Fiche de commandes

Autorisé pendant les observations individuelles

Adrien Técher · édition du 29 septembre 2026

Observer avant de modifier

Vu en séance 1.

```
git status
git log --graph --oneline --all --decorate
git diff
git diff --cached
git show HEAD
git show origin/communication:annonce.txt
git branch -av
```

`diff` compare le répertoire de travail à l'index, `diff --cached` l'index au commit courant. `show référence:fichier` lit un fichier dans une référence.

Enregistrer un changement cohérent

Vu en séance 1.

```
git switch -c assistance
git add contact.txt
git diff --cached
git commit -m "docs: clarify support desk"
```

Le commit enregistre l'index, pas ce qui a été modifié après `git add`.

Intégrer et comparer

Vu en séance 2.

```
git fetch origin
git log main..origin/main --oneline
git switch -c integration
git merge avortees
git merge flottes
python -m pytest -q        # version Python
node --test                # version TypeScript
```

```
git diff depart integration > revue.diff
```

Une fusion sans conflit peut violer une règle métier : relire le contrat, tester l'interaction. `revue.diff` s'échange entre binômes sans ouvrir les espaces privés.

Préserver et annuler

Vu en séance 2, repris en séance 4.

```
git branch sauvegarde HEAD
git revert IDENTIFIANT
git reflog
git branch retrouvee IDENTIFIANT
git switch --detach IDENTIFIANT
git restore --staged fichier.txt
git reset --hard IDENTIFIANT
```

Une branche préserve l'accès à un commit. `revert` ajoute un commit d'annulation et convient à une branche partagée. `reflog` liste les positions récentes des références ; `branch retrouvee` redonne un nom à un commit qui n'en a plus. `switch --detach` place HEAD sur un commit sans branche : en créer une avant de repartir. `restore --staged` retire un fichier de l'index sans toucher au répertoire de travail. `reset --hard` déplace la branche et remplace le répertoire de travail : les modifications non enregistrées sont perdues, et une branche partagée ne se déplace pas ainsi.

Exécuter les tests et construire

Vu en séance 3. Chaque binôme travaille dans une seule version.

```
# Version Python (dépôts s1 à s4)
python -m pip install -r requirements.txt
python -m pytest -q
python -m pytest -q --junitxml=reports/junit.xml
python build_artifact.py
# Version TypeScript, Node.js 24 (dépôts ts-s1 à ts-s4)
node --check charging.ts
node --test
node --test --test-reporter=junit --test-reporter-destination=reports/junit.xml
node build_artifact.ts
git rev-parse HEAD
```

L'installation ne sert que sur un poste : l'image de l'exécutant contient déjà les dépendances. Node.js 24 exécute les fichiers `.ts` et fournit son lanceur de tests.

Compléter le pipeline

Vu en séance 3. Le dépôt `s3-construction`, ou `ts-s3-construction`, commence par cette seule tâche :

```
decouverte:
  image: vci-python:2026-09      # ou vci-node:2026-09
  tags: [teaching-ci]
  script: ["python --version"]   # ou "node --version"
```

Ajouts dans l'ordre. Avant chacun, écrire ce que le pipeline fera ; après, comparer.

1. Vérifier l'outillage : `- python -m pip check`, ou `- node --check charging.ts`, dans `script`.
2. Exécuter les tests : `- python -m pytest -q`, ou `- node --test` ; un défaut volontaire doit rendre la tâche rouge.
3. Conserver le rapport même en cas d'échec : `--junitxml=reports/junit.xml`, ou `--test-reporter=junit --test-reporter-destination=reports/junit.xml`, puis `artifacts: {when : always, reports: {junit: reports/junit.xml}}`.
4. Construire après les tests : une seconde tâche `package`, avec `needs: [tests]`, `script: [python build_artifact.py]` ou `script: [node build_artifact.ts]`, et `artifacts: {paths : [dist/]}`.
5. Ordonner les étapes : `stages: [test, package]` en tête, et `stage:` dans chaque tâche.

Vérifier un fichier livré

Vu en séance 3, repris en séances 4 et 5.

```
sha256sum dist/recharge.tar
tar -tf dist/recharge.tar
tar -xOf dist/recharge.tar charging.py   # ou charging.ts
# Sans sha256sum, l'une ou l'autre :
python -c "import hashlib,pathlib; print(hashlib.sha256(pathlib.Path('dist/recharge.
    tar').read_bytes()).hexdigest())"
node -e "console.log(require('node:crypto').createHash('sha256').update(require('node:
    fs').readFileSync('dist/recharge.tar')).digest('hex'))"
```

Trois informations à comparer : le commit attendu, le commit du manifeste, l'empreinte du fichier reçu. Une empreinte identique prouve les octets, pas la correction du programme.

Routine de décision

Vue en séances 4 et 5.

1. Nommer l'état lu et l'état à modifier.

2. Prédire une conséquence observable.
3. Préserver ce qui doit l'être.
4. Effectuer l'action.
5. Vérifier avec une observation qui pourrait montrer un échec.

Décider avec un verbe : livrer, reconstruire, réparer, attendre ou refuser.