

Module 7 — Multi-Agent et Coordination

Jour 2 — Agent Teams, worktrees

The Bearded Bear

Février 2026



Table des matières

Module 7 : Multi-Agent et Coordination (1h15)	3
Objectifs	3
1. Agent Teams (Research Preview)	3
Concept	3
Architecture	3
Outils Agent Teams	4
SendMessage : modes de communication	4
Workflow complet	4
Leader-Teammates : coordination	5
Etat idle	5
2. Sub-Agents Paralleles	5
Le Task Tool	5
Execution parallele	5
run_in_background	6
Fork pour recherche	6
Metriques du Task tool	7
3. Git Worktrees	7
Concept	7
Le flag -w	7
Setup manuel	7
Sessions paralleles	8
Nettoyage	8
Recommandations	8
4. Pattern Writer/Reviewer	8
Principe	8
Implementation avec worktrees	9
Implementation avec Agent Teams	9
Avantages	10
5. Pattern Fan-Out	10
Principe	10
Cas d'usage	10
Implementation avec Task tool	10
Implementation avec Agent Teams	11
Fan-out avec worktrees	11
6. Pattern Interview	11
Principe	11
Cas d'usage	12
Implementation	12
Interview structuree pour onboarding	12
7. Arbre de Decision : Quel Pattern Utiliser?	13
Decision tree	13

Tableau comparatif	14
Regles de choix	14
Exercice Pratique (20min)	14
Partie 1 : Sub-agent de recherche (10min)	14
Partie 2 : Pattern Writer/Reviewer simule (10min)	15
Verification	15
Points Cles a Retenir	15

Module 7 : Multi-Agent et Coordination (1h15)

Objectifs

A la fin de ce module, vous serez capable de : - Comprendre et utiliser le systeme Agent Teams de Claude Code - Coordonner des agents leader et teammates pour des taches complexes - Exploiter les sub-agents paralleles avec le Task tool - Mettre en place des sessions paralleles avec git worktrees - Appliquer les patterns Writer/Reviewer, Fan-out et Interview - Choisir le bon pattern multi-agent selon le contexte

1. Agent Teams (Research Preview)

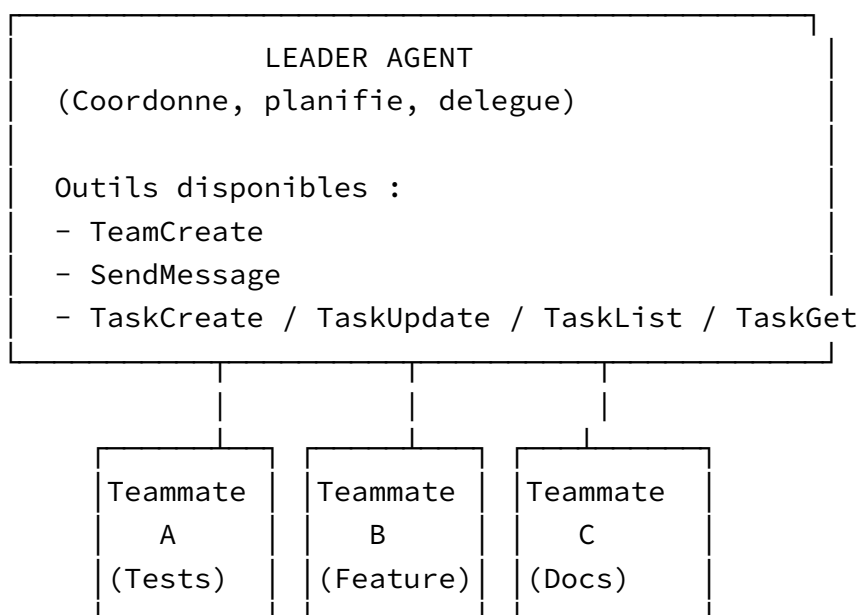
Activation require : Agent Teams est une fonctionnalite experimentale. Pour l'activer :

```
export CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1
```

Concept

Agent Teams est un systeme de coordination multi-agents introduit dans Claude Code v2.1.32. Il permet de creer une equipe d'agents qui collaborent sur une tache complexe, avec un leader qui coordonne et des teammates qui executent.

Architecture



Outils Agent Teams

Outil	Description	Utilise par
TeamCreate	Créer une équipe avec un nom	Leader
SendMessage	Envoyer un message à un teammate ou broadcast	Leader / Teammates
TaskCreate	Créer une tâche dans la liste partagée	Leader
TaskUpdate	Mettre à jour le statut d'une tâche	Leader / Teammates
TaskList	Lister toutes les tâches de l'équipe	Tous
TaskGet	Obtenir les détails d'une tâche spécifique	Tous

SendMessage : modes de communication

SendMessage modes :

- `message(teammate_name, content)` -> Message direct à un teammate
- `broadcast(content)` -> Message à tous les teammates
- `shutdown_request(teammate_name)` -> Demander à un teammate de s'arrêter

Workflow complet

Étape 1 : Créer l'équipe

```
Leader -> TeamCreate("feature-notifications")
```

Étape 2 : Créer les tâches

```
Leader -> TaskCreate("Ecrire les tests unitaires", assignee: "tests-agent")
```

```
Leader -> TaskCreate("Implémenter le service", assignee: "feature-agent")
```

```
Leader -> TaskCreate("Ecrire la documentation", assignee: "docs-agent")
```

Étape 3 : Spawner les teammates

```
Leader -> Task tool (team_name: "feature-notifications")  
-> Cree 3 sous-agents avec accès à la liste de tâches
```

Étape 4 : Assigner et coordonner

```
Leader -> SendMessage("tests-agent", "Commence par les tests du NotificationService")
```

```
Leader -> SendMessage("feature-agent", "Attends les tests avant d'implémenter")
```

Etape 5 : Les teammates travaillent

```
tests-agent -> TaskUpdate(task_id, status: "in_progress")
tests-agent -> ... ecrit les tests ...
tests-agent -> TaskUpdate(task_id, status: "done")
tests-agent -> SendMessage("leader", "Tests termines")
```

Etape 6 : Coordination

```
Leader -> SendMessage("feature-agent", "Les tests sont prêts, tu peux implementer")
```

Etape 7 : Shutdown

```
Leader -> shutdown_request("tests-agent")
Leader -> shutdown_request("feature-agent")
Leader -> shutdown_request("docs-agent")
```

Leader-Teammates : coordination

Le **Leader** est l'agent principal qui : - Cree l'equipe et les taches - Coordonne le travail entre teammates
- Decide quand un teammate doit s'arreter - Aggrege les resultats

Les **Teammates** sont des sous-agents qui : - Recoivent des instructions via SendMessage - Travaillent sur leurs taches assignees - Mettent a jour le statut via TaskUpdate - Entrent en etat **idle** quand ils n'ont plus de travail - Repondent au leader via SendMessage

Etat idle

Un teammate entre en etat **idle** quand il a termine sa tache et attend de nouvelles instructions. Le leader peut alors : - Lui assigner une nouvelle tache - Lui envoyer un shutdown_request - Lui demander de revoir le travail d'un autre teammate

2. Sub-Agents Paralleles

Le Task Tool

Le **Task tool** est le mecanisme de base pour creer des sous-agents. Chaque sous-agent a sa propre fenetre de contexte, ce qui permet de : - Isoler les investigations du contexte principal - Executer plusieurs taches en parallele - Eviter la pollution du contexte

Execution parallele

Agent principal

```

|
|-- Task("Analyse le module auth") -----> Sub-agent 1
|
|-- Task("Analyse le module orders") ---> Sub-agent 2
|
|-- Task("Analyse le module payments") -> Sub-agent 3
|
|   (attend les resultats)
|
v
Synthese des 3 analyses

```

run_in_background

Le flag `run_in_background` permet de lancer un sous-agent sans attendre son resultat :

Utilisation :

```
Task(description, run_in_background: true)
```

Le sous-agent s'exécute en arrière-plan.

Le resultat est disponible via `/tasks` quand termine.

Fork pour recherche

Le pattern **fork** est idéal pour les recherches exploratoires. Plutôt que de polluer le contexte principal avec des dizaines de fichiers, on fork la recherche dans un sous-agent :

Agent principal :

```
"Je dois trouver comment l'authentification est implementee"
```

```
-> Task("Explore le codebase et identifie :
      1. Les fichiers lies a l'authentification
      2. Les patterns utilises (JWT, sessions, etc.)
      3. Les middlewares de securite
      4. Les tests existants
      Retourne un resume structure.")
```

```
<- Resultat : resume de 20 lignes
      (au lieu de lire 30 fichiers dans le contexte principal)
```

Metriques du Task tool

Depuis v2.1.30, le Task tool retourne des metriques : - **Token count** : nombre de tokens utilises par le sous-agent - **Tool uses** : nombre d'outils utilises - **Duration** : temps d'exécution

Ces metriques aident a optimiser l'utilisation des sous-agents.

3. Git Worktrees

Concept

Les **git worktrees** permettent d'avoir plusieurs copies de travail d'un meme repository, chacune sur une branche differente. Combine avec Claude Code, cela permet de travailler sur plusieurs taches simultanement avec des sessions independantes.

Le flag -w

Note : Le flag -w n'est pas confirme dans la documentation officielle Claude Code 2.1.45. Utilisez le setup manuel ci-dessous pour une approche garantie.

```
# Si le flag -w est supporte dans votre version :  
claude -w feature/auth
```

```
# Equivalent a (methode manuelle recommandee) :  
git worktree add ../feature-auth feature/auth  
cd ../feature-auth  
claude
```

Setup manuel

```
# Creer un worktree pour une feature  
git worktree add ../project-feature-auth feature/auth
```

```
# Creer un worktree pour la review  
git worktree add ../project-review-auth feature/auth
```

```
# Lister les worktrees  
git worktree list
```

```
# Resultat :  
# /home/user/project          main          [main]  
# /home/user/project-feature-auth feature/auth  
# /home/user/project-review-auth feature/auth
```


Sessions paralleles

Chaque worktree peut avoir sa propre session Claude Code independante :

Terminal 1 :

```
cd ../project-feature-auth
claude "Implements le systeme d'authentification JWT"
```

Terminal 2 :

```
cd ../project-review-auth
claude "Revois le code d'authentification et identifie les problemes"
```

Terminal 3 :

```
cd ../project (main)
claude "Corrige le bug #456 sur le module de paiement"
```

Nettoyage

```
# Supprimer un worktree
git worktree remove ../project-feature-auth
```

```
# Supprimer tous les worktrees orphelins
git worktree prune
```

Recommandations

Pratique	Recommandation
Nombre maximum	3-5 worktrees simultanees
Scope	Un worktree = une tache
Duree de vie	Supprimer des que la tache est terminee
Partage	Ne pas partager de sessions entre worktrees
Nommage	../projet-but (ex: ../project-feature-auth)

4. Pattern Writer/Reviewer

Principe

Le pattern **Writer/Reviewer** utilise deux agents en parallele pour produire du code de meilleure qualite. Un agent ecrit le code, un autre le relit. La separation evite le **biais d'auteur** : le reviewer n'a pas

le contexte mental du writer et peut identifier des problemes que le writer ne verrait pas.

Implementation avec worktrees

Terminal 1 (Writer) :

```
cd ../project-feature
claude "Implements la fonctionnalité de notifications push.
Critères :
- Service NotificationService avec interface
- Support email et push
- Tests unitaires complets
- Documentation API"
```

Terminal 2 (Reviewer) :

```
cd ../project-review
claude "Relis le code du module notifications et effectue une review.
Vérifie :
- Respect SOLID
- Couverture de tests
- Sécurité
- Performance
- Lisibilité
Propose des corrections si nécessaire."
```

Implementation avec Agent Teams

Leader :

```
TeamCreate("feature-review")

TaskCreate("Implementer NotificationService", assignee: "writer")
TaskCreate("Review du code notifications", assignee: "reviewer")

SendMessage("writer", "Implemente le service de notifications")

# Quand le writer a termine :
SendMessage("reviewer", "Le code est pret, effectue une review du module notificat

# Le reviewer retourne ses commentaires
# Le leader peut demander au writer de corriger
SendMessage("writer", "Le reviewer a trouve ces problemes : [...]. Corrige-
les.")
```

Avantages

- Pas de biais d'auteur : le reviewer decouvre le code sans a priori
 - Deux perspectives differentes sur le meme probleme
 - Detection de bugs plus efficace
 - Documentation et lisibilite ameliorees
-

5. Pattern Fan-Out

Principe

Le pattern **Fan-out** distribue une meme operation sur plusieurs cibles en parallele. C'est ideal pour les operations batch qui doivent s'appliquer a de nombreux fichiers ou modules.

Cas d'usage

Situation	Fan-out sur
Refactoring multi-fichiers	Chaque fichier a modifier
Migration de schema	Chaque entite a migrer
Rename across codebase	Chaque occurrence a modifier
Audit multi-modules	Chaque module a auditer
Generation de tests	Chaque service sans tests

Implementation avec Task tool

Agent principal :

```
"Je dois renommer UserService en AccountService partout dans le projet"
```

```
-> Task("Renomme UserService en AccountService dans le module auth/")
-> Task("Renomme UserService en AccountService dans le module orders/")
-> Task("Renomme UserService en AccountService dans le module payments/")
-> Task("Mets a jour les tests pour utiliser AccountService")
-> Task("Mets a jour la documentation")
```

```
<- Resultats : chaque sous-agent traite son module
    Le principal verifie la coherence globale
```

Implementation avec Agent Teams

Leader :

```
TeamCreate("migration-rename")

# Creer les taches pour chaque module
for module in [auth, orders, payments, tests, docs]:
    TaskCreate("Rename UserService -> AccountService in $module")

# Spawner un teammate par module
# Chaque teammate traite son module independamment

# A la fin, le leader verifie la coherence
broadcast("Verification terminee ?")
```

Fan-out avec worktrees

Pour des operations sur des branches differentes :

```
# Creer des worktrees pour chaque module
git worktree add ../project-auth feature/rename-auth
git worktree add ../project-orders feature/rename-orders
git worktree add ../project-payments feature/rename-payments

# Terminal 1
cd ../project-auth
claude "Renomme UserService en AccountService dans ce module"

# Terminal 2
cd ../project-orders
claude "Renomme UserService en AccountService dans ce module"

# Terminal 3
cd ../project-payments
claude "Renomme UserService en AccountService dans ce module"
```

6. Pattern Interview

Principe

Le pattern **Interview** utilise Claude Code pour collecter des informations de maniere structuree aupres de l'utilisateur. L'outil AskUserQuestion permet a Claude de poser des questions et d'attendre les reponses avant de continuer.

Cas d'usage

Situation	Type d'interview
Onboarding nouveau projet	Configuration initiale
Scaffolding de feature	Specifications techniques
Configuration environnement	Parametres specifiques
Collecte de requirements	User stories

Implementation

Claude :

"Je vais configurer votre projet. Quelques questions :"

AskUserQuestion("Quel est le nom de votre projet ?")

-> Reponse : "e-commerce-api"

AskUserQuestion("Quel framework utilisez-vous ?")

-> Reponse : "FastAPI"

AskUserQuestion("Avez-vous besoin d'une base de donnees ?

1. PostgreSQL

2. MySQL

3. SQLite

4. MongoDB")

-> Reponse : "1"

AskUserQuestion("Quelles fonctionnalites voulez-vous ?

- [] Authentification JWT

- [] Upload de fichiers

- [] Envoi d'emails

- [] WebSockets")

-> Reponse : "JWT et emails"

Claude genere le projet avec toutes les informations collectees

Interview structuree pour onboarding

Scenario : Nouveau membre de l'equipe

Claude :

"Bienvenue ! Je vais t'aider a configurer ton environnement."

```
AskUserQuestion("Quel est ton role ? (dev backend, dev frontend, fullstack, devops  
-> "dev backend"
```

```
AskUserQuestion("Quel est ton editeur prefere ? (VS Code, JetBrains, Vim, autre)")  
-> "VS Code"
```

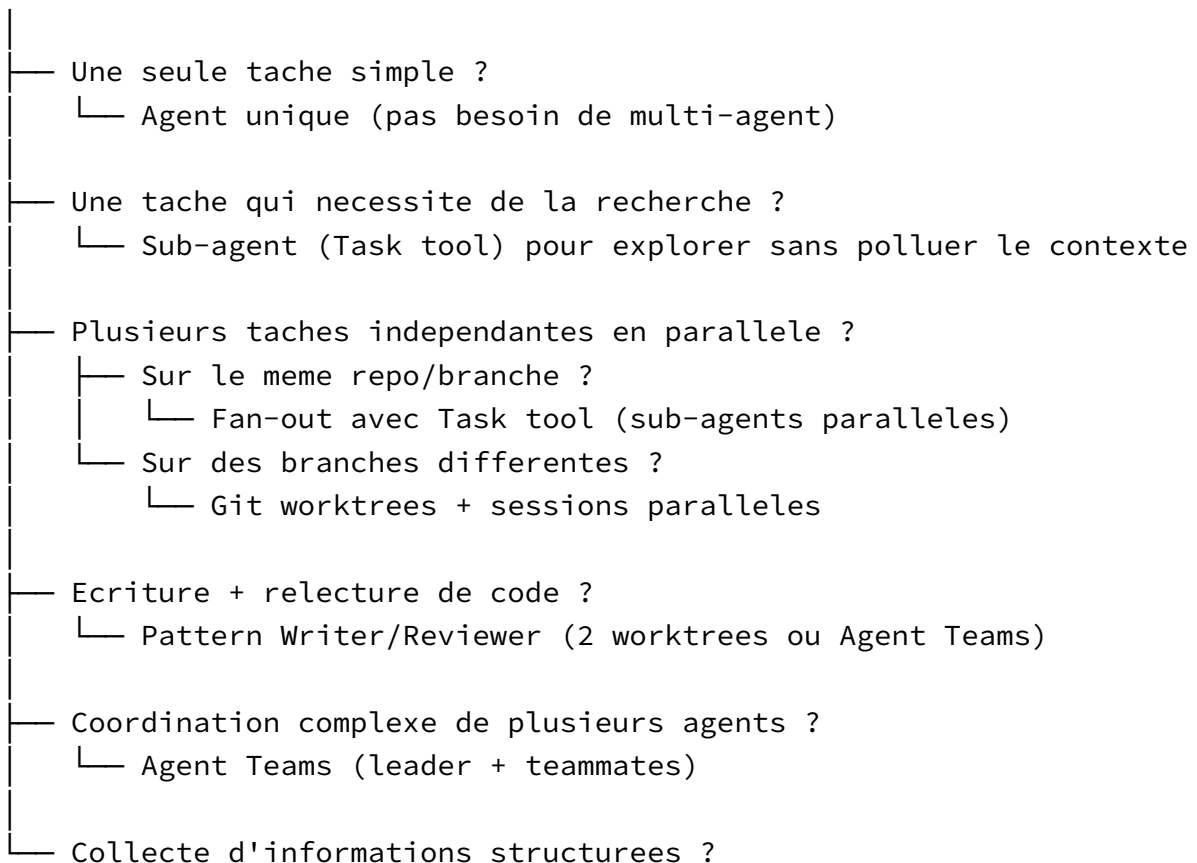
```
AskUserQuestion("As-tu Docker installe ? (oui/non)")  
-> "oui"
```

```
# Claude configure l'environnement en fonction des reponses  
# - Extensions VS Code recommandees  
# - Configuration Docker  
# - Acces aux services backend
```

7. Arbre de Decision : Quel Pattern Utiliser ?

Decision tree

Votre tache est-elle...



└─ Pattern Interview (AskUserQuestion)

Tableau comparatif

Pattern	Complexite	Agents	Parallelisme	Cas d'usage
Agent unique	Faible	1	Non	Bug fix, petite feature
Sub-agent	Faible	1+N	Oui	Exploration, recherche
Fan-out	Moyenne	1+N	Oui	Operations batch, migration
Writer/Reviewer	Moyenne	2	Oui	Feature + review
Agent Teams	Haute	1+N	Oui	Projet complexe, coordination
Interview	Faible	1	Non	Onboarding, configuration
Worktrees	Moyenne	N	Oui	Taches sur branches differentes

Regles de choix

1. **Commencer simple** : agent unique jusqu'a ce que la complexite l'exige
2. **Sub-agents pour la recherche** : toujours deleguer les explorations
3. **Worktrees pour le parallelisme** : des que 2+ taches independantes
4. **Agent Teams pour la coordination** : quand les taches ont des dependances
5. **Fan-out pour le batch** : operations repetitives sur N cibles
6. **Writer/Reviewer pour la qualite** : features importantes ou critiques

Exercice Pratique (20min)

Partie 1 : Sub-agent de recherche (10min)

Demandez a Claude Code d'analyser un projet en utilisant des sub-agents :

Utilise des sous-agents pour analyser ce projet en parallele :

- Sub-agent 1 : Identifie l'architecture et les patterns utilises
- Sub-agent 2 : Liste les dependances et leurs versions
- Sub-agent 3 : Trouve les fichiers sans tests correspondants

Synthetise les resultats des 3 sous-agents.

Observez comment Claude cree les sub-agents et synthetise les resultats.

Partie 2 : Pattern Writer/Reviewer simule (10min)

Demandez a Claude de simuler le pattern Writer/Reviewer en une session :

Simule le pattern Writer/Reviewer :

1. En tant que "Writer", cree une fonction de validation d'email robuste
2. En tant que "Reviewer", critique cette implementation
3. En tant que "Writer", applique les corrections du reviewer
4. Montre le resultat final

Verification

- ☐ Les sub-agents sont crees et executes en parallele
 - ☐ Chaque sub-agent retourne un resultat specifique
 - ☐ La synthese est coherente et complete
 - ☐ Le pattern Writer/Reviewer produit du code de meilleure qualite
-

Points Cles a Retenir

1. **Agent Teams** = coordination structuree avec leader et teammates
 2. **Sub-agents (Task tool)** = isolation du contexte + parallelisme
 3. **Git worktrees** = sessions independantes sur des branches differentes
 4. **Writer/Reviewer** = meilleure qualite grace a la relecture croisee
 5. **Fan-out** = operations batch parallelisees sur N cibles
 6. **Interview** = collecte structuree d'informations avec AskUserQuestion
 7. **Commencer simple** : un agent unique suffit pour la majorite des taches
 8. **Le contexte est la ressource critique** : les sub-agents le protegent
-

Duree : 1h15 **Prochain module** : Module 8 : Qualite et Securite