Claude Code

- 30 Étapes pour maîtriser Claude

30 Étapes pour maîtriser Claude

> Afin de bien prendre en main Claude Code il est nécessaire de bien comprendre certain point et d'optimiser la production en l'utilisant comme il se doit.

Gemini_Generated_Image_5l9h605l9h605l9h.png

1 - Installer Claude Code avec l'installateur Natif

Cela permet de garder Claude à jours, car les mises à jours seront réalisé automatiquement à chaque démarrage de session.

```
curl -fsSL https.../install.sh | bash
```

Par exemple si l'installation à était faite via Homebrew, les mises à jours ne seront pas automatiques.

Pour vérifier la version :

```
# Afficher la version (en dehors d'une session)
claude --version


# Afficher la méthode d'installation (dans une session)
/doctor
```

2 - Customiser l'interface de session

Personnaliser le thème

```
# Pour choisir le theme
/theme

# Pour choisir la couleur de l'invite
/color
```

Customiser la ligne de status

```
# Par exemple pour afficher des indications personnalisés
/statusline show active model in blue exemple : [Sonnet 1.6] and context usage in percent in green exemple : Ctx: 16% and token usage for the session in violet Tokens: 16k and the session limit in percent in orange exemple : (28%)
```

image.png

3 - Adapter l'affichage

Par défaut

Utiliser `/tui default` pour des échanges rapides

- Idéal pour des allez-retour rapides
- Questions / réponses
- en mode "chat" classique

Le mode fullscreen

Utiliser `/tui fullscreen` pour un rendu fluide

- Zéro scintillement
- Barre prompt fixe en bas pendant que claude travaille
- Mémoire constante durant la session

Bonus pour se concentrer

- Activer / Désactiver le mode focus avec `/focus`
- Une fois activé, vous ne verrez plus que le dernier prompt, un résumé des tools utilisés et la réponse de claude

4 - Initialiser la première session d'un projet

La commande suivante `/init` permet de générer la feuille de route. Elle est à faire seulement la première fois de l'utilisation de claude dans un projet.

- Analyser la codebase du projet : architecture, stack, convention
- Génère un CLAUDE.md
- Le fichier sera relu à chaque session par Claude

> Le fichier CLAUDE.md à deux porté : `<projet>/CLAUDE.md` (porté du projet) et `~/.claude/CLAUDE.md` (porté global, tous projets)

5 - Nommer / Renommer une session

La commande `/rename` permet de renommer la session en cours et d'afficher le nouveau nom dans la barre d'invite.

Il n'est pas obligé de renommer la session, Claude le fait automatiquement en se basant sur le contenu du prompt.

Mais il est conseillé d'avoir un nom de session approprié pour parcourir les sessions et retrouver les anciennes (prochain tip).

6 - Récupérer une ancienne session

La commande `/resume` permet de ré ouvrir une session avec son historique et tout son contexte.

> Par défaut, les sessions sont conservées pendant 30 jours. Ajustable avec la propriété `cleanupPeriodeDays` dans les settings (~/.claude/settings.json).

7 - Prises en main

Claude Code à une commande `/powerup` qui permet à l'utilisateur de faire un parcours interactif afin de découvrir les fonctionnalités.

- 10 leçons couvrant les fonctionnalités clés
- Chaque leçon inclus une explication + une mini démo
- Exemple de sujet : Naviguer dans la code base / Le mode permission / le contexte / etc..

8 - Les caractères clés dans l'invite de Claude

! Exécuter une commande bash

```
# Lancer la commande git fetch sans sortir de la session de claude
!git fetch
```

- Executer la commande + affiche le résultat
- L'auto complétion est également disponible

@ Réferencer un fichier dans un prompt

```
@src/app/login/login.component.ts écrit les tests unitaires du composant
```

- Injecte directement le contenu du fichier dans le contexte
- Auto complétion du chemin
- Résultats plus précis et moins de token utilisé

9 - Démarrer une session dans le bon dossier

Il est fortement conseiller de commencer une session claude à la racine du projet. **Pourquoi ?**

- Cela permet de lui donner un aperçu global de l'architecture
- Meilleure compréhension de la structure (front, back, config, scripts, ...)
- Récupère les dépendances entres les apps (front et back)
- Accède aux fichiers de configurations

> Il est aussi possible de spécifiez des répertoires supplémentaires externes avec : `--add-dir ../libs ../shared`

10 - Combinez CLAUDE.md et mémoire automatique

Deux systèmes de mémoires complémentaires existent dans Claude. Les deux systèmes sont chargés au début de chaque conversation. Plus vos instructions sont spécifiques et concises, plus Claude les suit.

Bien gérer le CLAUDE.md

- Court et précis (500 lignes max)
- Indiquer ce qu'il ne faut pas faire
- Mettre à jours avec les décisions techniques

Mémoire automatique

- C'est Claude qui la gère
- Se construit automatiquement via les conversations
- Ne duplique pas le contenu de CLAUDE.md

> Dans les deux cas il est possible de consulter et modifier ce que Claude à mémorisé avec la commande : `/memory`

11 - Maîtriser la gestion du contexte

Lorsqu'on lance Claude Code, on démarre une session. Cette session possède un contexte qui va s’agrandir progressivement :

- Les promptes et les réponses de l'agent
- Les fichiers lus
- Les résultats des commandes exécutés

Ce contexte est limité par une **fenêtre de tokens** qui dépend du modèle utilisé. (Sonnet, Opus, Haiku)

Plus la session avance, plus le contexte se remplit. Si la limite est atteinte, Claude peut commencer à oublier des instructions données plus tôt et produire des réponses incohérentes et avoir des hallucinations.

La commande `/compact` permet de compresser l'historique et de libérer l'espace sans repartir de zéro.

> Il ne faut surtout pas dépasser les **70% de contexte** sinon la session est bonne pour être recommencé de zéro. Utiliser la commande `/context` pour afficher le contexte utilisé par l'agent (voir l'étape 13).

12 - Vider la session à chaque changement de tâche

Suite au conseil précédent il existe la commande `/clear` qui permet de complètement vider une session.

- Efface l'historique de la conversation et repart de zéro (équivalent à recréer une session)
- Évite les compactions coûteuses sur du contexte qui est devenu inutile avec la nouvelle tâche
- Analyse plus précise et non biaisé par un ancien contexte

13 - Consulter le contexte

La commande `/context` permet d'afficher visuellement l'utilisation du contexte actuel.

- Affichage sous forme de grille colorée
- Répartitions des différents éléments chargés dans le contexte (Instructions, fichiers, historique, skills, etc..)

Pourquoi c'est utile ?

- Cela permet de comprendre ce que Claude à en mémoire réellement
- Identifier des éléments manquants ou inutiles
- Évite la surcharge de contexte et perdre de la précision de réponse

14 - Structurer les prompts

Structure

1. Être explicite sur le résultat attendu
2. Fournir le contexte et le pourquoi
3. Découper en étapes (step-by-step)

Précision

1. Inclure si possibles des exemples du résultat voulu
2. Formuler en positif (quoi faire, et ne pas quoi éviter)
3. Définir un critère de validation : tests, outputs

Pour les longs prompts il est possible d'utiliser la commande `/voice` pour activer la dictée vocale (taper sur <space> pour commencer à parler.

15 - Commencer en mode plan

Il est conseillé d'itérer les premiers échanges en mode plan avant de laisser Claude développer.

Cela permet de **challenger son plan d'exécution** et de voir si la demande a bien était comprise.

Pour changer entre le mode plan et le mode édition il faut réaliser la combinaison de touche : `Shift + Tab`

1. Mode plan pour initier la session et la demande
2. Challenger sur ses décisions et itérer avec lui tant que ce n'est pas parfait
3. Passer en mode écriture une fois le plan bien définit

16 - Utiliser le bon agent en fonction du besoin

Le choix du modèle impacte directement le résultat et le nombre de consommation de token.

En mode interactif avec la commande `/model` ou avec un mot clé : `default|sonnet|opus|haiku`

- **Haiku** -> tâche simples ou question simple, rapide et économique
- **Sonnet** -> usage quotidien, bon équilibre
- **Sonnet** 1M -> gros fichiers / longues sessions
- **Opus** -> Problèmes complexes, raisonnement avancés, analyses poussées

17 - Ajuster l'effort du modèle

Il existe plusieurs niveaux d'effort pour contrôler la profondeur du raisonnement. Le changement se fait avec la commande : `/effort`

- **low** -> tâches simples, faible latence, moins de tokens
- **medium** -> optimisation des coûts
- **high** -> bon équilibre coût / raisonnement
- **xhigh** -> recommandé pour les tâches complexes
- **max** -> raisonnement maximal, surcoût +++ (seulement avec Opus)

Il existe aussi le mot clé `ultrathink` pour demander un raisonnement poussé dans le prompt sans changer l'effort.

18 - Les raccourcis utiles

Pour gagner du temps il existe des raccourcis clavier :

- `Escape` pendant une réflexion en cours : Interruption de Claude proprement sans quitter la session
- Si input non vide `Escape+Escape` : Efface le prompt en cours de frappe

- Si input vide `Escape+Escape` : Affiche la liste des messages précédents (équivalent à `/rewind`)

19 - Glisser des pièces jointes dans Claude Code

Un Screenshots vaut mille explications. Claude Code "voit" très bien les images.

- Évite des allez-retour inutiles, pas besoin de reformuler ou clarifier le besoin
- Claude peut analyser : UI, erreurs visuelles, logs, structures, schéma etc...

Comment joindre une image ?

- Soit en spécifiant le chemin de son emplacement ./path/to/image.png
- Soit en Drag&Drop direct dans l'invite de Claude

20 - Tenter une autre approche dans une session

Il existe la commande `/fork` qui permet de bifurquer et tester différentes implémentations du problème.

Ce que ça fait :

- Créer une copie de la session courante pour explorer une approche différente
- La session principale reste intacte pendant l'expérimentation de l'autre approche

Cas d'usage idéal

- Hésitation entre deux stratégies d'architecture
- Tester une approche risquée
- Comparer deux implémentations en parallèle

Il sera toujours possible de retourner sur la branche initiale avec `/resume`.

21 - Inspecter et maîtriser les sorties

Il est important de garder la main sur l'output de Claude. Voici 4 commandes permettant d’interagir avec une réponse :

- `/diff` -> Ouvre un viewer intéractif (utiliser les flèches pour naviguer) entre le git diff global et les diffs de chaque étape de la session

- `/copy` -> Copie la dernière réponse de Claude dans le presse-papier. Utiliser `/copy N` pour copier les N dernières réponses.
- `/export` -> Ouvre une boîte de dialogue pour copier dans le presse-papier ou enregistrer le résultat dans un fichier. Uitliser `/export mon_fichier.md` pour écrire directement dans un fichier.
- `/undo` -> Annule toutes les modifications de la dernière demande

22 - Qualité et simplification

Claude permet de détecter ou de simplifier du code, sans quitter le terminal ou relancer une autre session.

`/review`

- Génère un retour structuré dans le terminal
- Identifie les bugs et edge casses potentiels
- Signales les mangues de doc ou de clarté
- Détecte les risques de performances

`/simplify`

- Lance trois agents d'analyse en parallèle
- Applique directement les corrections suggérées
- Possibilité de cibler un aspect précis : `/simplify focus sur ce point ...`

23 - Avoir un retour sur nos habitudes

Après plusieurs mois d'utilisation il est possible de générer un rapport d'analyse sur les sessions effectuées avec Claude Code : `/insights`

- Décortique l'usage réel (style de dev, demandes)
- Pointe les pertes de temps et les mauvais usages
- Propose des optimisations concrètes
- Aide à automatiser encore plus les workflows des demandes

> Ce rapport est une mine d'or ! Il vous montre ce que vous pouvez améliorer pour l'utiliser encore mieux.

24 - Automatiser des actions avec les Hooks (`/hooks`)

Il est également possible d'automatiser les actions répétitives dans le workflow de Claude Code.

- Déclencher des scripts automatiquement à des moments précis du workflow de Claude
- Permet de gérer des actions déterministes (formattage, tests, docs, etc..)
- Supportent des tâche asynchrones
- Events :
  - PreToolUse (avant chaque action de Claude)
  - PostToolUse (après chaque action)
  - Notification (sur notification)
  - Stop (quand Claude a terminé)
  - FileChanged (sur changement d'un fichier)

Il existe aussi deux autres commandes complémentaires mais attentions aux différences :

`/loop` (en local)

- Tourne dans le terminal actuel
- Idéal pour surveiller un déploiement, une PR, relancer une vérifications
- S'arrête quand on ferme le terminal

`/schedule` (dans le cloud)

- Tourne sur l'infrastructure d'Anthropic
- Déclenché par un calendrier, une API, ou des événement Github
- Continue même l'ordinateur éteint

25 - Créer ses propres skills

Cela permet d'étendre les capacités de Claude et de le spécialiser sur certain points.

Comment créer un skill ?

1. Créer un dossier pour le skill `~/.claude.skills/mon-skill`
2. Créer le fichier principal : `~/.claude.skills/mon-skill/SKILL.md`
3. Donner un nom, une description et des instructions (voir exemple ci dessous)
4. Utiliser le skill, de deux façons :
   1. Donne moi un "utilise le nom du skill" pour faire ...
   2. /nom-du-skill Effectue une action ...

> Il existe beaucoup de skills déjà prêt à l'emploi rédigé par Anthropic et la communauté !

```
---
name: mon-skill
```

```
description: Il faut mettre une description du skill, à quoi il sert, pourquoi faire
---


Maintenant on peut mettre toutes les instructions que l'agent doit faire avec ce skill.
```

26 - Les MCP

Qu'est ce qu'est les MCP ? Cela permet à Claude de réaliser des actions sur des serveurs externes, ce sont des outils pour Claude. Souvent hébergé par les vendors accessible via une simple URL.

MCP incontournable :

- Github -> PR, Issue, Actions
- Jira - Confluence -> Tickets, docs
- Sentry -> Erreurs, prod, debug
- Datadog -> logs métriques, traces
- Context7 -> docs à jours en contexte

La commande `/mcp` permet de voir les serveurs actifs.

> Attention d'activer seulement les MCP utiles à la session ! Dans le cas contraire les MCP utiliseront du contexte inutilement.