Eigenflow

GUIDE · CLAUDE CODE

Monter ton système de vidéos avec Claude Code

La file, le calendrier, le plan de contenu et le routeur

PAR
Abyane Nabeebaccus

INSTAGRAM
@abyane_nb

ÉDITION
novembre 2026

Tu as commenté SYSTÈME, voici le guide. Il décrit la file de production qui a fabriqué nos reels 85 à 93 entre le 7 et le 9 novembre 2026. Un bot Telegram reçoit un lien, un Mac passe toutes les dix minutes, Claude Code fait la vidéo, et tu valides avant toute publication.

Au programme : nos chiffres réels, l'architecture, un script de départ écrit pour ce guide, puis les trois cadeaux promis. Documentations relues le 9 novembre 2026 ; chiffres tirés du journal de notre file et d'un relevé de notre compte Instagram du même jour.

À savoir avant de lire

- La file ne publie rien seule. Chaque vidéo arrive sur Telegram avec trois boutons, et rien ne part avant que tu touches Valider. Sur nos 9 reels, 3 ont demandé une reprise.
- Il faut un abonnement Claude qui inclut Claude Code (Pro, Max, Ultra, Team ou Enterprise) ou un compte Console. Le plan gratuit ne l'inclut pas. Notre file tourne sur un abonnement Ultra.
- En mode `-p`, Claude Code passe par ton abonnement, sauf si la variable ANTHROPIC_API_KEY existe : la clé passe alors devant, et elle se paie à l'usage.
- Les chiffres de la vidéo viennent de reels faits avec Claude Code en session, avant que la file existe. Elle tourne depuis le 7 novembre, et ses premiers reels sortent le 9 novembre à 18 h. La fabrication montrée étape par étape est celle du reel 93, produit par la file.

---

1. Ce que fait le système

Le parcours d'une vidéo

1. Tu partages un lien au bot Telegram, avec une note si tu veux. Le bot le range dans une boîte.

2. Toutes les dix minutes, le Mac lance un script qui relit la boîte et les boutons touchés, puis choisit une seule tâche.

3. Pour une production, il lance Claude Code en mode non interactif (`claude -p`) avec une consigne écrite. Claude relève la vidéo de référence, vérifie les faits, écrit, fabrique voix, image, son et sous-titres, puis contrôle.

4. Claude écrit un fichier result.json. Le script le lit et t'envoie la vidéo avec Valider, À revoir et Rejeter.

5. Valider lance la publication au premier créneau libre sur Instagram, TikTok et YouTube Shorts. Neuf minutes après l'heure de sortie, un contrôle vérifie que tout est parti.

Nos chiffres, du 7 au 9 novembre 2026

La file a produit 9 reels (85 à 93), un post image et une fiche d'apprentissage. Première production lancée le samedi 7 novembre à 23 h 02, heure de Paris ; dernière vidéo prête le lundi 9 à 13 h 20, soit 38 h 19 min plus tard.

Temps de production, du lancement de la tâche au result.json, à l'horloge :

- le plus court : 54 min 50 s (reel 85) ;
- la médiane : 1 h 22 min 44 s (reel 91) ;
- le plus long : 5 h 56 min (reel 86), dont environ 4 h 50 de Mac éveillé. Ce reel contenait un vrai test : Claude cherchait des offres d'emploi et remplissait un formulaire, sans l'envoyer.

Les neuf productions cumulent 16 h 53 min ; une publication prend ensuite 8 à 13 minutes.

Trois reels sur neuf ont demandé une reprise. Sur le 88 et le 89, la file a refait l'accroche, en 17 puis 28 minutes (détail en partie 8). Sur le 90, le message disait surtout « c'est très fort » : la reprise l'a lu comme une validation et n'a rien touché. Les six autres ont été validés d'un bouton, sans un mot.

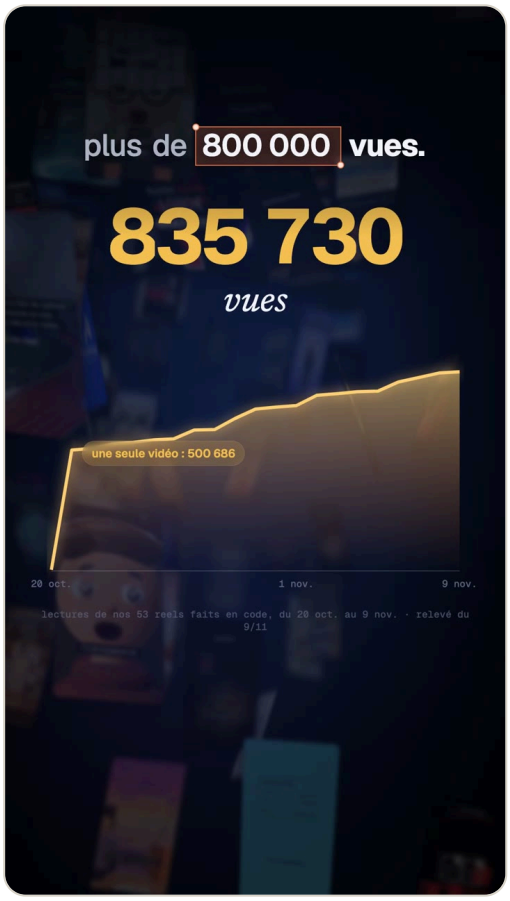
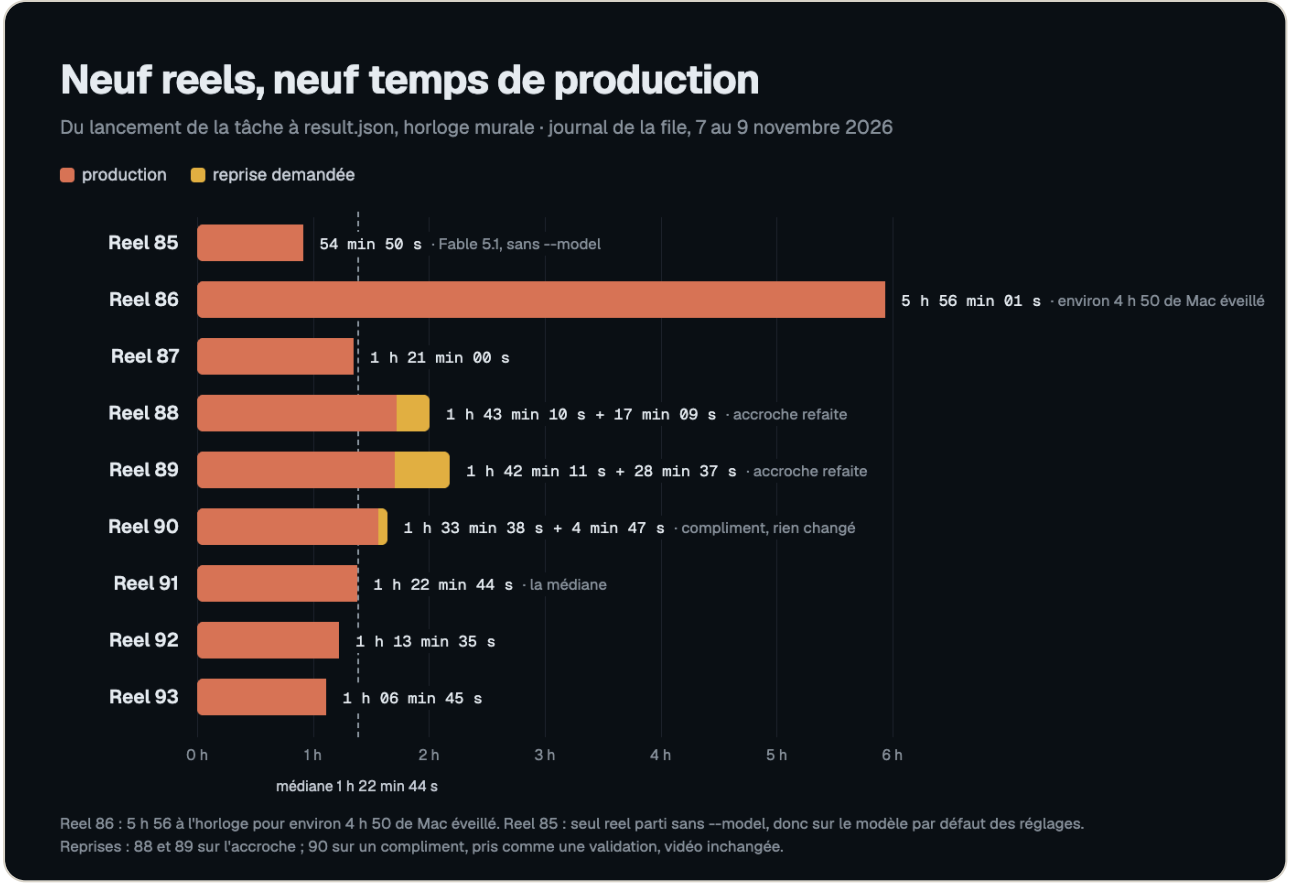


Image de la vidéo : 835 730 lectures en cumul pour les 53 reels faits en code publiés du 20 octobre au 9 novembre 2026, dont 500 686 pour un seul, sorti le 21 octobre (relevé Apify sans connexion du 9 novembre). Ces reels ont été faits en session, avant la file.



Le temps de production de chaque reel, d'après le journal de la file : de 54 min 50 s à 5 h 56 min, médiane 1 h 22 min 44 s, les trois reprises en jaune.

Ce qu'une production consomme

Avec `--output-format json`, Claude Code donne en fin de tâche un coût estimé au tarif de l'API (la doc parle d'une estimation locale, qui peut différer de la facture). Nos neuf productions : de 24,81 $ (reel 85) à 95,22 $ (reel 90, six styles rendus par six agents), médiane 37,00 $. Avec reprises et publications, 453,14 $, environ 50 $ par reel. On n'a rien payé en plus de l'abonnement.

D'où viennent les chiffres de la vidéo

Les 53 vidéos sont les reels faits en code publiés sur notre compte du 20 octobre au 9 novembre, relevés sans connexion le 9 novembre. Le seul reel tourné de la période, sorti le 1er novembre, n'est pas compté. Ces 53 reels ont cumulé 835 730 lectures, dont 500 686 pour un seul, sorti le 21 octobre. Les 40 styles sont les fiches de styles faits en code de notre catalogue interne, sans celui de cette vidéo.

Les douze reels qui jouent dans la vidéo vont du reel 29 (21 octobre) au reel 93. Neuf ont été faits avec Claude Code en session : on était dans la conversation et on donnait les liens à la main. Les trois autres (86, 91 et 93) viennent de la file, qui automatise cette marche depuis le 7 novembre ; ses reels sortent à partir du 9 novembre à 18 h, aucun ne compte encore dans ces lectures. Les pièces de la fabrication montrée (faits relevés par l'API GitHub, script, code de la ville, voix, mots horodatés, 1 h 06) sont celles du reel 93.

2. L'architecture

Les cinq pièces

- Le bot Telegram, sur un petit serveur toujours allumé. Il range chaque lien dans inbox.jsonl, chaque bouton et chaque note dans events.jsonl, et envoie ce que le Mac dépose dans un dossier outbox. Pour la file, il ne fait que relayer : quand tu écris une note, Marsouin répond « Noté, transmis au Mac », rien de plus.
- L'horloge : launchd sur Mac, toutes les 600 secondes ; une ligne cron sous Linux.
- Le script, qu'on appelle worker. Un passage, une tâche : il relit la boîte, choisit, lance Claude, lit le résultat, écrit le message.
- Les consignes, un fichier par type de tâche (production, reprise, publication, contrôle de sortie, apprentissage, bilan). Le script y remplace l'identifiant, le lien, ta note et tes remarques.
- result.json, le contrat entre Claude et le script. Pas de fichier, pas de message : le passage suivant réessaie, trois fois au plus, puis l'élément passe en erreur.

Ce que choisit un passage

Dans l'ordre : un contrôle de sortie dû, une vidéo validée à publier, une vidéo à reprendre, un lien « pour info » à apprendre, un mot de toi sur une vidéo déjà sortie, une production coupée. Un nouveau lien passe en dernier, seulement si aucune vidéo n'attend ta réponse, et dix minutes après son arrivée, le temps que ta note suive.

Les verrous et la reprise

Trois verrous : un pour les tâches longues, un pour les contrôles de sortie, un très court pour l'état. Un contrôle de sortie passe donc pendant une production, à condition d'avoir deux tâches launchd : d'après son manuel, launchd saute un lancement si la tâche tourne encore. Avec une seule tâche, le contrôle du reel 83, prévu à 18 h 09 le 8 novembre, a attendu 18 h 21.

Tout est sur le disque. L'état s'écrit dans un fichier temporaire puis se renomme : jamais de fichier à moitié écrit. Une production passe « en cours » avant le lancement ; si le Mac s'éteint, le passage suivant la relance, et la consigne dit de reprendre le dossier existant.

Le relais a eu sa panne : le 8 novembre, entre 0 h 05 et 0 h 20, il plantait à chaque message déjà marqué lu par Telegram. 17 liens perdus, puis récupérés. Depuis, chaque message est copié brut avant tout traitement, et la position de lecture avance après.

Pourquoi claude -p, sur l'abonnement

D'après la doc, `-p` lance Claude Code sans interface : il fait la tâche, puis s'arrête avec le code 0 si tout va bien. Côté facture, la page Authentication classe les identifiants, et la connexion de ton abonnement passe en dernier. Pour ANTHROPIC_API_KEY : « In non-interactive mode (-p), the key is always used when present. » Vérifie qu'aucune clé ne traîne dans l'environnement de launchd ou de cron.

- `--bare` démarre plus vite, la doc le recommande pour les scripts et l'annonce comme futur défaut de `-p`. Mais il ne lit ni ta connexion ni CLAUDE_CODE_OAUTH_TOKEN : il lui faut une clé API. Pour rester sur l'abonnement, pas de `--bare`.

- Sur un serveur sans navigateur, `claude setup-token` donne un jeton valable un an, à mettre dans CLAUDE_CODE_OAUTH_TOKEN. Il passe par ton abonnement.
- Donne toujours `--model` (un alias comme `opus` ou `sonnet`, ou un nom complet). Notre reel 85 est parti sans, sur le modèle par défaut de nos réglages. Depuis, la file passe `--model claude-opus-5-5`.

Les permissions, enfin. Depuis la version 2.1.302, une session `-p` sans réglage refuse tout ce qui demande une autorisation. Évite l'option qui saute toutes les permissions : la doc la réserve aux environnements isolés sans internet, et elle ne protège pas contre l'injection de consignes. Le script de ce guide passe `--permission-mode auto` (un classifieur relit chaque action) et `--permission-prompts none` (ce qui demanderait ton accord est refusé). Testé le 9 novembre : avec `sonnet`, tout passe ; avec `haiku`, l'écriture du result.json a été refusée, car le mode auto ne marche pas avec les anciens modèles, dont Haiku 4.5.

Le temps maximal par tâche

Une session qui ne finit jamais bloquerait toute la file. Le script coupe Claude après 6 h de production, 3 h de reprise, 2 h de publication, 45 min de contrôle de sortie. Les 6 h viennent des 4 h 50 du reel 86. La limite ne compte pas le temps où le Mac dort : le contrôle de sortie du reel 84 a duré 1 h 30 à l'horloge, pour 45 min de limite, sans être coupé. `caffeinate -i` empêche la veille d'inactivité pendant la tâche, pas celle du capot fermé.

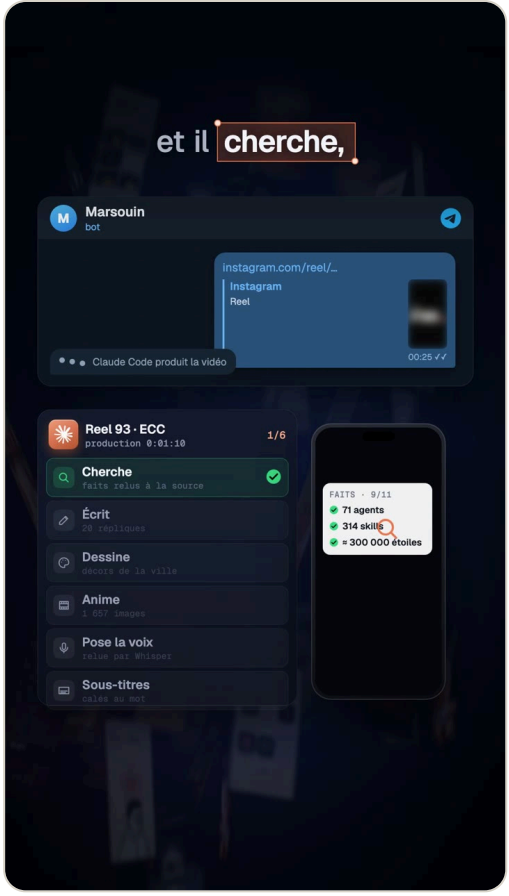
Pourquoi une version légère de la vidéo

La Bot API de Telegram : « Bots can currently send files of any type of up to 50 MB in size » (2 000 Mo avec un serveur Bot API local, à installer). À 8 Mbit/s, notre version de publication dépasse notre plafond de 48 Mo au-delà de 45 s environ. Six reels sur neuf ont eu besoin d'une version d'envoi : le 89, 169 s, pèse 158,2 Mo pour la publication et 44,2 Mo pour Telegram.

À COLLER

```
ffmpeg -i publication.mp4 -c:v libx264 -preset slow -b:v 3600k -maxrate 3600k -bufsize 7200k -c:a aac -b:a 128k -movflags +faststart envoi.mp4
```

Pour le débit vidéo en kbit/s, compte 360 000 divisé par la durée en secondes, moins 128 pour le son : 3 600k tient jusqu'à 90 s, 2 000k pour 169 s.



La conversation Telegram du reel 93, reconstituée d'après les vrais messages de la file : le lien envoyé le 8 novembre à 0 h 25, puis la première étape cochée. La bulle « Claude Code produit la vidéo » est une illustration, la file n'envoie rien au lancement.

Le journal de la file, le 9 novembre 2026

Extrait de worker.log, heures en UTC (Paris = UTC + 1 h) · lignes raccourcies, rien d'autre de changé

2026-11-09T11:14:09Z	lance production sur q1107-232544	12 h 14 · reel 93, la production démarre
2026-11-09T12:15:11Z	lance sortie sur outils	13 h 15 · contrôle de sortie du post de 13 h, pendant la production
2026-11-09T12:20:19Z	fin sortie sur outils : 0	
2026-11-09T12:20:54Z	fin production sur q1107-232544 : 0	13 h 20 · 1 h 06 min 45 s, la vidéo part sur Telegram
2026-11-09T12:40:23Z	événement valider sur q1107-232544 -> valide	Valider touché à 13 h 32, lu au passage de 13 h 40
2026-11-09T12:41:00Z	lance publication sur q1107-232544	
2026-11-09T12:50:38Z	fin publication sur q1107-232544 : 0	13 h 50 · programmé jeu. 12/11 à 18 h
2026-11-09T13:00:42Z	lance production sur q1107-232545	14 h 00 · la vidéo suivante démarre

Deux verrous : un contrôle de sortie (en bleu) passe pendant une production (en orange). Un bouton n'agit qu'au passage suivant de la file, toutes les 10 minutes.

Extrait du journal de la file le 9 novembre 2026, heures en UTC : la production du reel 93, un contrôle de sortie qui passe pendant ce temps, la validation lue au passage suivant, la programmation.

---

3. Le script de départ

Ce qu'il fait

Trois fichiers écrits pour ce guide ; notre code interne en fait plus, et le routeur (partie 6) dit quoi ajouter. file.py lit la boîte et les boutons, choisit une tâche, lance `claude -p` avec une limite de

temps, lit result.json, écrit l'état et le message. relais.py relève Telegram et envoie les vidéos ; il peut tourner chaque minute sur le même ordinateur, sans serveur, car Telegram garde 24 heures les messages non lus (doc de getUpdates). Les consignes vont dans un dossier consignes.

Testé le 9 novembre avec Claude Code 2.1.280, sur notre abonnement, avec `sonnet`, une consigne de trois lignes et `--safe-mode` en plus pour ne pas charger notre configuration : l'élément est passé « en cours », puis « prête », et le message est arrivé dans la boîte d'envoi. relais.py a été testé sans réseau, avec des messages Telegram simulés.

Installer

1. Installe Claude Code (`curl -fsSL https://claude.ai/install.sh | bash` sur Mac et Linux), lance `claude` une fois et connecte ton abonnement.

2. Crée un dossier file-videos avec file.py, relais.py et un sous-dossier consignes.

3. Dans Telegram, crée ton bot avec BotFather (commande /newbot) et garde son jeton.

4. Écris un message à ton bot, récupère ton identifiant de conversation, puis range les deux dans un fichier lisible par toi seul, jamais dans le code ni dans un dépôt git.

À COLLER

```
python3 -c "import
json,urllib.request;print(json.load(urllib.request.urlopen('https://api.telegram
.org/botTON_JETON/getUpdates'))['result'][-1]['message']['chat']['id'])"
mkdir -p ~/.config/file-videos
echo '{"token": "TON_JETON", "chat_id": 123456789}' > ~/.config/file-
videos/telegram.json
chmod 600 ~/.config/file-videos/telegram.json
```

file.py

À COLLER

```
#!/usr/bin/env python3
"""File de production minimale : un passage = une seule tâche. Lancée toutes les
10 minutes."""
import fcntl, json, subprocess, sys, time
from pathlib import Path

BASE = Path(__file__).resolve().parent
INBOX, EVENTS, STATE = BASE / "inbox.jsonl", BASE / "events.jsonl", BASE /
"state.json"
CLAUDE = "claude"     # chemin complet si launchd ou cron ne le trouvent pas
MODELE = "opus"       # toujours le donner, sinon c'est le modèle par défaut de
tes réglages
TIMEOUT = {"production": 4 * 3600, "reprise": 2 * 3600, "publication": 3600}
ATTENTE = ("pret", "revoir")  # une vidéo attend ta réponse : rien de neuf ne
démarre
REGLES = {("valider", "pret"): "valide", ("valider", "revoir"): "valide",
("revoir", "pret"): "revoir",
          "rejeter": "rejete", "remarque": "a_reprendre"}  # ce que fait chaque
bouton ou mot

def lire(p):
    return [json.loads(l) for l in p.read_text().splitlines() if l.strip()] if
p.exists() else []

def ecrire(p, data):  # fichier temporaire puis renommage : jamais un fichier
écrit à moitié
    p.with_suffix(".tmp").write_text(json.dumps(data, ensure_ascii=False,
indent=1))
    p.with_suffix(".tmp").replace(p)

def relever(st):
    connus = {i["id"] for i in st["items"]}
    st["items"] += [{"id": e["id"], "url": e["url"], "note": e.get("note", ""),
"status": "a_faire",
                     "essais": 0, "reprises": 0, "remarques": []} for e in
lire(INBOX) if e["id"] not in connus]
    evs = lire(EVENTS)
    for e in evs[st["ev_lus"]:]:
        it = next((i for i in st["items"] if i["id"] == e["item"]), None)
        if not it or it["status"] in ("programme", "rejete"):
            continue
        if e["type"] == "remarque":
            it["remarques"].append(e["text"])
        it["status"] = REGLES.get((e["type"], it["status"]),
REGLES.get(e["type"], it["status"]))
    st["ev_lus"] = len(evs)

def choisir(st):
    for statut, tache in (("valide", "publication"), ("a_reprendre", "reprise"),
("en_cours", "production")):
        for it in (i for i in st["items"] if i["status"] == statut):
            if tache == "reprise" and it["reprises"] >= 3:
                it["status"] = "ici"  # trois versions refusées : tu la finis
toi-même
                continue
            return tache, it
    if not any(i["status"] in ATTENTE for i in st["items"]):
        return next((("production", i) for i in st["items"] if i["status"] ==
```

```
"a_faire"), (None, None))
    return None, None

def lancer(tache, it):
    d = BASE / "items" / it["id"]
    d.mkdir(parents=True, exist_ok=True)
    res = d / "result.json"
    if res.exists():
        res.rename(d / f"result-{int(time.time())}.json")
    consigne = (BASE / "consignes" / f"{tache}.md").read_text()
    for k, v in {"ID": it["id"], "URL": it["url"], "NOTE": it["note"] or "
(aucune)", "RESULT": str(res),
                 "REMARQUES": "\n".join(it["remarques"]) or "(aucune)"}.items():
        consigne = consigne.replace("{{" + k + "}}", v)
    cmd = [CLAUDE, "-p", consigne, "--model", MODELE, "--output-format", "json",
           "--permission-mode", "auto", "--permission-prompts", "none"]
    if sys.platform == "darwin":
        cmd = ["caffeinate", "-i"] + cmd  # le Mac ne s'endort pas pendant la
tâche
    with open(d / f"{tache}-{int(time.time())}.log", "w") as log:
        try:
            subprocess.run(cmd, cwd=BASE, stdout=log, stderr=subprocess.STDOUT,
timeout=TIMEOUT[tache])
        except subprocess.TimeoutExpired:
            pass
    try:
        return json.loads(res.read_text())
    except (OSError, ValueError):
        return None

def message(it, texte, video=None, boutons=False):
    (BASE / "outbox").mkdir(exist_ok=True)
    ecrire(BASE / "outbox" / f"{time.time_ns()}.json",
           {"item": it["id"], "text": texte, "video": video, "boutons":
boutons})

def apres(tache, it, r):
    if not r:  # pas de result.json : le passage suivant réessaie, trois fois au
plus
        it["essais"] += 1
        if it["essais"] >= 3:
            it["status"] = "erreur"
            message(it, "Trois essais ratés, à regarder à la main.")
        return
    it["essais"], s = 0, r.get("status")
    if s == "pret":
        it["status"], it["reprises"] = "pret", it["reprises"] + (tache ==
"reprise")
        message(it, r.get("telegram", "Prête à valider."), r.get("envoi"),
boutons=True)
    else:
        it["status"] = s if s in ("programme", "deja_fait") else "erreur"
        message(it, r.get("telegram") or f"{s or 'erreur'} {r.get('dueAt',
'')}".strip())

def main():
    verrou = open(BASE / ".lock", "w")
    try:
        fcntl.flock(verrou, fcntl.LOCK_EX | fcntl.LOCK_NB)
    except BlockingIOError:
        return  # le passage précédent travaille encore
```

```
    st = json.loads(STATE.read_text()) if STATE.exists() else {"items": [],
"ev_lus": 0}
    relever(st)
    tache, it = choisir(st)
    if tache == "production":
        it["status"] = "en_cours"  # coupure en route : le passage suivant
reprend cet élément
    ecrire(STATE, st)
    if tache:
        apres(tache, it, lancer(tache, it))
        ecrire(STATE, st)

if __name__ == "__main__":
    main()
```

Un seul verrou ici, tenu tout le passage : plus simple, mais rien ne passe pendant une production. Les 4 h de limite sont un départ, à monter si tes vidéos contiennent de vrais tests.

relais.py

À COLLER

```
#!/usr/bin/env python3
"""Relais Telegram : range tes liens et tes boutons, envoie ce que la file
dépose. Aucune IA ici."""
import json, re, subprocess, time, urllib.request
from pathlib import Path

BASE = Path(__file__).resolve().parent
CONF = json.loads((Path.home() / ".config/file-
videos/telegram.json").read_text())  # chmod 600
API = f"https://api.telegram.org/bot{CONF['token']}/"
MOI = CONF["chat_id"]  # le bot ignore tous les autres comptes
BOUTONS = (("v", "Valider"), ("r", "À revoir"), ("x", "Rejeter"))
CODES = {"v": "valider", "r": "revoir", "x": "rejeter"}

def appel(methode, **params):
    req = urllib.request.Request(API + methode, json.dumps(params).encode(),
{"Content-Type": "application/json"})
    return json.load(urllib.request.urlopen(req, timeout=60))["result"]

def ajouter(nom, ligne):
    with open(BASE / nom, "a") as f:
        f.write(json.dumps(ligne, ensure_ascii=False) + "\n")

def traiter(u):
    if "callback_query" in u and u["callback_query"]["from"]["id"] == MOI:
        q = u["callback_query"]
        code, item = q["data"].split("|")
        ajouter("events.jsonl", {"type": CODES[code], "item": item, "ts":
time.time()})
        appel("answerCallbackQuery", callback_query_id=q["id"], text="Transmis à
la file")
    m = u.get("message") or {}
    if m.get("chat", {}).get("id") != MOI:
        return
    texte, lien = m.get("text", ""), re.search(r"https?://\S+", m.get("text",
""))
    vise = (m.get("reply_to_message") or {}).get("message_id")
    journal = BASE / "envoyes.jsonl"
    envois = {e["message_id"]: e["item"] for e in map(json.loads,
open(journal))} if journal.exists() else {}
    if vise in envois:  # une réponse à une vidéo envoyée : remarque sur cette
vidéo
        ajouter("events.jsonl", {"type": "remarque", "item": envois[vise],
"text": texte, "ts": time.time()})
    elif lien:
        ajouter("inbox.jsonl", {"id": f"q{u['update_id']}", "url": lien[0],
"note": texte.replace(lien[0], "").strip()})

def recevoir():
    f = BASE / "offset.txt"
    for u in appel("getUpdates", offset=int(f.read_text()) if f.exists() else 0,
timeout=0):
        ajouter("updates.jsonl", u)  # copie brute d'abord : rien ne se perd si
la suite plante
        f.write_text(str(u["update_id"] + 1))
        try:
            traiter(u)
        except Exception as e:
```

```
            ajouter("erreurs.jsonl", {"update": u["update_id"], "erreur":
repr(e)})

def envoyer():
    for f in sorted((BASE / "outbox").glob("*.json")):
        m = json.loads(f.read_text())
        clavier = {"inline_keyboard": [[{"text": t, "callback_data": f"{c}|
{m['item']}"} for c, t in BOUTONS]]}
        if m["video"]:  # moins de 50 Mo, sinon Telegram refuse
            cmd = ["curl", "-sf", API + "sendVideo", "-F", f"chat_id={MOI}", "-
F", f"video=@{m['video']}",
                   "-F", "supports_streaming=true", "--form-string", f"caption=
{m['text']}"]
            cmd += ["--form-string", f"reply_markup={json.dumps(clavier)}"] if
m["boutons"] else []
            res = json.loads(subprocess.run(cmd, capture_output=True, text=True,
check=True).stdout)["result"]
        else:
            res = appel("sendMessage", chat_id=MOI, text=m["text"], **
({"reply_markup": clavier} if m["boutons"] else {}))
        ajouter("envoyes.jsonl", {"message_id": res["message_id"], "item":
m["item"]})
        f.unlink()

if __name__ == "__main__":
    recevoir()
    envoyer()
```

Pour une remarque, réponds au message de la vidéo (appui long, Répondre). Garde le filtre sur ton identifiant : n'importe qui peut écrire à un bot.

La consigne de production

À enregistrer dans consignes/production.md. Le script remplace ce qui est entre doubles accolades.

À COLLER

```
Tu travailles seul, lancé par une file de production. Personne ne te répond.
Écris en français.

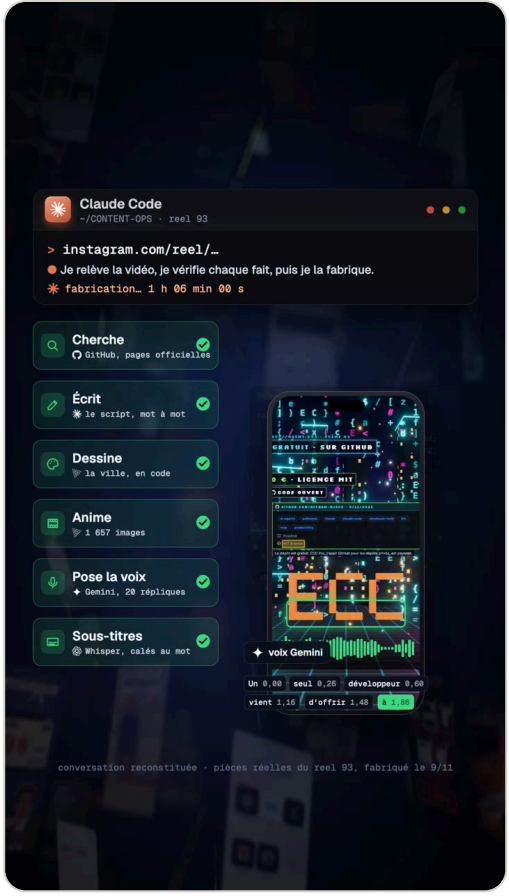
Élément : {{ID}}
Lien reçu : {{URL}}
Ma note : {{NOTE}}

Si un dossier existe déjà pour {{ID}} (production coupée), reprends où il en
est.

1. Relève. Ouvre le lien sans te connecter à aucun compte. Note les lectures,
les j'aime, la date et la médiane des 24 dernières vidéos du compte. Garde la
transcription.
2. Vérifie. Cherche si le sujet est déjà fait dans productions/ et dans plan.md
; si oui, arrête-toi avec le statut deja_fait. Relis chaque fait, chaque chiffre
et chaque prix sur la page officielle du jour, avec la date. Prix en euros pour
la France, HT ou TTC.
3. Écris. Un script court, réplique par réplique. Chaque affirmation a sa preuve
à l'écran : capture réelle, test fait par toi, chiffre daté. Ce que tu ne peux
pas prouver sort du script.
4. Produis. Dossier productions/<numéro>-<sujet>/ : voix, image, son, sous-
titres, avec les outils listés dans OUTILS.md. Tu ne lances aucun code tiers, tu
ne crées aucun compte, tu ne publies rien.
5. Contrôle. Fais relire la voix par une transcription automatique. Regarde la
vidéo en planches à 2 images par seconde, à côté de ma meilleure vidéo. Ce qui
est plus vide ou plus lent se reprend.
6. Rends. Fabrique une version de moins de 48 Mo pour Telegram, puis écris
{{RESULT}} :
{"status": "pret" | "deja_fait" | "erreur", "titre": "...", "dossier": "...",
 "envoi": "chemin de la version légère",
 "telegram": "une phrase de quinze mots au plus",
 "resume": "quatre lignes : le sujet et ses chiffres, ce que tu as vérifié, les
choix prudents, ce qui reste à voir"}

Si une décision me revient, prends l'option prudente et dis-le dans le résumé.
```

La reprise suit le même moule : elle reçoit {{REMARQUES}}, applique le changement, refait les contrôles et rend « pret ». La publication est la seule consigne qui touche à Internet : écris en tête que la validation vaut pour cet élément seulement, puis quoi publier, où, à quel créneau (partie 4). Elle rend « programme » avec l'heure dans dueAt.



Les six étapes de la consigne, cochées pour le reel 93 au bout de 1 h 06 de production, chacune avec son outil : cherche (API GitHub et pages officielles), écrit (le script, mot à mot), dessine (la ville en code, three.js), anime (1 657 images), pose la voix (Gemini, 20 répliques), sous-titres (Whisper, calés au mot). Derrière, les vraies pièces. Image tirée de la vidéo.

Lancer toutes les dix minutes

Sur Mac, un fichier dans ~/Library/LaunchAgents, avec ton nom d'utilisateur à la place de TOI. launchd ne lit pas ton profil de terminal : le PATH doit contenir le dossier de claude.

À COLLER

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0"><dict>
 <key>Label</key><string>com.toi.file-videos</string>
 <key>ProgramArguments</key><array>
  <string>/usr/bin/python3</string>
  <string>/Users/TOI/file-videos/file.py</string>
 </array>
 <key>WorkingDirectory</key><string>/Users/TOI/file-videos</string>
 <key>StartInterval</key><integer>600</integer>
 <key>RunAtLoad</key><true/>
 <key>StandardOutPath</key><string>/Users/TOI/file-videos/launchd.log</string>
 <key>StandardErrorPath</key><string>/Users/TOI/file-videos/launchd.log</string>
 <key>EnvironmentVariables</key><dict>
  <key>PATH</key>
<string>/Users/TOI/.local/bin:/opt/homebrew/bin:/usr/local/bin:/usr/bin:/bin</st
ring>
 </dict>
</dict></plist>
```

Copie-le pour le relais (Label com.toi.file-videos-relais, relais.py, StartInterval 60), puis charge les deux :

À COLLER

```
launchctl bootstrap gui/$(id -u) ~/Library/LaunchAgents/com.toi.file-videos.plist
launchctl bootstrap gui/$(id -u) ~/Library/LaunchAgents/com.toi.file-videos-relais.plist
```

Sous Linux, deux lignes dans `crontab -e`. Le fichier claude.env, en chmod 600, contient la ligne export CLAUDE_CODE_OAUTH_TOKEN=... avec le jeton de `claude setup-token`.

À COLLER

```
*/10 * * * * cd $HOME/file-videos && . $HOME/.config/file-videos/claude.env && python3 file.py >> file.log 2>&1
* * * * * cd $HOME/file-videos && python3 relais.py >> relais.log 2>&1
```

---

4. Le calendrier

Nos créneaux

Trois publications par jour depuis le 22 octobre, à 8 h, 13 h et 18 h, heure de Paris, sur Instagram, TikTok et YouTube Shorts à la fois (un post image : Instagram seul). On a tenté cinq créneaux le 7 novembre, abandonnés le jour même. Le 9 novembre à 14 h, tout était pris jusqu'au 12 : 10 reels et 2 posts image. Premier créneau libre : vendredi 13 à 8 h.

Le premier créneau libre

Le premier des trois créneaux qui tombe au moins 20 minutes après le moment où tu programmes, et où rien n'est prévu sur aucun des trois réseaux. Jamais deux publications sur un même créneau. Lis d'abord ce qui est déjà programmé dans ton outil (chez nous, Buffer). Le reel 93, validé le 9 novembre à 13 h 32, a été programmé le 12 à 18 h : tout était pris avant.

À COLLER

```
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo

PARIS = ZoneInfo("Europe/Paris")
CRENEAUX = (8, 13, 18)

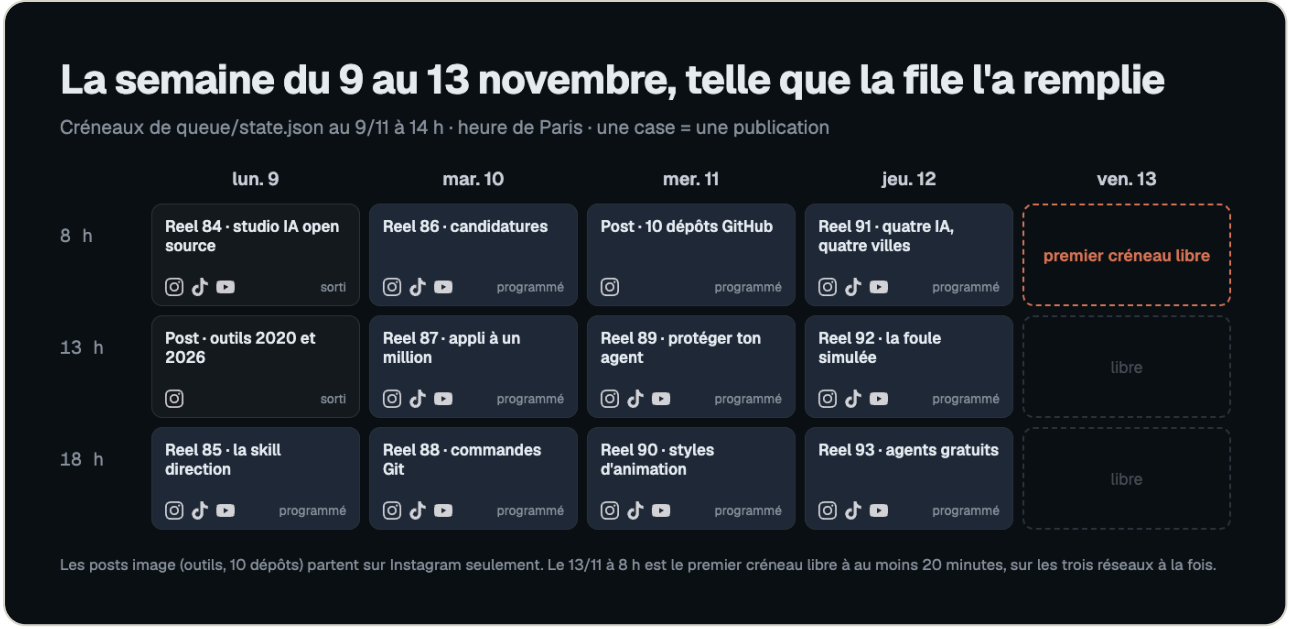
def premier_creneau(occupes, maintenant, marge=timedelta(minutes=20)):
    """occupes : les heures déjà programmées (datetime avec fuseau), lues dans
ton outil."""
    t = maintenant + marge
    jour = t.date()
    while True:
        for h in CRENEAUX:
            c = datetime(jour.year, jour.month, jour.day, h, tzinfo=PARIS)
            if c >= t and c not in occupes:
                return c
        jour += timedelta(days=1)
```

Stocke les heures en UTC et calcule les créneaux avec le fuseau Europe/Paris, comme ci-dessus : une heure UTC fixe glisserait d'une heure au changement d'heure.

Le plan gratuit de Buffer garde 10 publications programmées par réseau, sur 3 réseaux (page de prix, 9 novembre 2026) : un peu plus de trois jours d'avance à ce rythme. Le 9 novembre, notre file Instagram était à 10 sur 10.

À COLLER

```
Semaine du ___     8 h      13 h     18 h
lun.               ____     ____     ____
mar.               ____     ____     ____
mer.               ____     ____     ____
jeu.               ____     ____     ____
ven.               ____     ____     ____
sam.               ____     ____     ____
dim.               ____     ____     ____
Réseaux : Instagram, TikTok, YouTube Shorts (post image : Instagram seul)
Case : numéro, sujet en trois mots, mot à commenter
```



La semaine du 9 au 13 novembre 2026 telle que la file l'a remplie, d'après son fichier d'état : 12 créneaux pris, le premier libre le vendredi 13 à 8 h.

---

5. Le plan de contenu

La data avant les idées

Un sujet entre dans le plan quand une vidéo de référence prouve qu'il marche. La mesure : ses lectures divisées par la médiane des 24 dernières vidéos de son compte. 500 000 lectures sur un compte qui en fait 400 000 d'habitude ne prouvent pas grand-chose ; 50 000 sur un compte à 2 000, si. Les sujets de notre semaine s'appuyaient sur des vidéos de référence entre 3,3 et 333 fois la médiane de leur compte.

Relève ces chiffres sans te connecter : on passe par Apify, dont l'extracteur Instagram lit les pages publiques sans compte. On a cessé d'utiliser notre session Instagram le 4 novembre, après une erreur 429 (trop de requêtes).

La banque de sujets

Une fiche par sujet, dans un seul fichier. Avant de produire, deux vérifications :

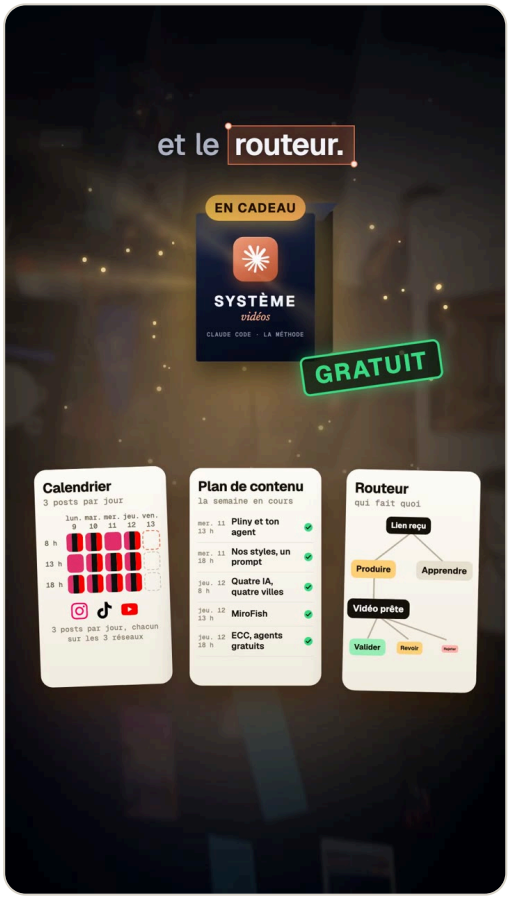
- Déjà fait ? Cherche le lien et le sujet dans tes productions et dans la banque. Une ligne « prête » ne prouve rien : chez nous, deux sujets notés prêts avaient déjà donné les reels 31 et 32.
- Toujours vrai ? Chaque fait et chaque prix se relit sur la page officielle, le jour même. Une vidéo de référence à plus de 550 000 lectures annonçait une offre fermée depuis cinq jours. Pour notre reel 83, l'offre gratuite d'un fournisseur cité avait disparu. Les prix s'écrivent en euros pour la France, HT ou TTC.

À COLLER

```
Sujet : _____ (une phrase)
Vidéo de référence : lien _____ · lectures _____ · médiane du compte _____ ·
ratio ___ × · relevé le __/__
Déjà fait chez moi ? non / oui, production n° ___
Faits à prouver : _____ → page officielle _____, lue le __/__
Preuves à l'écran : capture _____ · test _____ · chiffre daté _____
Format : _____ · durée visée ___ s · mot à commenter _____
Créneau : _____ · statut : à faire / en cours / prête / programmée / sortie
```

Le droit d'auteur

Une vidéo de référence dit qu'un sujet ou un format marche. Elle ne te donne pas ton contenu. En France, l'article L122-4 du Code de la propriété intellectuelle rend illicite toute reproduction, même partielle, faite sans l'accord de l'auteur, et ajoute : « Il en est de même pour la traduction, l'adaptation ou la transformation ». Le droit d'auteur protège la forme d'une œuvre ; l'idée reste libre. Ne reprends jamais le contenu de quelqu'un : refais tout avec tes propres preuves et ta propre voix.



Les trois cartes données avec la vidéo : le calendrier (8 h, 13 h, 18 h sur trois réseaux), le plan de contenu (la semaine en cours de la file) et le routeur. Image tirée de la vidéo.

---

6. Le routeur

La table qui dit à la file quoi faire de chaque élément, telle qu'elle tourne chez nous.

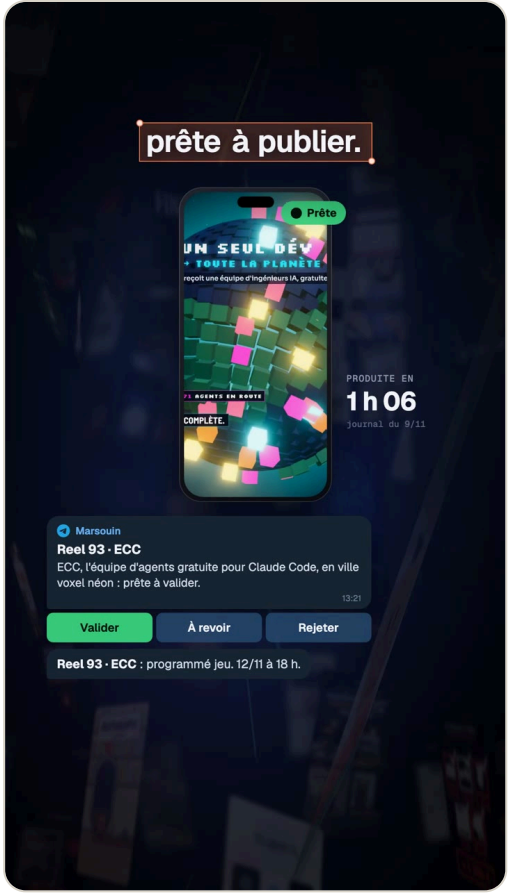
À COLLER

```
| Statut | Ce qui l'amène | Ce que fait la file |
|---|---|---|
| à faire | un lien reçu | attend 10 min ta note, puis produit si aucune vidéo
ne t'attend |
| apprise | ta note dit « pour info », « référence », « retiens » | décortique
et range ce qu'il faut retenir, rien à valider |
| en cours | la production a démarré | la reprend au passage suivant après une
coupure |
| question | un choix sans option prudente | t'envoie la question avec des
boutons |
| prête | result.json dit pret | t'envoie la vidéo avec Valider, À revoir,
Rejeter |
| à reprendre | ta remarque | applique le changement, renvoie une version |
| validée | Valider, ou un accord sans changement | publie |
| programmée | un créneau trouvé | contrôle la sortie 9 min après l'heure |
| sortie | tout est en ligne | rien, sauf retenir un mot de toi |
| rejetée | Rejeter, ou « laisse tomber » seul | rien |
| déjà faite | le sujet existe chez toi | te le dit en une ligne |
| dans la session | « reviens ici », ou trois versions refusées | t'attend dans
Claude Code, passe à la suivante |
| erreur | trois essais sans result.json | te prévient, attend ta réponse |
```

Ce que valent tes réponses :

- Valider, ou un mot court comme « ok », « validé », « gg » : la vidéo est validée. Seuls ces mots, ou un accord lu par une reprise, lancent une publication.
- À revoir : la file attend ta remarque, en réponse au message.
- Rejeter : abandonnée, sauf si elle est déjà programmée ou sortie.
- Une remarque : « le titre est trop petit » est un changement, « ce style est incroyable » un accord. Un compliment suivi d'un changement reste un changement.
- « Laisse tomber », seul : rejetée. « Laisse tomber la musique » reste une remarque.
- « Reviens ici » : la vidéo revient dans ta session Claude Code, la file continue.
- Une note pendant une production s'applique avant l'envoi, et ne vaut jamais validation : tu n'as pas vu la vidéo.
- Trois versions refusées : la quatrième se fait avec toi.

Les messages restent courts : trois lignes au plus pour une vidéo prête, le compte rendu reste dans result.json.



Le message de la vidéo prête et ses trois boutons, puis « programmé jeu. 12/11 à 18 h » après Valider. Conversation reconstituée d'après les vrais messages de la file.

---

7. Les garde-fous

Rien ne part sans toi

La publication ne se lance que sur une vidéo validée. Une production ne se valide jamais seule, ni sur une note arrivée avant que tu aies vu la vidéo. Les autres consignes interdisent à Claude de publier, déployer, écrire à quelqu'un, créer un compte ou faire un commit.

Aucun code tiers pendant une production

Quand une vidéo parle d'un outil ou d'un dépôt GitHub, la file ne le lance pas : elle montre le dépôt, le signale à l'écran et dans la légende, et refait le résultat par ses moyens quand c'est possible. Un script inconnu tourne avec tes droits. Le jeton du bot, lui, reste dans son fichier, jamais dans une consigne : tout ce que lit Claude peut finir dans un journal.

La mention IA

Nos reels sont animés en code, avec une voix de synthèse. On coche la mention IA sur les trois réseaux, à chaque fois. Les règles, relues le 9 novembre 2026 :

- Instagram : Meta exige sa déclaration pour toute vidéo photoréaliste ou tout son réaliste « digitally created or altered », et « may apply penalties » sinon (article du 6 février 2024, mis à jour le 1er avril 2025). Le label s'appelle « AI info ».
- TikTok : étiquette obligatoire pour le contenu généré par IA qui montre des scènes ou des personnes d'apparence réelle (réglage « AI-generated content » à la publication).

- YouTube : déclaration obligatoire pour un contenu réaliste modifié ou synthétique, pas pour un contenu clairement irréaliste ni pour une aide à la production (script, sous-titres). Qui omet régulièrement de déclarer s'expose à un label d'office, au retrait de contenus ou à une suspension du Programme Partenaire.

Piège vu chez nous : Buffer remet la case IA de YouTube à « non » après la sortie. On la revérifie dans YouTube Studio.

Ce que ça coûte vraiment

Prix relus le 9 novembre 2026, convertis au cours BCE du 6 novembre (1 € = 1,1165 $).

- Claude : Pro à 20 $ par mois (17 $ à l'année), Max à 100 $, Ultra à 200 $, « Les prix affichés n'incluent pas les taxes applicables ». Soit 17,91 €, 89,57 € et 179,13 € HT, ou 21,50 €, 107,48 € et 214,96 € TTC avec la TVA de 20 %. Le 7 novembre, la page nous avait affiché 18 €, 90 € et 180 € HT ; le montant exact se lit à la page de paiement.
- Apify : 2,30 $ les 1 000 résultats en plan Starter, 2,70 $ en plan gratuit (5 $ de crédit par mois). Un relevé : 0,002 $ pour la vidéo, 0,03 à 0,055 $ pour les 24 dernières du compte. Nos 9 reels : 0,36 $.
- Voix : Gemini 3.8 Flash TTS, sans offre gratuite, 9 $ le million de tokens audio jusqu'au 31 décembre 2026, 18 $ ensuite. Notre outil note 0,002 $ par réplique : 0,41 $ pour 205 répliques.
- Buffer et Telegram : gratuits à ce volume.

Hors abonnement Claude, neuf vidéos coûtent moins d'un euro.

---

8. Les limites

Elle rate encore des accroches

C'est le point qui nous gêne le plus : les deux vraies corrections de la semaine portaient sur l'accroche. Au 88, elle montrait un logo et des mots qui volent, là où il fallait l'aide-mémoire complet dès la première image. Au 89, elle restait figée 1,6 s et l'écran se vidait au moment de frapper. La file a corrigé les deux seule. Depuis, la consigne interdit tout plan immobile plus d'une demi-seconde dans les quatre premières secondes, et demande un impact qui remplit l'écran.

Il faut un ordinateur allumé

Mac fermé, rien ne tourne. Telegram garde les messages 24 heures, et une production coupée reprend au réveil, avec une partie du travail à refaire.

Le temps

1 h 22 min par vidéo en médiane, jusqu'à 6 h avec un vrai test. Et la file attend ta réponse avant de lancer la suivante : le reel 92, prêt le 8 novembre à 22 h 33, a été validé le 9 à 11 h 25. 12 h 52 sans production.

Ce qui casse encore

- Sous charge, des serveurs MCP ont dépassé leur délai au démarrage de `claude -p` (8 novembre) : on a écrit des solutions de secours en script.
- La voix a son quota : 100 requêtes par jour sur notre palier Gemini. Le reel 92 est passé sur la version Flash-Lite, avec la même voix.
- `--bare` doit devenir le défaut de `-p`, et il ne lit pas l'abonnement. Surveille les notes de version.
- 453 $ de calcul estimé pour 9 reels : sur un abonnement Pro, la limite d'usage arrivera vite. On ne l'a pas mesuré, c'est à tester.

Sources

Toutes lues le 9 novembre 2026.

- Claude Code : headless, cli-reference, authentication, setup, permission-modes.
- Telegram : Bot API. macOS 26.6.2 : manuels de launchd.plist, launchctl, caffeinate.
- Mention IA : Meta, TikTok, YouTube.
- Prix : Claude, Apify Instagram, Apify, Gemini, Buffer, cours BCE du 6 novembre 2026.
- Article L122-4 du Code de la propriété intellectuelle.
- Nos fichiers : journal et état de la file, sorties JSON des sessions `claude -p`, relevé Apify de notre compte.