDE.md</code> | (detaille) | → Complete |

Tous les contenus sont concatenes et envoyes a Claude comme contexte initial.

1.6 Bonnes pratiques de taille

Niveau	Taille recommandee	Contenu
Global	< 50 lignes	Preferences cles
Projet	< 100 lignes	Stack + conventions
Detaille	< 200 lignes	Resume + liens vers rules/

Regle d'or : Chaque ligne supplementaire dans CLAUDE.md **dilue l'attention** sur les instructions existantes. Gardez-le concis et utilisez `./.claude/rules/` pour les details.

2. CLAUDE.local.md : Instructions privées

2.1 Concept

CLAUDE.local.md est un fichier d'instructions **non commite** (ajoute au .gitignore). Il permet de définir des instructions personnelles spécifiques à un projet sans affecter les autres membres de l'équipe.

2.2 Emplacement

```
projet/
├── CLAUDE.md           ← Commite (equipe)
├── CLAUDE.local.md     ← Non commite (personnel)
└── .claude/
    ├── CLAUDE.md       ← Commite (equipe)
    └── CLAUDE.local.md ← Non commite (personnel)
```

2.3 Cas d'usage

Instructions locales (CLAUDE.local.md)

Mon environnement

- J'utilise Docker Desktop sur macOS
- Mon éditeur est VS Code avec Vim bindings
- Mon terminal est iTerm2 avec Zsh

Chemins locaux

- Le projet tourne sur http://localhost:3000
- La base de données est sur localhost:5432
- Les logs sont dans ~/logs/mon-projet/

Préférences personnelles

- Je préfère voir les diffs complets avant commit
- Toujours me demander avant de supprimer des fichiers
- Afficher les commandes Docker avant de les exécuter

2.4 Ne pas oublier le .gitignore

.gitignore

CLAUDE.local.md

.claude/CLAUDE.local.md

3. Settings.json : Configuration avancée

3.1 Les 3 niveaux de settings

Claude Code utilise un système de configuration JSON à 3 niveaux :

Niveau	Emplacement	Portée	Qui le gère
Global	~/.claude/settings.json	Tous les projets	Développeur
Projet	.claude/settings.json	Ce projet	Équipe (commite)
Entreprise	/etc/claude/settings.json	Toute la machine	Admin système

3.2 Priorité

La configuration est fusionnée avec la priorité suivante (du plus prioritaire au moins) :

1. Entreprise (/etc/claude/settings.json) → Priorité max
2. Projet (.claude/settings.json) → Priorité moyenne
3. Global (~/.claude/settings.json) → Priorité basse

Les niveaux supérieurs **écrasent** les niveaux inférieurs pour les mêmes clés.

3.3 Structure du fichier

```
{
  "permissions": {
    "allow": [],
    "deny": []
  },
  "env": {
    "VARIABLE": "valeur"
  }
}
```

3.4 Exemple complet : settings.json projet

```
{
  "permissions": {
    "allow": [
      "Bash(docker*)",
      "Bash(npm*)",
      "Bash(git*)",
      "Write(src/**)",
      "Write(tests/**)"
    ]
  }
}
```



```
    ],
    "deny": [
      "Bash(rm -rf*)",
      "Bash(sudo*)",
      "Write(.env*)",
      "Write(*.key)"
    ]
  },
  "env": {
    "NODE_ENV": "development",
    "LOG_LEVEL": "debug"
  }
}
```

3.5 Exemple : settings.json global

```
{
  "permissions": {
    "allow": [
      "Bash(git status)",
      "Bash(git diff*)",
      "Bash(git log*)"
    ],
    "deny": [
      "Bash(git push --force*)",
      "Bash(git reset --hard*)"
    ]
  }
}
```

3.6 Exemple : settings.json entreprise

```
{
  "permissions": {
    "deny": [
      "Bash(curl*)",
      "Bash(wget*)",
      "Bash(ssh*)",
      "Write(/etc/**)",
      "Write(/usr/**)"
    ]
  }
}
```

4. Permissions 3-tier

4.1 Les trois niveaux de permission

Claude Code utilise un systeme de permissions a 3 niveaux pour controler ce que l'agent peut faire :

Niveau	Comportement	Usage
Ask	Demande confirmation avant chaque action	Default pour les operations sensibles
Allow	Execute sans confirmation	Operations de confiance
Deny	Bloque completement l'action	Operations dangereuses

4.2 Syntaxe des regles de permission

Les permissions utilisent une syntaxe basee sur le nom de l'outil et un pattern glob :

Outil(pattern)

Outils disponibles :

Outil	Description	Exemples
Bash	Execution de commandes shell	Bash(docker*), Bash(npm test)
Write	Ecriture de fichiers	Write(src/**), Write(*.md)
Read	Lecture de fichiers	Read(.env)
Edit	Modification de fichiers	Edit(src/**)

4.3 Exemples de configuration

Autoriser les commandes Docker sans confirmation

```
{
  "permissions": {
    "allow": [
      "Bash(docker*)",
      "Bash(docker compose*)"
    ]
  }
}
```

Interdire les operations dangereuses

```
{
  "permissions": {
    "deny": [
      "Bash(rm -rf /)",
      "Bash(sudo*)",
      "Bash(chmod 777*)",
      "Write(.env*)",
      "Write(*.pem)",
      "Write(*.key)"
    ]
  }
}
```

Configuration complete et recommandee

```
{
  "permissions": {
    "allow": [
      "Bash(docker*)",
      "Bash(npm*)",
      "Bash(npx*)",
      "Bash(git status*)",
      "Bash(git diff*)",
      "Bash(git log*)",
      "Bash(git add*)",
      "Bash(ls*)",
      "Bash(cat*)",
      "Bash(find*)",
      "Bash(grep*)",
      "Write(src/**)",
      "Write(tests/**)",
      "Write(docs/**)"
    ],
    "deny": [
      "Bash(rm -rf*)",
      "Bash(sudo*)",
      "Bash(git push --force*)",
      "Bash(git reset --hard*)",
      "Write(.env*)",
      "Write(*.key)",
      "Write(*.pem)",
      "Write(*.secret)"
    ]
  }
}
```

4.4 Wildcards

Les patterns supportent les wildcards standard :

Pattern	Signification	Exemple
*	N'importe quoi (1 niveau)	Bash(docker*) → docker ps, docker build
**	N'importe quoi (multi-niveaux)	Write(src/**) → src/a/b/c.ts
?	Un caractere	Write(*.t?) → .ts, .tx

5. .claudeignore : Exclure des fichiers

5.1 Concept

Le fichier `.claudeignore` fonctionne exactement comme `.gitignore` : il indique a Claude Code les fichiers et dossiers a **ignorer complètement**. Ces fichiers ne seront pas lus, indexes ou inclus dans le contexte.

5.2 Emplacement

```
projet/  
├── .claudeignore      ← A la racine du projet  
├── .gitignore  
└── src/
```

5.3 Syntaxe

La syntaxe est identique a `.gitignore` :

```
# Dependances  
node_modules/  
vendor/  
.venv/  
  
# Build  
dist/  
build/
```

```
.next/

# Fichiers volumineux inutiles pour le contexte
*.min.js
*.min.css
*.map
*.lock

# Donnees
*.sql
*.dump
*.sqlite

# Medias
*.png
*.jpg
*.gif
*.mp4
*.pdf

# Secrets (par securite)
.env
.env.*
*.key
*.pem
credentials/

# IDE
.idea/
.vscode/
*.swp
*.swo

# Logs
logs/
*.log
```

5.4 Pourquoi c'est important ?

Chaque fichier lu par Claude Code consomme des tokens de la context window. Exclure les fichiers non pertinents : - **Economise des tokens** (et donc des couts) - **Ameliore la pertinence** des reponses - **Accelere** le traitement - **Protege** les donnees sensibles

5.5 Recommandation

Commencez avec un `.claudeignore` genereux et ajustez si Claude manque d'informations :

```
# Commencer restrictif
```

```
node_modules/  
dist/  
build/  
*.min.*  
*.map  
*.lock  
*.log
```

6. Les @ references

6.1 Concept

Les **@ references** permettent d'injecter du contexte supplémentaire dans votre prompt en referencant des fichiers, des dossiers ou des URLs.

6.2 Syntaxe

Reference	Syntaxe	Description
Fichier	@fichier.ts	Injecte le contenu du fichier
Dossier	@src/components/	Injecte l'arborescence du dossier
URL	@https://exemple.com/fichier	Injecte le contenu de la page web

6.3 Exemples pratiques

Referer un fichier spécifique

> Refactorise @src/services/auth.ts en suivant le pattern Strategy

Claude recevra le contenu complet de auth.ts dans son contexte.

Referer un dossier

> Analyse l'architecture de @src/components/ et propose des améliorations

Claude recevra l'arborescence des fichiers du dossier.

Referer une documentation en ligne

> Implemente un composant selon

↪ @https://ui.shadcn.com/docs/components/button

Claude télécharge et lit le contenu de la page web.

6.4 Utilisation dans CLAUDE.md

Les @ references sont également utilisables dans les fichiers CLAUDE.md pour pointer vers de la documentation :

```
# Mon Projet
```

```
## Architecture
```

Voir @docs/architecture.md pour l'architecture détaillée.

```
## References
```

- API : @docs/api-reference.md
- Conventions : @docs/conventions.md

6.5 Cas d'usage avancés

```
# Comparer deux fichiers
```

```
> Compare @src/old-service.ts et @src/new-service.ts
```

```
# Utiliser une spec comme reference
```

```
> Implemente @docs/specs/user-story-42.md
```

```
# Injecter une config
```

```
> Deploie selon @docker-compose.prod.yml
```

7. Structure recommandée d'un projet

7.1 Arborescence complète

```

projet/
├── .claude/
│   ├── CLAUDE.md           ← Instructions détaillées (equipe)
│   ├── CLAUDE.local.md     ← Instructions privées (personnel)
│   ├── settings.json       ← Permissions projet (equipe)
│   └── rules/              ← Regles détaillées par sujet
│       ├── 01-architecture.md
│       ├── 02-conventions.md
│       └── 03-security.md
├── .claudeignore            ← Fichiers exclus du contexte
├── CLAUDE.md               ← Instructions projet (equipe)
├── CLAUDE.local.md         ← Instructions privées (personnel)
├── .gitignore
└── src/

```

7.2 Fichiers commites vs non commites

Fichier	Commite?	Partage equipe?
CLAUDE.md	Oui	Oui
.claude/CLAUDE.md	Oui	Oui
.claude/settings.json	Oui	Oui
.claude/rules/*.md	Oui	Oui
.claudeignore	Oui	Oui
CLAUDE.local.md	Non	Non
.claude/CLAUDE.local.md	Non	Non

Exercice pratique : Configurer un CLAUDE.md optimal (30 min)

Etape 1 : Créer la structure (5 min)

```
# Créer un projet d'exercice
mkdir exercice-claude-config && cd exercice-claude-config
git init
```

```
# Créer la structure .claude
mkdir -p .claude/rules
```

```
# Créer les fichiers de base
touch CLAUDE.md
touch .claude/CLAUDE.md
touch .claude/settings.json
touch .claudeignore
```

Etape 2 : Rédiger le CLAUDE.md projet (10 min)

Ouvrez CLAUDE.md et rédigez les instructions pour un projet fictif (par exemple une API REST en Node.js) :

```
# Mon API REST

## Stack
- Node.js 22 + TypeScript 5.7
- Express.js 5
- PostgreSQL 17 + Prisma ORM
- Vitest pour les tests
```


Conventions

- Architecture Clean Architecture (4 couches)
- Nommage fichiers : kebab-case
- Nommage variables : camelCase
- Tests : fichier.test.ts adjacent au fichier source

Commandes

- ``npm run dev`` : Serveur de developpement
- ``npm run test`` : Tests unitaires
- ``npm run lint`` : Linting ESLint
- ``npm run build`` : Build production

Etape 3 : Configurer les permissions (5 min)

Ouvrez `.claude/settings.json`:

```
{
  "permissions": {
    "allow": [
      "Bash(npm*)",
      "Bash(npx*)",
      "Bash(git*)",
      "Write(src/**)",
      "Write(tests/**)"
    ],
    "deny": [
      "Bash(rm -rf*)",
      "Bash(sudo*)",
      "Write(.env*)"
    ]
  }
}
```

Etape 4 : Créer le `.claudeignore` (5 min)

```
node_modules/
dist/
build/
*.min.*
*.map
*.lock
*.log
.env
.env.*
```

Etape 5 : Tester avec Claude Code (5 min)

Lancer Claude Code dans le projet

claude

Verifier que les instructions sont chargees

> Quel est le stack technique de ce projet ?

Verifier les permissions

> Essaie de lire le fichier .env

Points Cles a Retenir

1. **3 niveaux de CLAUDE.md** : global (personnel), projet (equipe), detaille (technique)
 2. **CLAUDE.local.md** pour les instructions privees non commitees
 3. **settings.json a 3 niveaux** : global, projet, entreprise (priorite croissante)
 4. **Permissions 3-tier** : Ask (default), Allow (confiance), Deny (bloque)
 5. **Wildcards** dans les permissions : Bash (docker*), Write (src/**)
 6. **.claudeignore** pour exclure les fichiers inutiles du contexte
 7. **@ references** pour injecter du contexte : @fichier, @dossier/, @https://url
 8. **Garder CLAUDE.md concis** : chaque ligne supplementaire dilue l'attention
-

Prochain module : Module 3 - Patterns de Travail