

## Module 3 — Patterns de Travail

Jour 1 — Prompt, Plan Mode, sessions

The Bearded Bear

Février 2026



## Table des matières

<b>Module 3 : Patterns de Travail</b>	<b>3</b>
Objectifs . . . . .	3
1. Prompt engineering pour Claude Code . . . . .	3
1.1 Principes fondamentaux . . . . .	3
1.2 Structure d'un bon prompt . . . . .	3
1.3 L'itérativité . . . . .	4
1.4 Exemples bons vs mauvais . . . . .	4
2. Plan Mode . . . . .	5
2.1 Concept . . . . .	5
2.2 Activation . . . . .	5
2.3 Workflow du Plan Mode . . . . .	6
2.4 Quand utiliser le Plan Mode? . . . . .	6
2.5 Exemple pratique . . . . .	6
3. Gestion du contexte . . . . .	7
3.1 La context window . . . . .	7
3.2 /clear : Nettoyer le contexte . . . . .	7
3.3 /compact : Compacter le contexte . . . . .	8
3.4 Compaction automatique . . . . .	8
3.5 Hook SessionStart :compact . . . . .	8
3.6 Seuils d'action . . . . .	9
4. Sub-agents . . . . .	9
4.1 Concept . . . . .	9
4.2 Types de sub-agents . . . . .	9
4.3 Quand utiliser un sub-agent? . . . . .	10
4.4 run_in_background . . . . .	10
4.5 Exemple pratique . . . . .	10
5. Fast Mode . . . . .	11
5.1 Concept . . . . .	11
5.2 Activation . . . . .	11
5.3 Quand utiliser Fast Mode? . . . . .	11
6. Mode headless . . . . .	12
6.1 Concept . . . . .	12
6.2 Syntaxe de base . . . . .	12
6.3 Formats de sortie . . . . .	12
6.4 Piping et integration scripts . . . . .	12
6.5 Cas d'usage avancés . . . . .	13
7. Gestion des sessions . . . . .	13
7.1 Continuer une session . . . . .	13
7.2 Reprendre une session spécifique . . . . .	13
7.3 Renommer une session . . . . .	13

8. Checkpointing et rewind . . . . .	14
8.1 Concept . . . . .	14
8.2 Rewind . . . . .	14
8.3 Workflow de rewind . . . . .	14
8.4 Cas d'usage . . . . .	14
9. Support images . . . . .	15
9.1 Injection d'images . . . . .	15
9.2 Cas d'usage . . . . .	15
10. Sandboxing . . . . .	15
10.1 Concept . . . . .	15
10.2 Activation . . . . .	15
10.3 Ce que le sandbox empeche . . . . .	16
11. Keybindings et status line . . . . .	16
11.1 Raccourcis clavier principaux . . . . .	16
11.2 Status line . . . . .	16
12. Formats de sortie . . . . .	17
12.1 text (default) . . . . .	17
12.2 json . . . . .	17
12.3 stream-json . . . . .	17
Exercice pratique : Maitriser les patterns (30 min) . . . . .	17
Etape 1 : Prompt engineering (10 min) . . . . .	17
Etape 2 : Plan Mode (10 min) . . . . .	18
Etape 3 : Mode headless (10 min) . . . . .	18
Points Cles a Retenir . . . . .	18

## Module 3 : Patterns de Travail

**Duree :** 2h **Prerequis :** Module 2 - CLAUDE.md et Configuration **Prochain module :** Module 4 - Pratique Guidee

### Objectifs

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

1. Formuler des prompts efficaces pour Claude Code
2. Utiliser le Plan Mode pour explorer avant d'agir
3. Gerer la context window de maniere optimale
4. Deleger des taches aux sub-agents
5. Utiliser le mode headless pour l'automatisation
6. Gerer les sessions et le checkpointing
7. Exploiter toutes les fonctionnalites avancees de l'interface

## 1. Prompt engineering pour Claude Code

### 1.1 Principes fondamentaux

Claude Code est un agent : vos prompts doivent etre formulees comme des **objectifs** plutot que comme des instructions pas-a-pas. Laissez l'agent decider du comment.

❌ MAUVAIS (trop directif)

```
> Ouvre le fichier src/auth.ts, trouve la fonction login,  
> ajoute un parametre rememberMe de type boolean,  
> puis modifie le corps de la fonction pour creer un cookie  
> si rememberMe est true.
```

✅ BON (objectif clair avec contraintes)

```
> Ajoute une option "Se souvenir de moi" a la fonctionnalite  
> de login. Quand l'utilisateur coche cette option, la session  
> doit durer 30 jours au lieu de 24h. Utilise un cookie securise.
```

### 1.2 Structure d'un bon prompt

Un prompt efficace contient :

1. **Le contexte** : Qu'est-ce qui existe déjà ?
2. **L'objectif** : Que voulez-vous accomplir ?
3. **Les contraintes** : Quelles règles respecter ?
4. **Le résultat attendu** : Comment vérifier le succès ?

- > Notre API d'authentification utilise JWT (contexte).
- > Je veux ajouter un endpoint de refresh token (objectif).
- > Le refresh token doit expirer après 7 jours et être stocké
- > en base de données, pas en cookie (contraintes).
- > Ajoute des tests unitaires qui couvrent les cas nominaux
- > et les cas d'erreur (résultat attendu).

### 1.3 L'itérativité

Ne cherchez pas le prompt parfait du premier coup. Itérez :

# Iteration 1 : Demande initiale

- > Crée un service de notification par email

# Iteration 2 : Préciser

- > Utilise un template HTML plutôt que du texte brut

# Iteration 3 : Contrainte supplémentaire

- > Ajoute une file d'attente async pour ne pas bloquer la requête

# Iteration 4 : Tests

- > Ajoute des tests unitaires pour le service

### 1.4 Exemples bons vs mauvais

#### Pour du refactoring

❌ MAUVAIS

- > Refactorise ce code

✅ BON

- > Le fichier `src/services/order.ts` fait 500 lignes et gère à la fois
- > la validation, le calcul des prix et l'envoi d'emails. Découpe-le
- > en services distincts en suivant le principe SRP. Conserve les
- > tests existants et assure-toi qu'ils passent toujours.

### Pour du debugging

☒ MAUVAIS

> Ca marche pas, corrige

☒ BON

> L'endpoint POST /api/orders renvoie une erreur 500 quand le champ

> discount est null. Voici le stack trace : [coller le stack trace].

> Le comportement attendu est de traiter null comme 0% de remise.

### Pour de la generation

☒ MAUVAIS

> Fais un CRUD utilisateur

☒ BON

> Cree un CRUD complet pour l'entite User avec les champs :

> email (unique, valide), name (requis, 2-100 chars), role (enum: admin, user).

> Inclus la validation, les tests unitaires et la migration de base de donnees.

> Suis l'architecture Clean Architecture du projet.

---

## 2. Plan Mode

### 2.1 Concept

Le Plan Mode demande a Claude de **planifier** ses actions avant de les executer. Au lieu de modifier directement le code, Claude propose d'abord un plan que vous pouvez valider, modifier ou rejeter.

### 2.2 Activation

*# Via la commande slash*  
/plan

*# Via le raccourci clavier*  
Shift+Tab

*# Dans le prompt*

> Plan comment tu implementerais un systeme de cache Redis

## 2.3 Workflow du Plan Mode

PLAN MODE ACTIVE
<ol style="list-style-type: none"><li>1. Claude explore la codebase (lecture de fichiers, grep)</li><li>2. Claude propose un plan :<ul style="list-style-type: none"><li>- Fichiers a modifier</li><li>- Ordre des modifications</li><li>- Tests a ajouter</li><li>- Risques identifiés</li></ul></li><li>3. Vous validez / modifiez : "OK, proceed" "Modifie le point 3" "Ajoute aussi les tests E2E"</li><li>4. Claude execute le plan</li></ol>

## 2.4 Quand utiliser le Plan Mode ?

Situation	Plan Mode?
Bug simple, 1 fichier	Non
Renommer une variable	Non
Nouvelle feature (2-3 fichiers)	Recommande
Refactoring multi-fichiers	Oui
Changement d'architecture	Obligatoire
Impact incertain	Oui

## 2.5 Exemple pratique

```
# Activer le Plan Mode  
> /plan
```

```
# Poser la question
```

- > Comment implementer un systeme de pagination
- > pour notre API REST qui supporte cursor-based
- > et offset-based pagination ?

*# Claude repond avec un plan detaille*

*# Vous validez :*

- > Le plan me convient. Commence par les tests.

### 3. Gestion du contexte

#### 3.1 La context window

La context window (~200K tokens) est **LA ressource critique** de Claude Code. Tout ce que Claude “sait” doit tenir dans cette fenetre.

Context Window (~200K tokens)	
CLAUDE.md + Rules	~5K tokens
Historique conversation	~80K tokens
Fichiers lus par Claude	~40K tokens
Resultats de commandes	~20K tokens
Espace pour la reponse	~55K tokens

#### 3.2 /clear : Nettoyer le contexte

La commande /clear vide **toute** la conversation en cours. Utilisez-la entre deux taches non liees.

*# Apres avoir termine une tache*

/clear

*# Commencer une nouvelle tache avec un contexte propre*

- > Maintenant, travaillons sur le systeme de notifications

**Quand utiliser /clear :** - Entre deux taches non liees - Apres une longue investigation - Quand le contexte depasse 50% - Quand Claude commence a se repeter ou a confondre des choses

**Quand NE PAS utiliser /clear :** - Au milieu d’une tache en cours - Si le contexte precedent est necessaire - Juste apres avoir charge des fichiers pertinents



### 3.3 /compact : Compacter le contexte

La commande /compact **resume** la conversation existante au lieu de la supprimer. Claude garde l'essentiel mais libere de l'espace.

```
# La conversation devient longue mais le contexte est utile
/compact

# Claude resume automatiquement et continue
> Maintenant, passons a l'implementation du point 3 du plan
```

### 3.4 Compaction automatique

Quand le contexte approche les limites de la fenetre, Claude Code **compacte automatiquement** les messages anciens. Les messages recents sont preserves, les anciens sont resumes.

### 3.5 Hook SessionStart :compact

Vous pouvez configurer un hook pour re-injecter du contexte critique apres une compaction :

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "command": "cat .claude/context-essentials.md"
      }
    ]
  }
}
```

Le fichier .claude/context-essentials.md contient les informations critiques :

```
# Contexte essentiel

## Tache en cours
Implementer le systeme de pagination pour l'API REST.

## Decisions prises
- Cursor-based pagination par default
- Offset disponible en fallback
- Limite max : 100 items par page

## Fichiers cles
- src/middleware/pagination.ts
- src/types/pagination.ts
- tests/pagination.test.ts
```

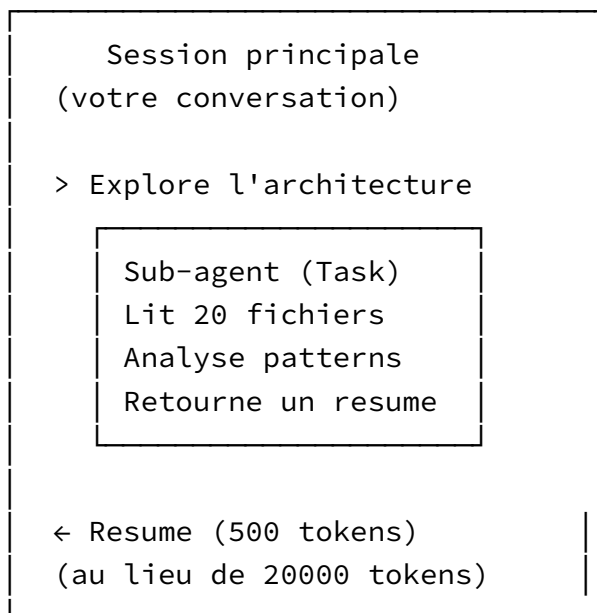
### 3.6 Seuils d'action

Contexte utilise	Action recommandee
< 30%	Normal, continuez
30-60%	Surveillez, evitez les lectures inutiles
60-80%	Deleguez aux sub-agents, envisagez /clear
> 80%	Compaction imminente, sauvegardez le contexte critique

## 4. Sub-agents

### 4.1 Concept

Les sub-agents sont des **sessions Claude independantes** lancees par votre session principale. Ils ont leur propre context window et ne polluent pas votre contexte principal.



### 4.2 Types de sub-agents

Type	Usage	Exemple
<b>Explore</b>	Investigation multi-fichiers	“Comment fonctionne l’auth dans ce projet?”

Type	Usage	Exemple
<b>Plan</b>	Planification d'implémentation	“Planifie l'ajout d'un cache Redis”
<b>General-purpose</b>	Tache independante	“Genere les tests pour ce module”

### 4.3 Quand utiliser un sub-agent ?

Situation	Action
Chercher un fichier precis	Glob/Grep directement
Lire 1-2 fichiers	Read directement
Explorer une architecture inconnue	Sub-agent Explore
Investigation multi-fichiers (> 3)	Sub-agent Explore
Planifier une implementation complexe	Sub-agent Plan
Tache independante en parallele	Sub-agent general

### 4.4 run\_in\_background

Les sub-agents peuvent s'exécuter en arrière-plan pendant que vous continuez à travailler :

```
# Claude lance un sub-agent en background
> Pendant que j'implemente le service, analyse les performances
> de la requete SQL principale en arriere-plan

# Vous continuez a travailler...
> Cree le service de notification

# Le resultat du sub-agent arrive quand il est pret
```

### 4.5 Exemple pratique

```
# Au lieu de lire 20 fichiers dans le contexte principal :
> Utilise un sub-agent pour explorer le systeme d'authentification
> de ce projet. Je veux savoir :
> - Quel framework d'auth est utilise
> - Ou sont stockes les tokens
> - Comment les permissions sont gerees
> - Y a-t-il des failles potentielles

# Claude delegate a un sub-agent qui :
# 1. Parcourt les fichiers auth/
```

```
# 2. Lit les configurations
# 3. Analyse les middlewares
# 4. Retourne un resume structure

# Votre contexte principal recoit uniquement le resume
```

---

## 5. Fast Mode

### 5.1 Concept

Le Fast Mode (/ fast) permet d'obtenir des reponses **2.5x plus rapides** avec Opus 4.6. Il utilise le meme modele mais avec une generation optimisee.

**Cout :** Le Fast Mode coute **6x le prix normal** (\$30 / \$150 par M tokens au lieu de \$5 / \$25). A utiliser avec parcimonie pour les taches ou la vitesse est critique.

### 5.2 Activation

```
# Toggle le Fast Mode
/fast

# La status line affiche "fast" quand actif
```

### 5.3 Quand utiliser Fast Mode ?

Situation	Fast Mode ?
Generation de code boilerplate	Oui
Taches repetitives	Oui
Modifications simples	Oui
Architecture complexe	Non (preferer la reflexion approfondie)
Debugging subtil	Non
Decisions critiques	Non

---

## 6. Mode headless

### 6.1 Concept

Le mode headless permet d'utiliser Claude Code de manière **non interactive**, directement depuis la ligne de commande ou dans des scripts.

### 6.2 Syntaxe de base

```
# Prompt direct avec resultat texte
claude -p "Liste les fichiers TypeScript du projet"

# Avec format de sortie specifique
claude -p "Analyse ce fichier" --output-format json
claude -p "Resume ce code" --output-format text
claude -p "Stream la reponse" --output-format stream-json
```

### 6.3 Formats de sortie

Format	Description	Usage
text	Texte brut	Lecture humaine, logs
json	JSON structure	Parsing programmatique
stream-json	JSON streamé ligne par ligne	Traitement temps réel

### 6.4 Piping et integration scripts

```
# Piping vers un fichier
claude -p "Genere un fichier de migration SQL" > migration.sql

# Piping depuis stdin
cat error.log | claude -p "Analyse ces logs et identifie le probleme"

# Integration dans un script bash
#!/bin/bash
ANALYSIS=$(claude -p "Analyse la qualite de $FILE" --output-format text)
echo "Resultat: $ANALYSIS"

# Chainer avec d'autres commandes
claude -p "Liste les TODO dans le code" --output-format text | grep
↵ "CRITICAL"
```

## 6.5 Cas d'usage avancés

*# Pre-commit hook*

```
claude -p "Verifie que les fichiers stages respectent les conventions" \  
  --output-format json
```

*# Revue de code automatique*

```
git diff main..HEAD | claude -p "Fais une revue de code de ce diff"
```

*# Generation de documentation*

```
claude -p "Genere la documentation JSDoc pour src/services/" \  
  --output-format text > docs/api.md
```

*# Analyse de logs*

```
tail -100 /var/log/app.log | claude -p "Y a-t-il des erreurs critiques ?"
```

---

## 7. Gestion des sessions

### 7.1 Continuer une session

*# Reprendre la derniere session*

```
claude --continue
```

*# Raccourci*

```
claude -c
```

Cela restaure le contexte de la derniere conversation et vous permet de continuer exactement ou vous en etiez.

### 7.2 Reprendre une session specifique

*# Lister les sessions*

```
claude --history
```

*# Reprendre une session par son ID*

```
claude --resume <session-id>
```

### 7.3 Renommer une session

*# Dans une session active*

```
/rename "Implementation pagination API"
```

Les sessions nommees sont plus faciles a retrouver dans l'historique.

---

## 8. Checkpointing et rewind

### 8.1 Concept

Claude Code cree automatiquement des **checkpoints** a chaque action majeure (ecriture de fichier, execution de commande). Vous pouvez revenir a n'importe quel checkpoint precedent.

### 8.2 Rewind

*# Revenir au dernier checkpoint*

Esc + Esc

*# Ou via la commande*

/rewind

### 8.3 Workflow de rewind

```
Checkpoint 1 : Code initial
  ↓
Checkpoint 2 : Tests ajoutes
  ↓
Checkpoint 3 : Service implemente
  ↓
Checkpoint 4 : Bug introduit !
  ↓
Esc+Esc → Retour Checkpoint 3
  ↓
Checkpoint 5 : Correction + avance
```

### 8.4 Cas d'usage

- Claude a pris une mauvaise direction → Rewind
- Une modification a casse les tests → Rewind
- Vous voulez essayer une approche differente → Rewind
- Claude a modifie un fichier qu'il ne devait pas → Rewind

## 9. Support images

### 9.1 Injection d'images

Claude Code peut analyser des images (screenshots, diagrammes, maquettes). Vous pouvez les injecter de plusieurs manieres :

```
# Drag & drop dans le terminal
# (glissez l'image directement dans la fenetre)

# Copier-coller (selon le terminal)
# Cmd+V / Ctrl+V avec une image dans le presse-papier
```

### 9.2 Cas d'usage

Usage	Exemple
<b>Debug UI</b>	“Voici un screenshot, le bouton est mal aligne”
<b>Implementation maquette</b>	“Implemente ce design” (+ image Figma)
<b>Analyse d'erreur</b>	“Voici le screenshot de l'erreur dans le navigateur”
<b>Documentation</b>	“Genere un diagramme Mermaid a partir de ce schema”

## 10. Sandboxing

### 10.1 Concept

Le sandbox fournit une **isolation active au niveau OS** pour l'execution de Claude Code. Sur macOS, il utilise Seatbelt (sandbox-exec) pour restreindre les operations. Sur Linux, il utilise des mecanismes de conteneurisation equivalents.

### 10.2 Activation

```
# Activer le sandbox
/sandbox

# Ou au lancement
claude --sandbox
```



### 10.3 Ce que le sandbox empeche

- **Ecriture de fichiers** en dehors du repertoire du projet (isolation OS-level)
- **Acces au reseau** (configurable)
- **Execution de commandes systeme** sensibles
- **Acces aux variables d'environnement** systeme

**Note :** Le sandbox est une isolation active au niveau du systeme d'exploitation, pas une simple prevention passive. Les tentatives de sortie du perimetre sont bloquees par l'OS.

## 11. Keybindings et status line

### 11.1 Raccourcis clavier principaux

Raccourci	Action
Enter	Envoyer le message
Shift+Enter	Nouvelle ligne dans le message
Ctrl+C	Annuler la generation
Esc	Annuler l'action en cours
Esc+Esc	Rewind au checkpoint precedent
Shift+Tab	Basculer en Plan Mode
Tab	Accepter une suggestion/completion
Up/Down	Naviguer dans l'historique des prompts

### 11.2 Status line

La status line en bas de l'interface affiche :

claude-opus-4-6	23% ctx	\$0.42	fast	plan
Modele actif	Context utilise	Cout session	Mode rapide	Mode plan

Surveillez le pourcentage de contexte : au-dela de 60%, envisagez /clear ou /compact.

## 12. Formats de sortie

### 12.1 text (default)

Sortie en texte brut, lisible par un humain :

```
claude -p "Quelle heure est-il ?" --output-format text
# Il est 14h30.
```

### 12.2 json

Sortie structuree en JSON, parseable programmatiquement :

```
claude -p "Liste les fichiers TS" --output-format json

{
  "type": "result",
  "subtype": "success",
  "cost_usd": 0.003,
  "is_error": false,
  "duration_ms": 1523,
  "duration_api_ms": 1200,
  "num_turns": 1,
  "result": "Voici les fichiers TypeScript..."
}
```

### 12.3 stream-json

Sortie streamee en JSON, une ligne par evenement :

```
claude -p "Genere du code" --output-format stream-json
```

Chaque ligne est un objet JSON independant, permettant le traitement en temps reel.

---

## Exercice pratique : Maitriser les patterns (30 min)

### Etape 1 : Prompt engineering (10 min)

Naviguez dans un projet existant et pratiquez differents styles de prompts :

```
# 1. Prompt vague (observez la reponse)
> Explique ce projet
```

```
# 2. Prompt precis (comparez)
> Explique l'architecture de ce projet en identifiant :
> - Les couches (presentation, business, data)
```

```
> - Les patterns utilises
> - Les dependances externes
> Presente le resultat sous forme de tableau

# 3. Prompt avec contrainte
> Identifie les 3 points d'amelioration les plus impactants
> pour la maintenabilite de ce projet. Pour chaque point,
> donne un exemple concret tire du code.
```

## Etape 2 : Plan Mode (10 min)

```
# Activez le Plan Mode
> Shift+Tab

# Demandez une modification complexe
> Ajoute un systeme de cache avec invalidation
> pour les endpoints les plus appeles de l'API

# Lisez le plan propose par Claude
# Modifiez-le si necessaire
> Ajoute aussi un endpoint pour vider le cache manuellement

# Validez le plan
> Le plan est bon, implemente-le
```

## Etape 3 : Mode headless (10 min)

```
# Prompt simple
claude -p "Combien de fichiers TypeScript dans ce projet ?" --output-format
  ↪ text

# Piping
claude -p "Genere un .gitignore pour un projet Node.js" > .gitignore

# JSON pour parsing
claude -p "Liste les dependances du projet" --output-format json | jq
  ↪ '.result'

# Enchainement
git diff HEAD~1 | claude -p "Resume les changements du dernier commit"
```

---

## Points Cles a Retenir

1. **Prompts = objectifs** : Decrivez le quoi et le pourquoi, laissez Claude gerer le comment
2. **Plan Mode** : Toujours planifier avant d'agir pour les taches complexes (> 3 fichiers)

3. **Context window** : Ressource critique, surveillez le %, utilisez `/clear` et `/compact`
  4. **Sub-agents** : Delegez les explorations pour garder votre contexte propre
  5. **Fast Mode** : 2.5x plus rapide pour les taches simples et repetitives
  6. **Headless** : `claude -p` pour l'automatisation et l'integration scripts
  7. **Sessions** : `--continue` pour reprendre, `--resume` pour une session specifique
  8. **Rewind** : Esc+Esc pour revenir en arriere quand Claude prend une mauvaise direction
  9. **Images** : Drag & drop pour le debug UI et l'implementation de maquettes
  10. **Iterez** : Ne cherchez pas le prompt parfait, affinez au fur et a mesure
- 

**Prochain module** : Module 4 - Pratique Guidee