T03 # Modes de Permission

Technique Beginner

Contrôle des accès outils, du plus sûr au plus permissif

Modes disponibles

Mode	Nom canonique	Activation	Usage recommandé
Default	`default`	`--permission-mode default`	Développement quotidien
Auto-accept edits	`acceptEdits`	`Shift+Tab`	Revues de code
Plan	`plan`	`Shift+Tab x2` ou `/plan`	Analyse sans modification
Don't Ask	`dontAsk`	`/permissions` ou règles `permissions.allow`	Workflows restrictifs, refus silencieux si non pré-approuvé
Auto (classifier IA)	`auto`	`permissions` → `defaultMode: "auto"`	Tâches longues, moins d'interruptions
Bypass total	`bypassPermissions`	`--dangerously-skip-permissions`	CI/CD headless, sandboxé
Moins de prompts	(aucun)	`/fewer-permission-prompts`	Skill intégré : ajoute une allowlist issue des appels en lecture seule fréquents de vos transcripts

Activation en CLI : `claude --permission-mode <mode>` accepte `default`, `plan`, `acceptEdits`, `dontAsk`, `auto`, `bypassPermissions`. **Activation persistante :** clé `permissions.defaultMode` dans `settings.json`. Depuis v2.1.283, les sessions interactives du terminal et de VS Code démarrent en `auto` sauf si `permissions.defaultMode` ou `--permission-mode` en décide autrement.

Note : `--dangerously-skip-permissions` ne prompte plus pour les écritures dans `.claude/skills/`, `.claude/agents/` et `.claude/commands/` (v2.1.121) ; le reste de `.claude/` (settings, hooks) et `.git/` restent protégés même en bypass total. Le mode `auto` s'appuie sur un classifieur (côté serveur par défaut ; `CLAUDE_CODE_AUTO_MODE_MODEL` choisit le modèle côté client) : friction en moins, pas une frontière de sécurité.

Whitelist d'Outils

```
# Autoriser seulement certains outils
claude --tools "Read,Grep,Glob" --allowedTools "Read,Grep,Glob"

# Bloquer des outils spécifiques
claude --disallowedTools "Bash,Write"

# Combinaisons utiles
claude --allowedTools "Read,Edit,Bash(git *)"
```

Configuration dans settings.json

```
{
  "permissions": {
    "allow": [
      "Bash(git log*)",
      "Bash(npm test*)",
      "Read",
      "Edit"
    ],
    "deny": [
      "Bash(rm*)",
      "Bash(sudo*)"
    ]
  }
}
```

Hiérarchie des Permissions

Les permissions se cumulent et s'héritent dans cet ordre :

1. `~/.claude/settings.json` : global user
2. `.claude/settings.json` : projet (partagé)
3. `.claude/settings.local.json` : projet (local, gitignored)
4. Flags CLI : session uniquement
5. Politique gérée : contraintes de l'organisation

Les règles de refus de toute portée applicable prennent priorité sur les autorisations.

Glob Patterns pour Bash

```
# Autoriser git seulement
"Bash(git *)"

# Autoriser npm test et build
"Bash(npm test*)", "Bash(npm run build*)"

# Autoriser lecture fichiers
"Bash(cat *)", "Bash(ls *)"
```

Bonnes pratiques

CI/CD : pré-approuver les outils nécessaires et limiter leur disponibilité avec `--tools`. Si le contournement est nécessaire, utiliser un environnement isolé avec identifiants et accès réseau restreints.

Projets sensibles : restreindre Bash avec des motifs précis dans `.claude/settings.json`. Committer les valeurs partagées ; imposer par une politique gérée les contraintes que les développeurs ne doivent pas remplacer.

Audit : inspecter les transcriptions locales sous `~/.claude/projects/`. Les logs de hooks nécessitent leur propre répertoire configuré ; les transcriptions ne sont pas une piste d'audit inviolable.