Table des matières

Vocabulaire utilisé

- **prompt structuré** (p. 2)
- **Markdown** (p. 2)
- **skill** (p. 5)
- **agent** (p. 6)
- **commande** (p. 9)
- **slash command** (p. 9)

1. L'IA pour LaTeX

1.1 Site internet dédié

Vous pourrez trouver sur le site internet **https://bfcours.dev/** toutes les ressources nécessaires pour utiliser des agents CLI modernes pour le langage LaTeX.
Ce site centralise :

1. La documentation complète du package bfcours
2. Les tutoriels d'installation et de configuration des agents IA
3. Les exemples d'utilisation des serveurs MCP
4. Les templates de prompts prêts à l'emploi

1.2 Principes généraux de communication avec les agents IA

Remarque(s) :

Les agents IA modernes comme Claude Code se distinguent des chatbots classiques :

IA conversationnelle (ChatGPT, Claude web)

1. Discussion en langage naturel
2. Pas d'accès aux fichiers
3. Copier-coller manuel
4. Une seule compétence générale

IA agentique (Claude Code, Cursor)

1. Manipulation directe des fichiers
2. Compilation et validation automatiques
3. Délégation à des agents spécialisés
4. Architecture modulaire avec skills

Pour tirer parti d'une IA agentique, il convient d'utiliser des formats précis qui seront décrits ci-après.

1.3 Prompt structuré

Définition

Prompt structuré

Un **prompt structuré** est une instruction à un agent IA organisée selon une hiérarchie claire de sections et sous-sections.
Par convention, on utilise le langage **Markdown** qui permet de formater du texte avec une syntaxe légère.

Notation

Syntaxe Markdown de base

Voici les éléments essentiels de Markdown utilisés dans les prompts :

Syntaxe Markdown	
Syntaxe	**Utilisation**
`# Titre`	Titre principal (une seule fois par document)
`## Section`	Section de niveau 2 (équivalent à `\section`)
`### Sous-section`	Sous-section de niveau 3 et ainsi de suite
`**texte**`	Texte en gras pour accentuer
`- Élément`	Liste à puces non énumérée
`1. Premier`	Liste énumérée avec numéros
`` `code` ``	Code inline (commandes, noms de fichiers)

Exemple(s) — **Prompt mal structuré vs bien structuré**

Prompt mal structuré (langage naturel non organisé) :
« Peux-tu formater mon fichier cours.tex en utilisant les conventions bfcours ? Il faut remplacer enumerate par tcbenumerate et textbf par acc. Aussi il faut ajouter les codes compétences du programme de 3ème. Et compile le document après pour vérifier que ça marche. Si ça compile pas corrige les erreurs. »
Prompt bien structuré (Markdown hiérarchisé) : voir le listing ci-après.

Listing 1 : Prompt structuré efficace

```
# Formatage d'un cours selon bfcours

## Objectif
Formater le fichier cours.tex selon les conventions bfcours et valider la compilation.

## Étapes à suivre

### 1. Transformations syntaxiques
- Remplacer enumerate par tcbenumerate
- Remplacer \textbf par \acc
- Remplacer tabular par tcbtab

### 2. Ajout des compétences
- Utiliser competences-server pour le niveau 3ème
- Intégrer les codes dans chaque exercice

### 3. Validation
- Compiler le document avec lualatex
- Corriger les erreurs éventuelles
- Analyser le PDF généré
```

Dans les parties suivantes, on présentera des templates de prompts structurés pour différents types d'interactions avec les agents IA (serveurs MCP, skills, agents spécialisés, commandes).

1.4 Protocoles de communication

Définition — **Protocoles MCP et A2A**

Les agents IA utilisent deux protocoles principaux pour communiquer avec leur environnement :

1. **MCP (Model Context Protocol)** : protocole standardisé permettant aux modèles IA de communiquer avec des applications externes (bases de données, outils de calcul formel, compilateurs, etc.). Un serveur MCP expose des outils spécialisés que l'agent peut invoquer.

2. **A2A (Agent-to-Agent)** : protocole de communication interne permettant à un agent principal de déléguer des sous-tâches à des agents spécialisés. L'agent principal orchestre le travail en appelant des sous-agents experts dans leur domaine.

Exemple(s) — **Gestion de boites mail avec MCP**

Vous souhaitez connecter votre boite mail gmail avec votre agent principal et vous tenir informé.
Vous téléchargez le serveur MCP fourni par google pour gmail en suivant les instructions de la documentation.
Prompt à l'agent :
« Dans ma boite mail, y a-t-il du nouveau courrier lié à [domaine] ? avec le serveur mcp gmail. »
L'agent utilisera les différentes commandes fournies pour réaliser la tâche.
Il commencera bien souvent par la commande « help » du serveur mcp puis se laissera guider.
L'agent n'a pas besoin de connaître tous les détails : les serveurs MCP fournissent les services nécessaires pour le connecter à leur système.

Remarque(s) :

Un serveur MCP est chargé automatiquement dans le contexte de votre agent principal : veiller à ne pas trop en avoir car cela surcharge rapidement les agents.

La communauté IA développe de nombreux serveurs MCP dédiés : à vous d'explorer.

Pour des raisons évidentes de sécurité, il est recommandé de choisir le serveur MCP mis en place par l'entreprise spécifique que vous souhaitez connecter.

1.4.1 Template : Guide de serveur MCP

Chaque serveur MCP possède un guide d'utilisation structuré de cette manière :

Listing 2 : Structure type d'un guide de serveur MCP

```
# Guide du Serveur MCP [nom-serveur]

## Description
[Décrire en 1-2 phrases le rôle du serveur]
Exemple : Résout automatiquement les problèmes d'encodage UTF-8 des fichiers LaTeX.

## Objectif
[Expliquer le problème résolu]
Exemple : Quand Claude Code génère des accents français, Windows peut mal
interpréter l'encodage, produisant des caractères corrompus dans VSCode.

## Installation
bash
cd .claude/mcp_servers
setup_[nom]_mcp.bat


Le script installe automatiquement les dépendances nécessaires.

## Outils Disponibles

### 1. `nom_outil(param1, param2="valeur_defaut")`

[Description détaillée de ce que fait l'outil]

Paramètres :
- param1 (type, obligatoire) : Description du paramètre
- param2 (type, optionnel) : Description (défaut: valeur)

Retour : JSON avec structure détaillée

Exemples :
python
# Exemple 1 : cas typique
nom_outil("fichier.tex")

# Exemple 2 : avec paramètres optionnels
nom_outil("fichier.tex", param2="custom")


### 2. `autre_outil(param)`

[Description du second outil...]
```

1.5 Skill

Définition — **Skill**

Un **skill** est une collection organisée de ressources permettant de doter un agent d'une expertise spécialisée. Il regroupe :

1. Un fichier **SKILL.md** : prompt système décrivant les workflows, connaissances et meilleures pratiques
2. Un dossier **references/** : fichiers Markdown contenant les conventions et procédures détaillées
3. Un dossier **scripts/** : programmes Python, Bash ou autres pour automatiser des tâches
4. Un dossier **assets/** : templates, images, fichiers de configuration et exemples

Le fichier `SKILL.md` est le **cœur du skill** : il contient le prompt système que l'agent charge pour acquérir l'expertise. Les autres dossiers fournissent les ressources nécessaires.

Exemple(s) — **Skill de formatage LaTeX bfcours**

Le skill `bfcours-latex` (dans `.claude/skills/bfcours-latex/`) aide les agents à formater des documents selon les conventions bfcours.

Situation : Un collègue vous envoie 15 exercices de géométrie rédigés en LaTeX standard. Vous voulez les convertir aux conventions bfcours.

Prompt à l'agent :

« Charge le skill bfcours-latex et formate ces 15 exercices selon les conventions bfcours. Ajoute les codes compétences du programme de $\mathbf{4}^{\text{ème}}$. »

Structure du skill bfcours-latex :

Fichier principal	Connaissances
SKILL.md (prompt système)	1. references/bfcours-conventions.md 2. references/exemples-usecases.md

Le skill **centralise l'expertise** : modifier les conventions ne nécessite que d'éditer le fichier .md correspondant, sans toucher au code des agents.

Ces informations peuvent être consultées à la fois par des humains, l'agent principal ou des agents secondaires.

1.5.1 Template : Fichier SKILL.md

Le fichier SKILL.md d'un skill suit cette structure avec une bannière YAML :

Listing 3 : Structure type d'un fichier SKILL.md

```
---
name: [nom-skill]
description: Utiliser ce skill pour [décrire l'expertise - ex: éditer des
documents LaTeX éducatifs avec le package bfcours]. [Contexte d'utilisation].
Utiliser de manière proactive pour [quand l'utiliser].
---

# Expert [Domaine]

Système expert pour [décrire le domaine précisément].

## Objectif

[Décrire l'objectif principal du skill]
Exemple : Éditer du contenu LaTeX en respectant les conventions strictes
du package bfcours pour l'enseignement des mathématiques.

## Séparation des Responsabilités

Ce skill se concentre uniquement sur [domaine]. Pour les autres tâches :

- [Tâche A]  Utiliser le skill [autre-skill-A]
- [Tâche B]  Utiliser le skill [autre-skill-B]

## Workflows Principaux

### Workflow 1 : [Nom du workflow]

[Description du cas d'usage]

Étapes :

1. [Étape 1] : [Description détaillée]
2. [Étape 2] : [Description avec outils à utiliser]
3. [Étape 3] : [Description]

### Workflow 2 : [Nom du workflow]

[Description d'un cas d'usage alternatif]

Étapes :
[Liste des étapes...]

## Architecture

[Décrire la structure des projets manipulés si pertinent]

## Connaissances

### Références documentaires
- references/[fichier1].md : [Description du contenu]
- references/[fichier2].md : [Description du contenu]

### Scripts disponibles
- scripts/[script].py : [Ce que fait le script]

### Assets
- assets/[dossier]/ : [Description des ressources]

## Meilleures Pratiques

1. [Pratique 1] : [Description]
2. [Pratique 2] : [Description]
```

1.6 Agent

Définition — **Agent**

Un **agent** est une instance d'IA spécialisée configurée par un fichier Markdown dans `.claude/agents/`. Ce fichier définit :

1. Une **bannière YAML** : métadonnées (nom, description, outils autorisés, couleur UI)
2. Un **prompt système** : instructions définissant le rôle, les objectifs et le workflow
3. Les **connaissances spécifiques** : références aux fichiers de connaissances à charger

Un agent est un **expert autonome** dans un domaine précis. L'instance principale peut déléguer des tâches complexes à des sous-agents spécialisés via le protocole A2A.

Exemple(s) — **Agent de Fact checking**

Situation : Vous avez quelques notes prises pendant une conférence. Vous voulez les convertir aux conventions bfcours, trouver les références et proposer des éléments pour fact-checker les informations de la conférence. Vous souhaitez un document LaTeX contenant tous ces éléments mis au propre.
Solution : Puisque vous allez voir un nombre incalculable de conférences, vous fabriquez un agent chargé de procéder comme dans la situation décrite ci-dessus.
« Utilise l'agent de mise au propre des notes pour toutes ces conférences : [chemin/vers/le/dossier/de/notes]. »
Votre instance principale apellera ainsi autant d'agents que nécessaire pour effectuer les tâches **en parallèle**.

1.6.1 Template : Fichier agent

Un agent est défini par un fichier Markdown avec une bannière YAML obligatoire :

Listing 4 : Structure type d'un fichier agent

```
---
name: [nom-agent]
description: Utiliser pour [décrire l'expertise - ex: mise au propre et fact-checking de notes]. Expert en
    [domaine spécifique].
tools: [Liste des outils autorisés séparés par des virgules]
color: [Couleur pour l'UI - ex: Blue, Green, Red]
---

# Rôle

Tu es [description du rôle - ex: un expert en fact-checking et production de documents LaTeX].

## Objectif

Ton objectif est de [objectif principal - ex: rendre un document qui
respecte la demande et qui compile sans erreurs].

Tu es responsable de [responsabilités - ex: recherche, mise au propre, compilation].

## Contexte de travail

Tu travailles sur des projets qui respectent l'architecture suivante :

[Décrire la structure des projets - répertoires, fichiers importants]

## Workflow

Quand tu es appelé, tu procèdes de la manière suivante :

1. [Étape 1 - ex: Lire le document dans lequel tu dois travailler]
2. [Étape 2 - ex: Faire des recherches pour vérifier les informations]
3. [Étape 3 - ex: Produire le .tex avec le modèle "chemin/vers/modele"]
4. [Étape 4 - ex: Valider le résultat par compilation]
5. [Étape 5 - ex: Analyser le PDF généré et corriger les erreurs]

## Connaissances

Les connaissances d'expertise sont stockées dans :

### À lire systématiquement
- [Chemin vers fichier 1] : [Description du contenu]

### Fichiers d'exemples
- [Chemin vers dossier d'exemples] : [Description]

## Règles spécifiques

[Lister les contraintes et règles importantes]
- [Règle 1 - ex: Toujours compiler avec lualatex]
```

1.7 Commande

Définition — **Commande**

Une **commande** (ou **slash command**) est un **prompt utilisateur** préenregistré stocké dans `.claude/commands/[nom].md`. Elle permet :

1. **D'encapsuler une instruction complexe** : transformer une requête détaillée en une commande courte
2. **D'accepter des paramètres** : personnaliser l'exécution avec des valeurs variables
3. **De définir un workflow précis** : spécifier les étapes, les skills à utiliser et les règles à respecter

Les commandes utilisent le préfixe / et sont invoquées par l'utilisateur. Contrairement aux agents et skills qui ont une bannière YAML, les commandes sont de simples fichiers Markdown structurés.

Exemple(s) — **Commande de formatage de contenu**

La commande `/formatTexBetter` (dans `.claude/commands/formatTexBetter.md`) formate un document LaTeX existant de manière compacte et rigoureuse.
Situation : Vous avez un document dont vous souhaitez améliorer le formatage et le rendre plus compact.
Utilisation : Comme cela arrive souvent, vous fabriquez une commande dédiée.
`/formatTexBetter`
La commande automatise tout le processus d'optimisation. Elle spécifie **précisément** quels skills et agents utiliser et dans quel ordre.
Ce sont des instructions que vous donner à votre instance principale.

Remarque(s) :

Une commande bien conçue contient :

1. Une **description** : ce que fait la commande en 1-2 phrases
2. Un **cadre d'intervention** : contexte et responsabilités
3. Les **skills et agents utilisés** : quels skills charger et pourquoi, à quel moment.
4. Un **workflow détaillé** : étapes numérotées avec actions précises
5. Des **exemples contextualisés** : où trouver des références dans la configuration

Contrairement aux agents qui ont des outils limités, une commande peut orchestrer plusieurs skills et agents pour réaliser une tâche complexe.

1.7.1 Template : Fichier commande

Une commande est un simple fichier Markdown sans bannière YAML :

Listing 5 : Structure type d'un fichier commande

```
# /[nom-commande] - [Description courte]

## Description

[Expliquer en 2-3 phrases ce que fait la commande]
Exemple : Adapte automatiquement un document LaTeX existant en optimisant
l'espace et en adoptant une présentation la plus rigoureuse possible.

## Cadre d'intervention

[Décrire le contexte de travail et les responsabilités]
Exemple : Tu devras utiliser les skills appropriés pour créer/modifier
le document. Ton travail consiste à optimiser l'espace dédié au contenu
sur chaque page.

## Skills

[Lister les skills à utiliser et leur rôle]
- [skill-1] : [Pour quelle tâche - ex: implémentation LaTeX]
- [skill-2] : [Pour quelle tâche - ex: analyse du PDF]
- [skill-3] : [Pour quelle tâche - ex: compilation]

## Exemples contextualisés disponibles

[Indiquer où trouver des exemples dans la configuration]
Exemple : Dans le dossier .claude/skills/bfcours-latex/assets/usecase
se trouvent des projets LaTeX complets entrés en production.

## Workflow

[Décrire le processus étape par étape]

1. [Étape 1 - ex: Lire la source que l'utilisateur veut adapter]
2. [Étape 2 - ex: Utiliser le skill X pour analyser Y]
3. [Étape 3 - ex: Faire une todolist des optimisations possibles]
4. [Étape 4 - ex: Modifier le code avec le skill Z]
5. [Étape 5 - ex: Compiler et valider]

Ce procédé doit être itératif et tu dois pouvoir améliorer jusqu'à
obtenir un document impeccable.

Critical : [Notes importantes]
Exemple : À la fin de ton intervention, nettoie les fichiers intermédiaires
non nécessaires pour ne pas créer de bazar.

## Paramètres (optionnel)

Si la commande accepte des paramètres :

- [param1] (obligatoire) : [Description et valeurs possibles]
- [param2] (optionnel) : [Description - défaut: valeur]

## Exemples d'utilisation


/[nom-commande] param1=valeur1 param2=valeur2
```