

De l'exigence au prototype

Analyser un besoin, puis construire son outil de sauvegarde en C

Ressource R3.03 — BUT2 Informatique

4 TD et 4 TP de 1 h 15, en alternance — travail en groupe de 3 ou 4

Abstract

Huit séances pour mener une analyse complète et en tirer un outil qui fonctionne. Le laboratoire *BioSeq* demande un système de sauvegarde ; vous mènerez l'enquête, vous formulerez les exigences, vous les formaliserez, vous modéliserez le domaine, puis vous écrirez en C votre propre outil de sauvegarde incrémentale sur système Unix.

Les séances vont par paires : **chaque TP exploite directement ce que le TD qui le précède vient de produire**. Vous n'avez donc rien à préparer entre un TD et son TP — ce que vous avez décidé en TD, vous le codez en TP.

Contents

Organisation générale	3
1 TD1 — De la demande au besoin, et les exigences	7
1.1 Confrontation des lectures (15 min)	7
1.2 Jeu de rôle : l'entretien qui n'a pas eu lieu (20 min)	7
1.3 Écrire les exigences (25 min)	7
1.4 La frontière EF / ENF (15 min)	8
2 TP1 — Laboratoire système et socle du projet	9
2.1 Le laboratoire d'observation (30 min)	9
2.2 Vos seuils tiennent-ils ? (15 min)	9
2.3 Le socle du projet (30 min)	10
3 TD2 — User Stories, critères d'acceptation, scénario nominal	11
3.1 Correction croisée (15 min)	11
3.2 Débat : la story qu'il ne fallait pas écrire (20 min)	11
3.3 Des critères d'acceptation exécutables (20 min)	11
3.4 Le scénario nominal, et ce qu'il révèle (20 min)	11
4 TP2 — Le port et l'adaptateur POSIX	13
4.1 Pourquoi une interface ? (10 min)	13
4.2 Écrire le contrat (20 min)	13
4.3 Écrire l'adaptateur (35 min)	13
4.4 Vérification (10 min)	14
5 TD3 — Les exceptions, et comment on les testera	15
5.1 Le jeu des pannes (30 min)	15
5.2 Mise en commun et arbitrages (25 min)	15
5.3 Comment testera-t-on tout cela ? (20 min)	16

6	TP3 — L'incrémental et le laboratoire de mesures	17
6.1	Point de blocage collectif (10 min)	17
6.2	La règle de gestion, isolée (10 min)	17
6.3	Brancher l'incrémental et ses replis (25 min)	17
6.4	Le laboratoire de mesures (25 min)	17
6.5	Attribution des variantes individuelles (5 min)	18
7	TD4 — Modèle du domaine, processus, automate à états	19
7.1	Le tri (20 min)	19
7.2	Associations, multiplicités, classe-association (20 min)	19
7.3	Le processus enrichit le modèle (15 min)	19
7.4	L'automate à états (20 min)	20
8	TP4 — Robustesse et revue croisée	21
8.1	L'automate est-il respecté ? (25 min)	21
8.2	Corriger : l'atomicité (25 min)	21
8.3	Revue croisée — note G3 (25 min)	21
	Les variantes individuelles	23
	Grilles d'évaluation	23
	Mémo : les appels système	24
	Auto-évaluation	24

Organisation générale

Ce que vous construisez

Le laboratoire de recherche *BioSeq* vous demande un outil de sauvegarde. En huit séances, vous mènerez l'analyse complète de ce besoin, puis vous écrirez **votre propre outil** en C sur système Unix : chaque exécution produit un *instantané* — une copie complète et navigable d'un répertoire — mais les fichiers inchangés depuis l'instantané précédent ne sont pas recopiés. Trente instantanés tiennent alors dans l'espace d'un seul. C'est le principe de *rsnapshot* et de *Time Machine*.

Ce n'est pas un module de programmation C

Le code est le moyen, pas le but. Ce qui est évalué, c'est votre capacité à faire **descendre une exigence jusqu'au code, et à la faire remonter jusqu'à un test**. À tout moment, chaque membre du groupe doit pouvoir répondre à : « *quelle exigence cette fonction sert-elle, et quel test le prouve ?* »

Le principe : TD puis TP, immédiatement

Les séances vont par paires. **Chaque TP exploite directement ce que le TD qui le précède vient de produire** — et rien d'autre. Vous n'avez donc aucun travail à faire entre un TD et son TP : ce que vous avez décidé en TD, vous le codez en TP.

Paire	Le TD produit...	... que le TP consomme
1	des exigences chiffrées	un laboratoire qui vérifie qu'elles sont crédibles
2	le scénario nominal du cas d'utilisation	la liste des opérations système, donc le port et son adaptateur
3	les scénarios d'exception (cartes incident)	le code de l'incrémental et de ses replis
4	l' automate à états de l'instantané	l'expérience qui prend votre code en défaut, et sa correction

Le travail personnel se place **entre deux paires**, jamais à l'intérieur d'une paire. Sur 1 h 15, tout ce qu'un étudiant peut faire seul est du temps encadré gaspillé : le temps de séance est réservé à ce qui *ne peut pas* se faire seul — confronter des interprétations, arbitrer, chercher collectivement les cas d'exception, débloquer un bug système, mesurer et interpréter à plusieurs.

Budget horaire

Séance	Durée	Objet	Après
TAM 0	—	lire le dossier BioSeq, surligner demande / besoin	45 min
TD1	1 h 15	De la demande au besoin ; les exigences	
TP1	1 h 15	Laboratoire système, socle du projet	1 h 30
TD2	1 h 15	User Stories, Gherkin, scénario nominal	
TP2	1 h 15	Le port et l'adaptateur POSIX	2 h
TD3	1 h 15	Les exceptions, et comment on les testera	
TP3	1 h 15	L'incrémental et le laboratoire de mesures	2 h 30
TD4	1 h 15	Modèle du domaine, processus, automate	
TP4	1 h 15	Robustesse et revue croisée	2 h 30
Total	encadré : $8 \times 1 \text{ h } 15$	travail personnel	10 h ≈ 9 h

Le groupe : 3 ou 4, avec des rôles qui tournent

Vous travaillez en groupe de **3 ou 4**. Un groupe de quatre où une seule personne tape n'apprend pas quatre fois plus vite : il apprend pour une. Les rôles sont donc explicites et **changent à chaque séance**, de sorte que chacun tienne chaque rôle au moins une fois.

♥ Répartition des rôles — En TD

Client	porte la voix de Karim. Relit la transcription et refuse tout ce qui n'y figure pas : « ça, il ne l'a jamais dit ».
Scribe	tient la production écrite du groupe. C'est elle qui alimentera le dossier.
Avocat du diable	cherche systématiquement le contre-exemple, le cas qui casse.
Rapporteur (groupe de 4)	présente les conclusions du groupe en mise en commun. À 3, le scribe assure ce rôle.

♥ Répartition des rôles — En TP

Pilote	tient le clavier. Il n'écrit que ce qui a été dit à voix haute.
Copilote	lit la spécification à voix haute, relit ce qui est tapé, surveille les avertissements du compilateur.
Testeur	prépare les jeux d'essai dans un second terminal, exécute le point d'arrêt.
Intendant (groupe de 4)	tient le journal des décisions, les mesures, le README, et prépare les commits. À 3, ce rôle est partagé.

Rotation obligatoire : le pilote du TP_n devient testeur au TP_n+1. Notez la répartition de chaque séance dans votre journal de décisions : elle est vérifiée.

✗ À ne pas faire : L'organisation qui fait échouer les groupes de 4

« On se répartit : toi le code, moi les tests, vous le dossier. » Au bout de deux séances, personne ne comprend plus le travail des autres, le dossier décrit un code qui n'existe pas, et la note individuelle est catastrophique pour trois d'entre vous. **Vous travaillez sur un seul écran, ensemble**, sauf indication contraire explicite dans le sujet.

Règles de production

- Un **dépôt Git par groupe**, créé au TP1. Chaque membre commite sous sa propre identité — le journal Git est une pièce de l'évaluation individuelle. Message de commit référençant l'exigence traitée : "TP3: RG-02 detection des fichiers inchanges".
- Compilation obligatoire avec `cc -std=c11 -Wall -Wextra -Wpedantic`. **Un avertissement = un défaut.**
- Aucune bibliothèque externe : libc et POSIX.1-2008 uniquement. Le code doit fonctionner à **l'identique sur Linux et macOS** : pas de `#ifdef __linux__`, pas de `copy_file_range()`.
- Interdiction d'appeler `system()`, `popen()`, ou d'invoquer `cp`, `ln`, `rsync`. Vous écrivez l'outil, vous ne l'habilitez pas.
- **Ne testez jamais sur vos vraies données.** Travaillez exclusivement dans `/tmp`. Vous écrivez un programme qui crée des fichiers : une erreur de chemin est vite arrivée.

Évaluation : trois notes de groupe, une note individuelle

Évaluation — Vue d'ensemble

Note	Porte sur	Rendu	Coef.
G1 <i>groupe</i>	Spécification : exigences, backlog de User Stories, cas d'utilisation complet (TD1–TD3)	après le TP3	2
G2 <i>groupe</i>	Prototype : code, tests, modèle de domaine, matrice de traçabilité (TD4, TP1–TP4)	final	3
G3 <i>groupe</i>	Revue croisée : le rapport que votre groupe produit sur l'outil d'un autre groupe	fin du TP4	1
I <i>individuelle</i>	Contribution personnelle : évaluée sur le journal Git du dépôt de votre groupe	final	2

La note individuelle, en détail

Elle est établie à **partir du seul journal Git** de votre dépôt. Aucun oral, aucun document supplémentaire : ce que vous avez fait doit être lisible dans l'historique du projet.

Élément	Attendu	Pts
Votre variante	Chaque membre reçoit sa variante au TP3 (liste en fin de fascicule). Elle doit être implémentée, testée, et commitée sous votre identité . Le README indique qui porte quelle variante et quelle exigence elle satisfait.	12
Votre présence dans l'historique	Au moins un commit de votre main à chaque séance de TP , et une contribution au code commun (pas seulement à votre variante). Messages référant l'exigence traitée.	8

✗ À ne pas faire : Ce que l'enseignant verra, et que vous ne pouvez pas contourner

Le dépôt sera audité automatiquement : nombre de commits *et* volume de lignes modifiées par auteur, répartition dans le temps, et fichiers touchés par chacun.

- Cinquante commits d'une ligne le dernier soir ne valent pas mieux que zéro : c'est la **répartition dans le temps** qui est regardée.
- Un membre dont tous les commits sont concentrés sur un seul fichier n'a pas contribué au projet, il a fait son exercice.
- Committer à la place d'un camarade (ou sous son nom) est une fraude, au même titre qu'un plagiat.

Conséquence directe sur votre façon de travailler

Le *pilote* tape, mais **c'est le dépôt du groupe** : à chaque rotation de rôle, le nouveau pilote committe sous son identité. Un groupe qui laisse une seule personne au clavier pendant quatre TP produit un historique où trois membres sont absents — et trois notes individuelles à l'avenant.

Vérifiez dès le TP1 : `git config user.name` et `git config user.email` doivent être renseignés sur chaque poste, avec votre adresse universitaire.

La matrice de contribution

À joindre au rendu final : un tableau nominatif indiquant, pour chaque livrable, qui a piloté et qui a contribué. **Signé par tous les membres**. En cas de déséquilibre manifeste et non signalé, les notes de groupe peuvent être modulées individuellement.

 Travail à la maison — TAM 0 — avant la première séance, 45 min + 15 min de logistique

A. La lecture — individuelle (45 min). Chaque membre du groupe arrive avec sa propre lecture : c'est leur confrontation qui ouvre le TD1.

1. Lire le **dossier client BioSeq** : le contexte, le courriel de Karim et la transcription de l'entretien.
2. Surligner de deux couleurs : en jaune ce que Karim **demande**, en vert ce dont il a manifestement **besoin**. Ce ne sont pas les mêmes phrases.
3. Relever les **trois faits chiffrés** de l'entretien : ils serviront à justifier des exigences dès le TD1.
4. Arriver avec **une question** que vous auriez posée à Karim et qui n'a pas été posée dans l'entretien.

B. La logistique — collective (15 min). Le TD1 et le TP1 se suivent sans intervalle : tout doit être en place avant le premier cours.

1. **Constituer votre groupe** de 3 ou 4, et le déclarer. Chaque membre doit savoir avec qui il travaille *avant* d'entrer en salle : on ne forme pas les groupes pendant la séance.
2. **Choisir un nom de groupe**, en minuscules, sans accent ni espace (lynx, sequoia, lampyre...). Il servira de nom de dépôt.
3. **Créer le dépôt** sur la forge de l'établissement, nommé r303-<nom-de-groupe>, en **visibilité privée**.
4. **Y ajouter votre enseignant** avec le rôle *Maintainer* (GitLab) ou *Admin* (GitHub). Sans cet accès, votre travail ne peut être ni suivi ni évalué.
5. **Configurer Git sur chaque poste** que vous utiliserez, avec votre adresse universitaire :

```
git config --global user.name "Prenom Nom"
git config --global user.email "prenom.nom@etu.univ-exemple.fr"
```

 À ne pas faire : Pourquoi ces cinq points ne sont pas de l'intendance

Votre **note individuelle** est établie sur le seul journal Git. Trois conséquences immédiates :

- un commit signé root@localhost ou Utilisateur n'est attribuable à personne : il ne compte pour aucun d'entre vous ;
- un dépôt auquel l'enseignant n'a pas accès est un dépôt qui n'existe pas ;
- un groupe qui se forme pendant le TD1 perd le quart de la séance — et le TP1 suit immédiatement.

Ces quinze minutes sont les plus rentables du semestre.

1 TD1 — De la demande au besoin, et les exigences

Objectifs de la séance

1. Distinguer une demande (une solution déjà choisie) d'un besoin (un problème à résoudre).
2. Formuler des exigences **vérifiables**, fonctionnelles et non fonctionnelles.
3. Produire les **seuils chiffrés** que le TP1 mettra à l'épreuve.

1.1 Confrontation des lectures (15 min)

Dans chaque groupe, les membres comparent leurs surlignages. Le *client* arbitre : une phrase ne peut être classée « besoin » que si elle s'appuie sur ce que Karim a réellement dit.

Question de dossier

Q1.1 — Formulez en une phrase la **demande** de Karim, puis en une phrase son **besoin**. Combien de mots ont-ils en commun ?

Déroulement collectif des « 5 Pourquoi » à partir de « je veux un tar.gz chaque nuit », au tableau.

Question de dossier

Q1.2 — À quel « pourquoi » la cause racine apparaît-elle ? Reformulez-la. En quoi rend-elle la solution initialement demandée non seulement insuffisante, mais **contre-productive** ?

1.2 Jeu de rôle : l'entretien qui n'a pas eu lieu (20 min)

L'enseignant joue Karim. Chaque groupe pose **deux** questions, par la voix de son rapporteur — celles préparées en TAM 0. Les autres groupes classent chaque question au vol : *ouverte* ou *fermée* / *suggestive*.

✓ À faire : La règle du jeu

Karim répond **strictement** à ce qu'on lui demande. Une question fermée obtient « oui » ou « non », et rien d'autre. Une question qui suggère une solution obtient un acquiescement poli. Vous verrez très vite le coût d'une mauvaise question.

Question de dossier

Q1.3 — Relevez deux questions fermées ou suggestives posées pendant le jeu et reformulez-les. Pour chacune, indiquez l'information qu'elle aurait permis d'obtenir.

1.3 Écrire les exigences (25 min)

Par groupe, à partir des problèmes identifiés :

- **3 exigences fonctionnelles**, chacune accompagnée de la **commande** ou de l'**observation** qui prouvera qu'elle est satisfaite ;
- **3 exigences non fonctionnelles** tirées de trois familles ISO 25010 *différentes*, chacune avec une **métrique chiffrée**.

✗ À ne pas faire : Les formulations qui seront refusées

« Le système doit être rapide », « l'outil doit être fiable », « la sauvegarde doit bien se passer ». Si vous ne pouvez pas écrire la commande qui prouve qu'une exigence est satisfaite, ce n'est pas une exigence : c'est un vœu.

Balayage collectif des huit caractéristiques ISO 25010 : pour chacune, la classe cherche *une* exigence qui manquait. C'est l'exercice qui fait apparaître **ENF-COMP-01** — *un instantané doit rester exploitable avec les outils Unix standard, sans notre outil* — que personne ne formule spontanément.

Question de dossier

Q1.4 — Pourquoi **ENF-COMP-01** est-elle sans doute l'exigence la plus importante du cahier, alors qu'aucun utilisateur ne l'a demandée ? Quelles solutions techniques disqualifie-t-elle d'emblée ?

1.4 La frontière EF / ENF (15 min)

Classification collective des huit énoncés distribués. Le premier — « le système doit créer un lien dur plutôt qu'une copie pour un fichier inchangé » — est un piège : ce n'est ni une EF ni une ENF, c'est **une solution déguisée en exigence**. Corrigez-la, ne la classez pas.

✓✓ Point d'arrêt — Ce que vous emportez au TP1

Trois ENF avec des seuils chiffrés, écrites de votre main. **Au TP1, vous allez vérifier au clavier si ces seuils tiennent debout** — et découvrir le mécanisme système qui rend l'exigence centrale réalisable.

2 TP1 — Laboratoire système et socle du projet

Objectifs de la séance

1. Comprendre *expérimentalement* le mécanisme qui rend la sauvegarde incrémentale possible.
2. Confronter les seuils chiffrés du TD1 à la réalité de la machine.
3. Mettre en place le projet : dépôt, Makefile, premier programme.

2.1 Le laboratoire d'observation (30 min)

Le *testeur* est au clavier pour cette partie, le *scribe* note les observations.

```
mkdir -p /tmp/lab0 && cd /tmp/lab0
echo "version 1" > original.txt
ls -li original.txt           # notez l'inode ET le compteur de liens

ln original.txt copie_dure.txt # lien DUR (surtout pas -s !)
ln -s original.txt copie_douce.txt
ls -li ; du -sh /tmp/lab0

echo "version 2" >> original.txt
cat copie_dure.txt           # que contient le lien dur ?

rm original.txt
cat copie_dure.txt           # et maintenant ?
cat copie_douce.txt          # et le lien symbolique ?

# Sous Linux : /dev/shm est un autre systeme de fichiers (tmpfs).
# Sous macOS : remplacez /dev/shm par un point de montage distinct
# (cle USB, ou image disque montee).
ln /tmp/lab0/copie_dure.txt /dev/shm/x
ln copie_dure.txt /tmp/lab0  # et sur un repertoire ?
```

Questions de dossier

- Q2.1** — Que vaut le compteur de liens (2^e colonne de `ls -li`) avant et après la création du lien dur ?
- Q2.2** — Après `rm original.txt`, pourquoi `copie_dure.txt` contient-il toujours les données alors que `copie_douce.txt` est cassé ? Reformulez en une phrase ce que `rm` supprime réellement.
- Q2.3** — Un lien dur consomme-t-il de l'espace disque ? Justifiez avec `du`.
- Q2.4** — *La question qui compte.* Expliquez à Karim, qui n'est pas informaticien, pourquoi ce mécanisme lui permet de garder 30 jours d'historique au lieu de 2. **Interdiction d'employer les mots « inode » et « lien dur ».**
- Q2.5** — Les deux dernières commandes échouent. Quelles **limites** des liens durs découvrez-vous ? Notez-les : elles reviendront au TD3 sous forme de cas d'exception.

2.2 Vos seuils tiennent-ils ? (15 min)

Reprenez les trois ENF chiffrées écrites au TD1. Confrontez-les à une mesure réelle.

```
mkdir -p /tmp/mesure && cd /tmp/mesure
for i in $(seq 1 2000); do echo "contenu $i" > f$i.txt; done

time cp -r /tmp/mesure /tmp/copie1 # combien de temps pour 2000 fichiers ?
time find /tmp/mesure -type f | wc -l # et pour les parcourir seulement ?
df -h /tmp                          # de combien d'espace disposez-vous ?
```

Questions de dossier

- Q2.6** — Extrapolez : combien de temps prendrait une copie intégrale des 400 Gio de BioSeq ? Votre seuil de performance du TD1 était-il réaliste ? Corrigez-le si nécessaire, en justifiant.
- Q2.7** — Une exigence chiffrée qui s'avère intenable après mesure : que fait-on ? On la baisse, on

change de solution, ou on renégocie le besoin avec le client ? Argumentez.

✓ À faire : Ce que vous venez de faire s'appelle un *spike*

Une courte investigation technique destinée à réduire l'incertitude sur une exigence, avant de s'engager. Ce n'est pas du développement : le code produit est jetable. Ce qui reste, c'est la **décision informée**.

2.3 Le socle du projet (30 min)

Le *pilote* prend le clavier.

1. Verser le squelette `squelette-tp/` dans le dépôt créé avant la séance, puis pousser un premier commit. **Chaque membre présent en pousse un** : c'est la seule façon de vérifier que les identités sont correctes.
2. **Choisir le nom de votre outil** — distinct du nom de votre dépôt, et vôtre pour sept séances. L'inscrire dans le `README.md` avec une phrase décrivant le besoin qu'il résout (pas ce qu'il fait : ce qu'il *résout*).
3. Écrire dans `main.c` un parcours récursif affichant, pour chaque entrée : son chemin complet, sa taille, sa date de modification et son type.

✓ À faire : Trois pièges, à connaître avant de les rencontrer

1. Oublier "." et ".." → récursion infinie.
2. `e->d_type` n'existe pas partout et vaut souvent `DT_UNKNOWN`. Utilisez `lstat()` et `S_ISDIR()`.
3. `snprintf` **peut tronquer**. Testez sa valeur de retour : si elle est $\geq$ à la taille du tampon, vous manipulez un chemin faux.

✓✓ Point d'arrêt — À montrer avant de partir

- `./mon-outil /tmp/labo` affiche l'arborescence sans boucler ; un chemin inexistant produit un message sur `stderr` et un code de retour non nul ;
- l'enseignant accède bien à votre dépôt distant ;
- `git shortlog -sne` fait apparaître **tous les membres présents**, chacun avec son adresse universitaire.

Si un nom manque ou si une adresse est incorrecte, corrigez-le **maintenant** : réécrire l'historique plus tard est pénible, et l'oublier vous coûtera des points.

📖 Travail à la maison — Après le TP1, 1 h 30

1. **Groupe** : compléter la liste à 6 EF et 6 ENF, toutes vérifiables, les ENF couvrant au moins quatre familles ISO 25010. Rédiger Q1.1 à Q2.7.
2. **Groupe** : terminer le parcours récursif si le point d'arrêt n'a pas été atteint.
3. **Individuel** : chaque membre rédige **deux User Stories** au format « *En tant que ... , je veux ... , afin de ...* », avec des rôles différents. Elles seront corrigées entre vous au TD2. Ne cherchez pas la perfection.

3 TD2 — User Stories, critères d'acceptation, scénario nominal

Objectifs de la séance

1. Écrire des *User Stories* centrées sur la valeur, pas sur l'interface ni sur la technique.
2. Traduire une story en critères d'acceptation exécutables.
3. Produire le **scénario nominal** dont le TP2 tirera l'architecture.

3.1 Correction croisée (15 min)

Les stories individuelles circulent à l'intérieur du groupe : chacun évalue celles de son voisin avec INVEST. Une croix par lettre défaillante, avec une justification en cinq mots maximum.

✓ À faire : Les trois défauts que vous allez trouver

Ils reviennent toujours : le **rôle générique** (« en tant qu'utilisateur »), le **bénéfice tautologique** (« afin de pouvoir le faire »), et la **story qui décrit un écran** plutôt qu'un besoin.

3.2 Débat : la story qu'il ne fallait pas écrire (20 min)

Au tableau :

« En tant que développeur, je veux utiliser les liens durs POSIX afin d'optimiser le stockage. »

Beaucoup l'auront écrite après le TP1 — elle décrit la trouvaille technique du projet. L'avocat du diable de chaque groupe est sollicité en premier. Trois questions dirigées : *qui* est le développeur dans cette phrase ? *quel* bénéfice le laboratoire en retire-t-il ? et : si demain le système de fichiers offre un mécanisme de déduplication natif, cette story est-elle encore valide ?

Question de dossier

Q3.1 — Réécrivez cette story au niveau du besoin. Montrez que votre version reste valide même si l'implémentation change complètement. Que gagne l'équipe technique à cette formulation ?

3.3 Des critères d'acceptation exécutables (20 min)

Prenez **US-01** : « En tant qu'administrateur, je veux qu'une sauvegarde quotidienne ne consomme que l'espace des fichiers réellement modifiés, afin de conserver un mois d'historique sur le NAS existant plutôt que deux jours. »

Scenario : <un titre qui décrit une situation, pas une fonction>
 Etant donné <le contexte initial, vérifiable>
 Quand <une action, une seule>
 Alors <un résultat observable et mesurable>
 Et <un autre résultat observable>

Contrainte imposée

Vos critères ne doivent contenir **aucun nom de fonction, aucun type, aucune structure de données**. Ils doivent rester compréhensibles par Karim. C'est ce qui vous permettra, plus tard, de les traduire directement en script de test.

3.4 Le scénario nominal, et ce qu'il révèle (20 min)

Rédaction collective, au tableau, du scénario nominal du cas d'utilisation « Créer un instantané » : la suite numérotée des interactions quand tout se passe bien. Neuf étapes environ. **Aucun cas d'erreur** : ce sera l'objet du TD3.

Puis, immédiatement, la question qui prépare le TP2 :

L'exercice qui fait le pont

Relisez votre scénario nominal ligne à ligne et **listez les opérations élémentaires sur le système de fichiers** dont il a besoin. Rien d'autre : uniquement ce que le scénario réclame.

Q3.2 — Combien en trouvez-vous ? Comparez avec les autres groupes. Pourquoi le scénario nominal est-il un meilleur point de départ que « la liste des fonctions de la libc » ?

✓✓ Point d'arrêt — Ce que vous emportez au TP2

Une liste de quatre à six opérations, écrite au tableau. **C'est le contrat que vous allez coder au TP2**, et il sortira de votre analyse, pas d'un catalogue technique.

4 TP2 — Le port et l'adaptateur POSIX

Objectifs de la séance

1. Traduire une exigence de maintenabilité en décision d'architecture.
2. Écrire une **interface** en C, avec une structure de pointeurs de fonctions.
3. Isoler tous les appels système dans un unique fichier.

4.1 Pourquoi une interface ? (10 min)

Relisez **ENF-MAIN-01** : *la logique métier doit être testable sans accès au système de fichiers réel.*

Le raisonnement, à mener en groupe avant de coder

Vous avez découvert au TP1 qu'un lien dur ne peut pas franchir la frontière d'un système de fichiers. Votre outil devra donc gérer ce cas. Mais si votre fonction de sauvegarde appelle directement `link()`, **comment le testerez-vous** ? Il vous faudrait un second système de fichiers, des droits root et un montage dédié. Pour *un* test.

Q4.1 — Quel principe SOLID cette difficulté appelle-t-elle ? Énoncez-le, puis expliquez comment il résout le problème.

Le C n'a ni interface ni class. On obtient le même effet avec une structure de pointeurs de fonctions — exactement ce que fait le noyau Linux avec ses `struct file_operations`.

4.2 Écrire le contrat (20 min)

À partir de la liste d'opérations produite au TD2, écrivez `src/fs_port.h`.

```
1 typedef struct fs_port {
2     void *ctx;                                /* l'état de l'implémentation : le "this" */
3     int (*lister)(void *ctx, const char *chemin, entree_t *tab, size_t max);
4     int (*stat) (void *ctx, const char *chemin, entree_t *info);
5     /* ... à compléter d'après VOTRE liste du TD2 ... */
6 } fs_port_t;
7
8 const fs_port_t *fs_posix(void);             /* le SEUL symbole exporté par l'adaptateur */
```

Grille de contrôle — le copilote coche avant de passer à la suite

- ☐ au plus six opérations
- ☐ aucun en-tête système dans le `.h` (seulement `stddef`, `sys/types`, `time`)
- ☐ chaque opération prend `void *ctx` en premier argument
- ☐ une structure de métadonnées limitée aux champs utiles au scénario nominal
- ☐ les opérations d'écriture renvoient un code d'erreur, jamais `void`

Contrainte de conception

Si vous avez huit opérations, c'est que vous décrivez la libc plutôt que les besoins de votre scénario nominal. **Une interface qui expose tout n'abstrait rien.**

Q4.2 — À quoi sert `void *ctx` alors que l'implémentation POSIX n'en aura aucun usage (il vaudra `NULL`) ? À quel mécanisme de la programmation objet correspond-il ?

4.3 Écrire l'adaptateur (35 min)

`src/fs_posix.c` : toutes les fonctions `static`, sauf `fs_posix()`. Ce fichier est le **seul** du projet à inclure `<dirent.h>`, `<sys/stat.h>`, `<unistd.h>` et `<fcntl.h>`.

Opération	Exigence à respecter
lister	ignorer "." et ".." ; ne retenir que fichiers ordinaires et répertoires
stat	lstat et non stat : on ne suit pas les liens symboliques
creer_repertoire	idempotent : EEXIST n'est pas une erreur
copier	blocs de 64 Kio, écritures partielles gérées , retour de close() testé
copier	droits du fichier créé = droits de la source
lier_dur	renvoie un code d'erreur, ne décide jamais lui-même quoi en faire

Adaptez ensuite `main.c` pour qu'il parcoure un répertoire **à travers le port** : plus aucun `opendir()` en dehors de l'adaptateur.

4.4 Vérification (10 min)

```
make && make verif
./mon-outil /tmp/labo
```

Question de dossier

Q4.3 — `make verif` cherche les appels système hors de l'adaptateur. Ajoutez volontairement un `opendir()` dans `main.c` et vérifiez qu'elle échoue. Pourquoi cette vérification mécanique vaut-elle mieux qu'une discipline ?

✓✓ Point d'arrêt — À montrer avant de partir

Le parcours fonctionne à travers le port et `make verif` est vert.

🏠 Travail à la maison — Après le TP2, 2 h — le métier, sans l'incrémental

Écrivez `src/sauvegarde.h` et `.c`. Ce module ne connaît que `fs_port.h`.

```
1 code_t creer_instantane(const fs_port_t *fs,
2                         const char *source,
3                         const char *precedent, /* NULL : sauvegarde complete */
4                         const char *destination,
5                         rapport_t *rapport);
```

1. Définir l'énumération des codes de retour (succès, arguments invalides, source illisible, destination non créable, terminaison partielle) et les documenter dans le README : c'est l'interface publique de votre outil pour cron.
2. Implémenter le **scénario nominal du TD2, et lui seul** : préconditions, création de la destination, parcours récursif, copie, alimentation du rapport.
3. Rapport sur `stdout`, erreurs sur `stderr`, code de retour significatif.

Test à réussir avant le TD3 : `diff -r source depot/j1` ne signale aucun écart ; et une source inexistante ne laisse **aucune trace** dans le dépôt.

Q4.4 — Après un échec de précondition, le répertoire de destination a-t-il été créé ? Quelle promesse faite au client cela romprait-il ?

5 TD3 — Les exceptions, et comment on les testera

Objectifs de la séance

1. Découvrir les scénarios d'exception par une méthode systématique plutôt que par accident.
2. Arbitrer : certaines exceptions appellent une décision **métier**, pas technique.
3. Concevoir la doublure de test qui rendra ces exceptions vérifiables.

5.1 Le jeu des pannes (30 min)

Chaque groupe tire **trois cartes incident**. Pour chacune, il rédige le flux alternatif : à quelle étape du nominal l'incident survient-il, que fait le système, où le scénario reprend-il — ou s'arrête-t-il ?

Carte Incident

A	Le dépôt est plein au milieu du parcours.
B	Un fichier de la source est illisible (permission refusée).
C	Il n'existe aucun instantané antérieur : c'est la toute première sauvegarde.
D	Le dépôt est sur un autre disque : impossible d'y créer un lien vers l'instantané de la veille.
E	Le volume source est démonté pendant l'exécution.
F	Un chemin dépasse la taille du tampon prévu.
G	Un fichier est modifié <i>pendant</i> que le système le copie.
H	Un fichier existe dans la source mais pas dans l'instantané de référence.

✓ À faire : Le format attendu

- 7a. La création du lien échoue (dépôt sur un autre système de fichiers).
Le système copie le fichier au lieu de le lier et poursuit le parcours.
L'instantané reste valide ; seul l'espace consommé augmente.
Reprise à l'étape 8.

Numérotez d'après l'étape du nominal où l'incident survient. Terminez toujours par « **reprise à l'étape *n*** » ou « **le cas d'utilisation se termine** ».

Les cartes D et E, vous les avez rencontrées au TP1

Vous savez déjà, pour l'avoir constaté au clavier, qu'un lien dur ne franchit pas la frontière d'un système de fichiers. La carte D n'est donc pas une hypothèse d'école : c'est le comportement que vous avez observé.

5.2 Mise en commun et arbitrages (25 min)

Reconstitution collective du cas d'utilisation complet. Deux cartes appellent une véritable **décision métier**, et non une décision technique.

- **Carte D** : faut-il échouer, ou se replier sur la copie ? Chaque camp argumente — *l'avocat du diable* de chaque groupe est sollicité. La décision retenue au laboratoire : *la disponibilité de la sauvegarde prime sur l'économie d'espace*.
- **Carte B** : faut-il tout arrêter pour un seul fichier illisible ? Si l'on poursuit, **comment l'instantané obtenu se distingue-t-il d'un instantané complet** ? C'est de cette question que naît la notion de *statut*, dont le TD4 tirera l'automate à états.

Questions de dossier

Q5.1 — Pour les cartes B et D, exposez les deux options possibles, la décision retenue et sa justification métier. Qui, dans une vraie équipe, doit trancher : le développeur ou le client ?

Q5.2 — Combien de scénarios d'exception le cas d'utilisation complet contient-il, alors que **US-01** tient en une phrase ? Que dit cet écart sur l'usage respectif des deux artefacts ?

Rédigez enfin les préconditions, la postcondition de succès et la postcondition d'échec.

La postcondition d'échec est la plus importante

« Aucun instantané complet n'a été ajouté ; **les instantanés antérieurs sont inchangés.** »
 Cette phrase, écrite aujourd'hui sur le papier, sera mise en défaut par votre propre code au TP4.
 Retenez-la.

5.3 Comment testera-t-on tout cela ? (20 min)

Vous venez d'écrire huit flux alternatifs. Question directe : **comment vérifierez-vous que votre code les traite correctement** ? Prenez la carte D. Il vous faudrait deux systèmes de fichiers. La carte A : un disque réellement plein. La carte E : démonter un volume en plein parcours.

La réponse : une doublure de test

Puisque votre métier ne parle au système que par le port conçu au TP2, il suffit de lui fournir **une autre implémentation de ce port** — une arborescence en mémoire, qui échoue quand on le lui demande. C'est ce qu'on appelle une *doublure de test*.

Concevez-la sur papier, en groupe :

```

1 typedef struct {
2     char    chemin[MAX_CHEMIN];
3     off_t   taille;
4     time_t  mtime;
5     int     est_repertoire;
6 } noeud_t;
7
8 typedef struct {
9     noeud_t noeuds[64];
10    int      nb_noeuds;
11    char     journal[128][MAX_CHEMIN]; /* trace de chaque operation d'écriture */
12    int      nb_journal;
13    int      echec_lien_dur;           /* pour forcer le flux de la carte D */
14 } fake_fs_t;
```

Questions de dossier

Q5.3 — À quoi sert le champ `journal` ? Donnez un exemple d'affirmation que l'on peut vérifier grâce à lui et qu'on ne pourrait pas vérifier autrement.

Q5.4 — Cette doublure est-elle un *stub*, un *mock* ou un *fake* au sens de Meszaros ? La réponse peut être « les trois, selon l'usage » — à condition d'expliquer lequel dans quel cas.

✓✓ Point d'arrêt — Ce que vous emportez au TP3

Le cas d'utilisation complet, avec les décisions prises sur les cartes B et D. **Ce sont exactement les lignes de code que vous allez écrire.**

6 TP3 — L'incrémental et le laboratoire de mesures

Objectifs de la séance

1. Implémenter **RG-02** et **RG-03** sous une forme traçable.
2. Coder les flux alternatifs décidés au TD3, en particulier le repli de la carte D.
3. **Mesurer** le gain et le confronter à la promesse de **US-01**.

6.1 Point de blocage collectif (10 min)

Tour de table éclair : quels groupes n'ont pas réussi le diff -r du travail personnel ? Les trois causes habituelles sont la récursion qui perd le chemin relatif, le tampon de chemin, et l'ordre de création des répertoires. Déblocage en groupes croisés : un groupe qui a réussi explique, il ne code pas à la place.

6.2 La règle de gestion, isolée (10 min)

RG-02 : un fichier est réputé inchangé si sa taille et sa date de modification sont identiques à celles de l'instantané précédent. Écrivez-la comme une fonction à part entière, nommée d'après la règle :

```
1 /* RG-02 : detection d'un fichier inchangé depuis l'instantane precedent. */
2 static int est_inchange(const entree_t *courant, const entree_t *ancien)
3 {
4     /* ... */
5 }
```

Question de dossier

Q6.1 — Cette fonction tiendrait en une ligne au milieu de votre boucle. Donnez deux raisons **concrètes** d'en faire une fonction nommée : une liée aux tests, une liée à l'évolution du produit (Karim demandera un jour une détection par empreinte).

6.3 Brancher l'incrémental et ses replis (25 min)

Lorsqu'un instantané précédent est fourni et que le fichier est inchangé, créez un lien dur vers l'exemplaire du dépôt au lieu de copier. Puis codez les trois flux décidés au TD3 : la carte D (repli sur la copie), la carte B (fichier illisible : journaliser, compter, *poursuivre*) et la carte F (chemin tronqué : ne pas traiter l'entrée).

Vers quoi pointe le lien ?

Vers le fichier **dans l'instantané précédent**, jamais dans la source. Réfléchissez à ce qui se passerait autrement le jour où la source est modifiée : votre « instantané » d'hier changerait tout seul.

6.4 Le laboratoire de mesures (25 min)

Le testeur pilote cette partie.

```
mkdir -p /tmp/t3/src /tmp/t3/depot
for i in $(seq 1 20); do dd if=/dev/zero of=/tmp/t3/src/f$i.dat bs=1024 count=100; done

./mon-outil /tmp/t3/src /tmp/t3/depot/j1
echo "modification" >> /tmp/t3/src/f7.dat
./mon-outil /tmp/t3/src /tmp/t3/depot/j2 /tmp/t3/depot/j1

ls -li /tmp/t3/depot/j1 /tmp/t3/depot/j2 | head -30
du -sk /tmp/t3/depot/j1 ; du -sk /tmp/t3/depot/j2 ; du -sk /tmp/t3/depot
```

Mesure	Attendu	Obtenu
Fichiers copiés lors du 2 ^e instantané	1	
Fichiers liés lors du 2 ^e instantané	19	
Inodes identiques entre j1/f1.dat et j2/f1.dat	oui	
Inodes identiques entre j1/f7.dat et j2/f7.dat	non	
du -sk depot/j1 seul	≈ 2000	
du -sk depot/j2 seul	≈ 2000	
du -sk depot (les deux ensemble)	≈ 2100	

Questions de dossier

Q6.2 — du -sk depot/j2 lancé seul annonce 2000 Kio, mais du -sk depot n'en annonce que 2100 pour les deux instantanés réunis. Expliquez cette apparente contradiction. Laquelle des deux mesures répond à la question de Karim ?

Q6.3 — Vérifiez que j1/f7.dat n'a pas changé après la modification de la source. Quelle règle de gestion venez-vous de valider expérimentalement ?

Q6.4 — Supprimez entièrement depot/j1. depot/j2 reste-t-il exploitable ? Testez, puis expliquez à l'aide du compteur de liens observé au TP1.

Q6.5 — Comparez au seuil de performance corrigé en Q2.6. Une sauvegarde incrémentale quotidienne de BioSeq tiendrait-elle dans la fenêtre de nuit ?

6.5 Attribution des variantes individuelles (5 min)

Chaque membre du groupe reçoit sa variante personnelle (liste en fin de fascicule). Elle est à implémenter dans le travail personnel, et c'est le journal Git qui en attestera. Commencez par identifier l'exigence du cahier qu'elle satisfait.

✓✓ Point d'arrêt — À montrer avant de partir

ls -li prouve le partage d'inodes, le tableau de mesures est rempli, et diff -r source depot/j2 ne signale aucun écart.

🏠 Travail à la maison — Après le TP3, 2 h 30

- Groupe — rendu G1** : finaliser le **dossier de spécification** (exigences, backlog INVEST, critères Gherkin, cas d'utilisation complet, réponses Q1 à Q6). À rendre avant le TD4.
- Groupe** : écrire la **doublure** tests/fs_fake.c d'après la conception du TD3, puis les **cinq tests unitaires** du tableau ci-dessous. Ajouter la cible make test : moins d'une seconde, aucun fichier créé.
- Individuel** : démarrer votre variante. **Commitez régulièrement sous votre identité** : c'est la seule trace de votre contribution.

Test	Ce qu'il vérifie	Rattaché à
T1	sans instantané précédent, tout est copié ; la récursivité fonctionne	carte C
T2	un fichier inchangé est lié, pas copié	RG-02, RG-03
T3	un fichier dont la date diffère est recopié	RG-02
T4	si le lien échoue, repli sur la copie sans erreur	carte D
T5	une source invalide ne produit aucun effet de bord	précondition

Q6.6 — Le test T5 vérifie une **absence**. Pourquoi ce type de test est-il presque impossible à écrire proprement sans doublure ?

7 TD4 — Modèle du domaine, processus, automate à états

Objectifs de la séance

1. Extraire un modèle de domaine et le débarrasser de tout ce qui relève de la solution.
2. Constaté qu'un processus métier enrichit le modèle statique.
3. Produire l'**automate à états** dont le TP4 montrera qu'il n'est pas respecté par votre code.

7.1 Le tri (20 min)

Chaque groupe surligne les substantifs de son cas d'utilisation complet, puis tranche pour chaque candidat : **classe**, **attribut**, **fusion**, ou **rejet**.

Le cas « Lien dur »

Il figurera dans presque toutes les listes — vous venez de passer deux TP dessus. C'est la trouvaille technique du projet, et elle n'a **aucune place** dans un modèle de domaine. Karim ne dit jamais « lien dur » : il dit « je ne veux pas payer trente fois le même fichier ».

Q7.1 — Citez deux autres candidats de votre liste qui relèvent de la solution et non du métier. À quoi les reconnaît-on ?

Trois candidats méritent un débat : Administrateur/Chercheur (deux classes, ou une spécialisation ?), Rapport (un affichage éphémère, ou un concept qui a une existence propre ?) et Exclusion (un attribut, ou autre chose ?).

7.2 Associations, multiplicités, classe-association (20 min)

Les questions se posent au client, une par une, et chaque réponse fixe une multiplicité.

La question qui fait tout basculer

Une même source peut-elle relever de deux politiques de sauvegarde ? Karim répond : « oui, */data* est sauvegardé chaque nuit en local et chaque semaine sur le site distant — mais avec des exclusions différentes ».

Q7.2 — Pourquoi cette réponse rend-elle une classe-association **obligatoire** plutôt que commode ? Où les motifs d'exclusion pourraient-ils être stockés, sinon ?

Q7.3 — En base de données relationnelle, en quoi cette classe-association se traduira-t-elle ? Que garantira sa clé primaire ?

7.3 Le processus enrichit le modèle (15 min)

Lecture collective du processus « *Sauvegarde nocturne* » en neuf étapes. À chaque étape, une seule question : *notre modèle de classes permet-il de faire cela ?* Cinq manques apparaissent. Deux sont évidents, trois ne le sont pas.

Question de dossier

Q7.4 — Listez les cinq attributs ou classes que la description du processus fait apparaître, en indiquant l'étape qui les a révélés. Lequel n'auriez-vous jamais trouvé en restant sur le modèle statique ?

✓ À faire : L'étape 8 mérite un arrêt

Le processus dit : *le système notifie l'administrateur si et seulement si le statut n'est pas « complet »*. Karim a été clair : « si je reçois un mail toutes les nuits, dans un mois je les filtre, et le jour où

ça plante je ne le vois pas ». Notifier uniquement les échecs est une **exigence métier**, pas une paresse d'implémentation.

7.4 L'automate à états (20 min)

L'attribut statut, né de la carte B au TD3 et confirmé par le processus, ne prend qu'un nombre fini de valeurs. Vous décrivez donc, sans l'avoir nommé, un **automate à états finis**. Dessinez-le en groupe : états, état initial, états finaux, transitions et leur événement déclencheur.

L'intérêt de l'automate est ailleurs

Il ne sert pas à lister ce qui est permis, mais à rendre explicite **ce qui est interdit**. Énoncez les transitions interdites et justifiez chacune :

- *Complet* → *EnCours* : un instantané est **immuable**. On en refait un, on ne répare pas l'ancien.
- *Échoué* → *Complet* : un instantané échoué ne se rattrape pas.
- *Purgé* → quoi que ce soit : état final.

Q7.5 — Ces trois règles sont invisibles sur le diagramme de classes. Que perdrait une équipe qui ne dessinerait jamais d'automate ?

Q7.6 — Karim demande une reprise sur incident : « si la sauvegarde plante à 80 %, je veux la reprendre le lendemain ». Quelle transition cela introduirait-il ? Quelle règle viole-t-elle ? Proposez une modélisation qui satisfait le besoin **sans** la violer.

✓✓ Point d'arrêt — Ce que vous emportez au TP4

Un automate qui affirme qu'un instantané ne peut pas être à la fois interrompu et complet. **Au TP4, vous allez vérifier que votre code respecte cette affirmation.** Il ne la respecte pas.

8 TP4 — Robustesse et revue croisée

Objectifs de la séance

1. Découvrir par l'expérience qu'une exigence non testée n'est pas satisfaite.
2. Corriger par une écriture atomique.
3. Confronter son outil à celui d'un autre groupe, et le mettre à l'épreuve.

8.1 L'automate est-il respecté ? (25 min)

Vous avez affirmé au TD4 qu'un instantané interrompu ne peut pas être *Complet*. Vérifions.

```
mkdir -p /tmp/t4/src /tmp/t4/depot
for i in $(seq 1 400); do dd if=/dev/zero of=/tmp/t4/src/f$i.dat bs=1024 count=200; done
./mon-outil /tmp/t4/src /tmp/t4/depot/j1

./mon-outil /tmp/t4/src /tmp/t4/depot/j2 /tmp/t4/depot/j1 &
sleep 0.3 ; kill -9 $!          # coupure brutale en plein parcours

ls /tmp/t4/depot                # j2 existe-t-il ?
ls /tmp/t4/depot/j2 | wc -l     # combien de fichiers contient-il ?
./mon-outil /tmp/t4/src /tmp/t4/depot/j3 /tmp/t4/depot/j2
diff -r /tmp/t4/src /tmp/t4/depot/j3
```

Questions de dossier

Q8.1 — Que contient j2 après la coupure ? **Rien** ne le distingue-t-il d'un instantané complet ? Dans quel état de votre automate se trouve-t-il ?

Q8.2 — L'instantané j3, construit en prenant j2 pour référence, est-il correct ? Que révèle diff -r ? Comment un défaut peut-il ainsi *se propager* d'un instantané au suivant, nuit après nuit ?

Q8.3 — Quelle exigence est violée ? Pourquoi aucun de vos tests ne l'avait-il détectée ?

8.2 Corriger : l'atomicité (25 min)

Le principe est celui de tous les outils Unix qui écrivent des fichiers importants : **construire ailleurs, publier d'un coup**.

1. Construire l'instantané dans depot/.en-cours-<pid>.
2. Une fois le parcours terminé **et seulement alors**, le renommer vers son nom définitif avec `rename(2)`.
3. En cas d'échec fatal, laisser (ou supprimer) le répertoire de travail : dans les deux cas, **aucun instantané au nom valide n'apparaît**.
4. Ignorer les répertoires .en-cours-* lors de la recherche de l'instantané de référence.

Rejouez l'expérience, puis ajoutez-la comme test de non-régression.

Question de dossier

Q8.4 — Pourquoi `rename(2)` convient-il, alors qu'une boucle de déplacement fichier par fichier ne conviendrait pas ? Quelle condition faut-il respecter pour que `rename()` soit effectivement atomique ?

8.3 Revue croisée — note G3 (25 min)

Les groupes s'apparient et échangent leurs dépôts. **Vous n'ouvrez pas le code d'abord : vous exécutez.** Le rapport que vous produisez est remis en fin de séance et constitue votre note G3.

- ☐ `make` compile sans aucun avertissement
- ☐ `make` `verif` est vert
- ☐ `make` `test` passe en moins d'une seconde, sans créer de fichier
- ☐ une source inexistante ne laisse aucune trace dans le dépôt
- ☐ `kill -9` en cours d'exécution ne laisse aucun instantané au nom valide
- ☐ `ls -i` prouve le partage d'inodes entre deux instantanés
- ☐ le README documente les codes de retour
- ☐ chaque règle **RG-0x** correspond à une fonction nommée dans le code

✓ À faire : Le vrai exercice de la revue

Ne vous contentez pas de cocher. **Essayez de casser l'outil** : un chemin avec un espace, un répertoire vide, une destination déjà existante, un lien symbolique dans la source, une source qui est un fichier et non un répertoire. Chaque défaut trouvé est signalé par écrit, **avec la commande qui le reproduit**. C'est cela qui est noté : une revue qui ne trouve rien est une revue qui n'a pas cherché.

Question de dossier

Q8.5 — Rapportez les deux défauts les plus intéressants trouvés chez l'autre groupe et les deux trouvés chez vous. Pour chacun : à quelle étape (analyse, conception, codage, test) aurait-il pu être évité ?

✓✓ Point d'arrêt — Avant de partir

Chaque membre du groupe a poussé au moins un commit aujourd'hui, sous son identité. Vérifiez-le ensemble : `git shortlog -sne` doit faire apparaître tout le monde.

📖 Travail à la maison — Après le TP4, 2 h 30 — le rendu final G2

1. **Groupe** : corriger les défauts remontés par la revue croisée ; mettre au propre le modèle de domaine et l'automate à états ; écrire `tests/test_acceptation.sh` en traduisant les critères Gherkin du TD2.
2. **Groupe** : compléter la **matrice de traçabilité** et la **matrice de contribution** signée.
3. **Individuel** : terminer votre variante avec son test. **Le dernier commit fait foi** : une variante terminée mais jamais poussée n'existe pas.

La matrice de traçabilité (à joindre au rendu G2)

Problème	Exigence	US / CU	Où dans mon code	Quel test
P1	EF-02	US-01 / CU-01 §6–8	<code>est_inchange()</code>	T2, T3
...				

Une ligne par exigence. Une exigence sans colonne « test » remplie est une exigence non satisfaite, quel que soit l'état du code.

Les variantes individuelles

Attribuées au TP3, **une par membre du groupe** (un groupe de quatre en implémente donc quatre). Chacune doit être accompagnée d'au moins un test, et le README doit indiquer son auteur et l'exigence qu'elle satisfait.

C'est votre journal Git qui atteste de votre travail : commitez sous votre propre identité, en plusieurs étapes, et non en un seul dépôt final.

N°	Variante	Exigence
V1	Option <code>--exclude</code> MOTIF (répétable), via <code>fnmatch(3)</code>	EF-07
V2	Option <code>--simulation</code> : affiche ce qui serait fait, n'écrit rien	précondition, prudence
V3	Journalisation <code>syslog(3)</code> , niveau selon le statut	US-06
V4	Purge des instantanés de plus de N jours, en respectant RG-05	EF-06
V5	Rapport au format CSV (<code>--format=csv</code>) pour l'auditeur	US-07
V6	Commande <code>liste</code> : instantanés d'un dépôt, date et statut	EF-04
V7	Détection renforcée par empreinte calculée en interne	RG-02 renforcée
V8	Option <code>--profondeur</code> N, avec report des entrées ignorées	performance
V9	Commande <code>restaurer</code> : un fichier d'un instantané vers un chemin cible	EF-05
V10	Vérification préalable de l'espace disponible (<code>statvfs</code>)	carte A du TD3
V11	Option <code>--quiet</code> / <code>--verbose</code> et respect strict de <code>stdout/stderr</code>	ENF-UTI-01
V12	Statut persistant : un fichier <code>.statut</code> par instantané, lu par la recherche de référence	automate du TD4

Grilles d'évaluation

G1 — Spécification (groupe, coefficient 2, rendu avant le TD4)

Critère	Pts
Demande et besoin distingués ; cause racine identifiée et argumentée	3
6 EF vérifiables : chacune accompagnée de sa méthode de vérification	3
6 ENF mesurables couvrant ≥ 4 familles ISO 25010, dont une corrigée après la mesure du TP1	4
Backlog de User Stories : rôles réels et distincts, bénéfice métier, INVEST vérifié	3
Critères d'acceptation Gherkin, sans référence à l'implémentation	3
Cas d'utilisation complet : nominal, ≥ 6 flux alternatifs, pré/postconditions	4
Total	20

G2 — Prototype (groupe, coefficient 3, rendu final)

Critère	Pts
Compile sans aucun avertissement ; <code>make</code> <code>verif</code> vert	2
Port : ≤ 6 opérations, aucun en-tête système dans le <code>.h</code> , dérivé du scénario nominal	2
Sauvegarde complète et incrémentale conformes ; replis des cartes B, D et F codés	3
Règles de gestion identifiables : une fonction nommée par règle	2
Suite unitaire (≥ 5 tests), rapide, sans accès disque, avec la doublure	3
Test d'acceptation exécutable, traduit des critères Gherkin	2
Correction de l'atomicité, avec test de non-régression	2
Modèle de domaine sans classe technique ; classe-association justifiée ; automate à états	2
Matrice de traçabilité complète et exacte	2
Total	20

Pénalités : usage de `system()` ou d'une commande externe (−5) ; code non portable macOS/Linux sans justification (−2) ; tests qui écrivent hors de `/tmp` (−2).

G3 — Revue croisée (groupe, coefficient 1, remise en fin de TP4)

Critère	Pts
Les huit points de la grille sont renseignés avec la preuve d'exécution	6
Au moins deux défauts trouvés au-delà de la grille, chacun avec la commande qui le reproduit	8
Chaque défaut est rattaché à l'étape de la démarche qui aurait dû l'éviter	4
Ton professionnel : on décrit un comportement, on ne juge pas des personnes	2
Total	20

I — Note individuelle (coefficient 2, établie sur le journal Git)

Critère	Pts
Votre variante fonctionne et satisfait l'exigence annoncée dans le README	5
Elle est couverte par au moins un test, écrit par vous	4
Elle est entièrement committée sous votre identité, en plusieurs étapes	3
Au moins un commit de votre main à chacune des quatre séances de TP	4
Vous avez contribué au code commun, pas seulement à votre variante	2
Messages de commit exploitables, référençant l'exigence traitée	2
Total	20

Un historique où un membre est absent de deux séances sur quatre plafonne sa note individuelle, quelle que soit la qualité du rendu du groupe.

Évaluation — Calcul de la note finale

$$\text{Note} = \frac{2 \times G1 + 3 \times G2 + 1 \times G3 + 2 \times I}{8}$$

Les notes de groupe peuvent être modulées individuellement si la matrice de contribution, le journal Git font apparaître un déséquilibre manifeste et non signalé.

Mémo : les appels système du projet

Fonction	Manuel	Piège principal
opendir / readdir	man 3 readdir	ne pas oublier "." et ".." ; ne pas se fier à d_type
lstat	man 2 lstat	stat suit les liens symboliques, lstat non
mkdir	man 2 mkdir	EEXIST n'est pas une erreur ici ; le umask rabote les droits
open	man 2 open	le 3 ^e argument n'est lu qu'avec O_CREAT
read / write	man 2 write	les transferts partiels sont légaux : bouclez
close	man 2 close	peut échouer : c'est là que se voient les erreurs différées
link	man 2 link	impossible entre systèmes de fichiers ; impossible sur un répertoire
rename	man 2 rename	atomique seulement au sein d'un même système de fichiers
fnmatch	man 3 fnmatch	(V1) attention au drapeau FNM_PATHNAME
statvfs	man 3 statvfs	(V10) f_bavail et non f_bfree
syslog	man 3 syslog	(V3) openlog avant, closelog après

Auto-évaluation : sommes-nous prêts à rendre ?

À cocher avant le rendu final

- ☐ make ne produit aucun avertissement ; make verif est vert.
- ☐ make test passe en moins d'une seconde et ne crée aucun fichier.
- ☐ make acceptation passe.
- ☐ Une source invalide ne laisse aucune trace dans le dépôt.

- ☐ `kill -9` ne laisse aucun instantané au nom valide.
- ☐ Chaque règle **RG-0x** correspond à une fonction nommée.
- ☐ Le README documente les codes de retour.
- ☐ Chaque membre a au moins un commit sous son identité à chaque séance.
- ☐ `git shortlog -sne` montre les quatre membres, avec des volumes comparables.
- ☐ Chaque variante est nommée dans le README avec son auteur et l'exigence qu'elle satisfait.
- ☐ La matrice de contribution est remplie et signée par tous.
- ☐ Aucun test n'écrit en dehors de `/tmp`.