Fiche 03 : cut, tr, sed

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-11-10

Table des matières

1 Objectif 2
2 Ce que cette fiche ne contient pas 2
3 Spécification : transformations de texte 2
4 Démarrage 2
4.1 Environnement de travail 2
4.2 Matériel de la fiche 4
5 Ce que tu sauras faire 4
6 Les 5 commandes de la fiche 4
7 Découverte guidée 5
7.1 Découper avec `cut` 5
7.2 Nettoyer et transformer avec `tr` et `sed` 5
7.3 Enchaîner les étapes 6
7.4 Vérifier un décalage de colonnes 7
7.5 Deux sens de lecture pour la ligne de commande 7
8 Point de contrôle 8
8.1 Les acquis en 5 questions 8
9 Application autonome 9
10 Exercice 1 — `tr` : nettoyer une sortie 9
10.1 Consigne 9
10.2 Correction pas à pas 9
11 Exercice 2 — `sed` : remplacer partout 9
11.1 Consigne 9
11.2 Correction pas à pas 9
12 Exercice 3 — chaîne de nettoyage 10
12.1 Consigne 10
12.2 Correction pas à pas 10
13 Exercice 4 — `cut -c` et la position des colonnes 11
13.1 Consigne 11

13.2 Correction pas à pas 11
14 Exercice 5 — lire une ligne de commande 11
14.1 Consigne 11
14.2 Correction pas à pas 11
15 Exercice 6 — lire une valeur saisie 14
15.1 Consigne 14
15.2 Correction pas à pas 14
16 Récapitulatif des commandes de la fiche 15
17 Auto-vérification 16
17.1 Consigne 16
17.2 Correction pas à pas 16

1 Objectif

Construire des transformations successives de texte avec `cut`, `tr` et `sed`, les entraîner sur des entrées fixes et comparer les sorties obtenues aux sorties attendues.

2 Ce que cette fiche ne contient pas

Les expressions régulières et la recherche par motif (`grep`) sont vues en **fiche 02**, les tubes et l'ordre des étapes en **fiche 04**. Les commandes `sort`, `uniq`, `head` et `wc` sont réutilisées ici sans être réexpliquées.

3 Spécification : transformations de texte

Le répertoire de travail de cette fiche est `$HOME/work/assignments/shell_tp/fiche03`.

Les outils attendus sont :

- `cut` : découpe par délimiteur (`-d`, `-f`) ou par colonnes (`-c`) ;
- `tr` : conversion et compression de caractères (`-d`, `-s`, `--help`) ;
- `sed` : substitution (`s/anomalie/erreur/g`) ;
- la comparaison de la sortie obtenue à la sortie attendue.

---

4 Démarrage

4.1 Environnement de travail

Crée un dossier de travail dédié, puis les fichiers de test. Les sorties varient avec les colonnes et les séparateurs choisis.

```
WORK=~/work/assignments/shell_tp/fiche03
rm -rf "${WORK:?}"
mkdir -p "${WORK:?}" || { echo "ERREUR : setup"; exit 1; }
```

```
printf '%s\n' \
  'Bruno,bruno@example.com,06 12 34 56 78' \
  'Chen,chen@example.com,06 98 76 54 32' \
  'Diallo,diallo@example.com,06 45 67 89 01' \
  'Dupont,dupont@example.com,06 12 34 56 78' \
  'Nadia,nadia@example.com,06 11 22 33 44' \
  > "${WORK:?}/contacts.csv"
printf   '%s\n'   'Bruno:DT:12'   'Chen:PO:8'   'Diallo:DT:3'   'Dupont:PO:15'
'Nadia:DT:7' \
  > "${WORK:?}/notes.txt"
printf '%s\n' \
  'bruno:x:1000:1000:bruno:/home/bruno:/bin/zsh' \
  'etudiante:x:1001:1001:etudiante:/home/etudiante:/bin/bash' \
  'prof:x:100:100:prof:/home/prof:/bin/zsh' \
  > "${WORK:?}/notes.passwd"
echo "Répertoire réinitialisé : ${WORK}"
```

```
Répertoire réinitialisé : /home/jovyan/work/assignments/shell_tp/fiche03
```

`rm -rf "${WORK:?}"` supprime **uniquement** le dossier de la fiche : `${WORK:?}` arrête la commande si la variable était vide, pour ne jamais supprimer `.` par accident. Deux messages peuvent signaler cet incident, et il faut les connaître tous les deux. Le premier est `ERREUR : dossier absent` : la garde fait `exit 1`, ce qui **termine le shell** de la cellule ; le moteur de calcul en redémarre un neuf, `${WORK}` et le répertoire courant sont perdus, et la cellule en échec est malgré tout marquée comme exécutée. Le second est `bash: WORK: parameter null or not set` : la variable était déjà perdue ; dans ce cas la garde ne s'affiche même pas et **le reste de la cellule continue de s'exécuter dans un répertoire non maîtrisé** — ce second message vient du shell lui-même, pas de nous. Une seule recette est sûre dans les deux cas : relancer la cellule de préparation ci-dessus — elle recrée le dossier **depuis zéro**, tout ce que vous aviez produit dans `${WORK}` est donc perdu — **puis rejouer toutes les cellules de la fiche, dans l'ordre**. Un `pwd` exécuté juste après indique dans quel répertoire la cellule a réellement travaillé ; les seuls fichiers emportés par la manœuvre sont ceux de `${WORK}`.

> [!IMPORTANT] Ces fichiers sont fabriqués pour la fiche. Ils ne contiennent aucune donnée réelle : aucune adresse électronique, aucun numéro de téléphone, aucun compte système. La dernière partie, sur `notes.passwd`, est un **exercice de syntaxe** sur les `:` : les lignes ci-dessus ne sont pas de vrais comptes.

4.2 Matériel de la fiche

- Deux entrées fixes : `notes.txt` (colonnes séparées par `:`, 5 lignes) et `contacts.csv` (3 colonnes séparées par des virgules, dont une contient des espaces).
- Une table `notes.passwd` (7 colonnes séparées par `:`, un login en 1 champ, un identifiant numérique en 3, un shell en dernier champ) pour la dernière partie.
- Une méthode de contrôle unique : on note la sortie attendue, on lance la commande, on compare.
- Un repère d'organisation : les transformations s'enchaînent avec des tubes, étape par étape (la technique des tubes est vue en fiche 04).

5 Ce que tu sauras faire

- Extraire un champ avec `cut -d -f` et vérifier un décalage de colonnes avec `cut -c`.
- Supprimer et compresser des caractères avec `tr -d` et `tr -s`.
- Remplacer un motif avec `sed s///` et son indicateur global `g`.
- Construire une chaîne de transformations et la tester entrée par entrée.

6 Les 5 commandes de la fiche

Commande	Usage
`cut -d: -f1`	extrait le champ 1 avec `:` comme séparateur
`cut -c1`	extrait le 1 caractère de chaque ligne
`tr -d '0-9'`	supprime tous les chiffres
`tr -s ' '`	remplace chaque suite de plusieurs espaces par un seul espace
`sed 's/anomalie/erreur/g'`	remplace toutes les occurrences de `anomalie` par `erreur`

Un seul schéma de transformation, du début à la fin de la fiche :

```
entrée
  └── cut    extraire une partie d'une ligne
  └── tr     convertir, supprimer, compresser des caractères
  └── sed    remplacer un motif par une chaîne
```

> [!TIP] Outils de découverte `man cut`, `man tr`, `man sed` ouvrent la page de manuel complète ; `less` l'affiche écran par écran. Ces deux outils affichent une page plein écran qui attend une touche : ils ne fonctionnent pas dans ce site publié.

> Pour vérifier une option ici, utilise l'aide intégrée des commandes, comme `tr --help`, et les entrées fixes créées plus haut.

---

7 Découverte guidée

7.1 Découper avec `cut`

Quand une ligne regroupe plusieurs informations, on extrait seulement la colonne utile.

```
printf '%s\n' 'Bruno:DT:12' 'Chen:PO:8' | cut -d: -f1
```

```
Bruno
Chen
```

`-d:` définit le séparateur, `-f1` sélectionne le premier champ.

À faire : Extraire le deuxième champ (`DT`, `PO`), puis les deuxième et troisième champs dans cet ordre.

Réponse

```
printf '%s\n' 'Bruno:DT:12' 'Chen:PO:8' | cut -d: -f2
printf '%s\n' 'Bruno:DT:12' 'Chen:PO:8' | cut -d: -f2,3
```

7.2 Nettoyer et transformer avec `tr` et `sed`

`tr` s'occupe des **caractères**.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
mkdir -p fiche03-sandbox
cd fiche03-sandbox || { echo "ERREUR : dossier absent fiche03-sandbox"; exit 1; }
mkdir -p scripts
printf '%s\n' 'a1b2c3' | tr -d '0-9'
printf '%s\n' 'bruno--dt--12' | tr -s '-'
printf '%s\n' 'bonjour le monde' | tr 'a-z' 'A-Z'
```

```
abc
bruno-dt-12
BONJOUR LE MONDE
```

`tr -d '0-9'` supprime tous les chiffres. `tr -s '-'` remplace chaque suite de plusieurs `-` par un seul `-`. `tr 'a-z' 'A-Z'` convertit les minuscules en majuscules.

> [!NOTE] `tr` travaille caractère par caractère : les ensembles (`a-z`, `0-9`) doivent être équilibrés, et un séparateur comme une virgule se traite par position, pas par mot.

À faire : Converter une ligne en majuscules, puis supprimer tous les chiffres d'une chaîne mélangée.

Réponse

```
printf '%s\n' 'shell 2026' | tr 'a-z' 'A-Z'
printf '%s\n' 'abc123def456' | tr -d '0-9'
```

`sed` remplace du texte :

```
cd "${WORK:?}/fiche03-sandbox" || { echo "ERREUR : dossier absent ${WORK}/fiche03-sandbox — exécuter d'abord la cellule « Nettoyer et transformer avec tr et sed »"; exit 1; }
printf '%s\n' 'anomalie détectée, anomalie confirmée' | sed 's/anomalie/erreur/g'
```

```
erreur détectée, erreur confirmée
```

> [!IMPORTANT] Sans `g`, `sed 's/anomalie/erreur/'` ne remplace que **la première occurrence** de chaque ligne. Pense à l'indicateur global `g`.

À faire : Remplacer `PO` par `Projet`, puis `DT` par `Démo`, sur le fichier `notes.txt`.

Réponse

```
sed 's/PO/Projet/' notes.txt
```

7.3 Enchaîner les étapes

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
printf '%s\n' 'Dupont,  dupont@example.com,  06 12 34 56 78' \
  | tr -s ' ' \
  | cut -d, -f1 \
  | tr 'a-z' 'A-Z' \
  | sed 's/^/Fiche03 /'
```

```
Fiche03 DUPONT
```

7.4 Vérifier un décalage de colonnes

Un même fichier, trois lectures, trois sorties différentes (les colonnes déterminent tout) :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
echo 'coupe par champs (cut -d: -f3)'; printf '%s\n' 'bruno:x:1000:1000' | cut
-d: -f3
echo "coupe par colonnes (cut -c3)   "; printf '%s\n' 'bruno:x:1000:1000' | cut
-c3
echo 'convertit les caractères (tr)  '; printf '%s\n' 'bruno:x:1000:1000' | tr
':' '#'
```

```
coupe par champs (cut -d: -f3)
1000
coupe par colonnes (cut -c3)
u
convertit les caractères (tr)
bruno#x#1000#1000
```

7.5 Deux sens de lecture pour la ligne de commande

Le même fichier se lit dans deux sens différents. Les deux sont utiles : il n'y a pas de bonne direction, seulement celle qui correspond à ta question. Ici, `<<<` injecte directement une chaîne dans la commande, sans fichier ni tube.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
echo "1. cut -d: -f1 : le login (1er champ)"
cut -d: -f1 notes.passwd
echo '2. cut -d: -f6 : le répertoire personnel'
cut -d: -f6 notes.passwd
```

```
1. cut -d: -f1 : le login (1er champ)
bruno
etudiante
prof
2. cut -d: -f6 : le répertoire personnel
/home/bruno
/home/etudiante
/home/prof
```

[!TIP] Tu peux lire le fichier dans l'autre sens : partir du dernier champ (le shell) et regrouper les identifiants qui l'utilisent. Les commandes `sort` et `uniq` font ce travail, elles sont vues en **fiche 04**.

`sed` s'applique aussi à une ligne de commande, mais son motif doit rester très simple (ici, un mot entier) et son effet doit être visible immédiatement.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
sed 's#/home/data/facture-2026.pdf#~/Documents/facture-2026.pdf#' \
  <<< 'Je dois ouvrir /home/data/facture-2026.pdf'
```

```
Je dois ouvrir ~/Documents/facture-2026.pdf
```

Autre possibilité, avec un autre séparateur pour éviter d'écrire `\/` partout :

```
sed 's|/home/data/facture-2026.pdf|~/Documents/facture-2026.pdf|'
```

---

8 Point de contrôle

8.1 Les acquis en 5 questions

1. Avec `cut -d: -f2`, que récupère-t-on ?
2. Quelle est la différence entre `tr -d '-'` et `tr -s '-'` ?
3. Que fait `sed 's/DT/Démo/'` ? Pourquoi cette forme peut-elle être insuffisante ?
4. Un fichier a une colonne qui peut contenir des `:`. Que se passe-t-il si on découpe avec `-d:` ?
5. Où regarder l'aide d'une commande quand la page `man` ne s'affiche pas ?

Corrigé

1. Le 2 champ, en utilisant `:` comme séparateur.
2. `-d` supprime tous les `-` ; `-s` les compresse (une suite de plusieurs `-` devient un seul `-`).
3. Remplace la première occurrence de `DT` par `Démo` sur chaque ligne ; il faut souvent `s/DT/Démo/g` pour tout remplacer.
4. Découpage faux : les champs sont décalés. La sortie produite est fausse, elle n'est pas correcte.
5. L'aide intégrée de la commande, par exemple `tr --help` ou `cut --help` (la page `man` ne s'affiche pas dans ce site publié).

---

9 Application autonome

10 Exercice 1 — `tr` : nettoyer une sortie

10.1 Consigne

1. Construis une commande qui, à partir de la ligne `a1b2c3`, enlève tous les chiffres.
2. Construis une commande qui, à partir de `a  b  c`, remplace chaque suite de plusieurs espaces par un seul espace.

10.2 Correction pas à pas

1. Attendu : `abc`, puis `a b c`.
2. Vérification :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
printf '%s\n' 'a1b2c3' | tr -d '0-9'
printf '%s\n' 'a  b  c' | tr -s ' '
```

```
abc
a b c
```

3. Résultat : deux lignes, `abc` et `a b c`.

11 Exercice 2 — `sed` : remplacer partout

11.1 Consigne

Remplace toutes les occurrences de `anomalie` par `erreur` dans la ligne `anomalie détectée, anomalie confirmée`, en une seule transformation.

11.2 Correction pas à pas

1. Attendu : la ligne entièrement remplacée, les deux occurrences traitées.
2. Vérification :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
printf '%s\n' 'anomalie détectée, anomalie confirmée' | sed 's/anomalie/erreur/g'
```

```
erreur détectée, erreur confirmée
```

3. Résultat : une ligne, avec deux `erreur`. Sans `g`, une seule des deux occurrences est remplacée.

12 Exercice 3 — chaîne de nettoyage

12.1 Consigne

1. Sur `contacts.csv` : extraire la colonne des identifiants (le nom est le 1 champ), la convertir en majuscules, et ajouter le préfixe `Fiche03`.
2. Reconstruire la même chaîne à partir de `contacts.csv`, une étape à la fois, et dire ce que chaque étape ajoute.

12.2 Correction pas à pas

1. Attendu : cinq lignes en majuscules, chacune commençant par `Fiche03`.
2. Les trois passages :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
echo '-- etape 1 : cut -d, -f1'
cut -d, -f1 contacts.csv
echo '-- etape 2 : tr a-z → A-Z'
cut -d, -f1 contacts.csv | tr 'a-z' 'A-Z'
echo '-- etape 3 : sed prefixe'
cut -d, -f1 contacts.csv | tr 'a-z' 'A-Z' | sed 's/^/Fiche03 /'
```

```
-- etape 1 : cut -d, -f1
Bruno
Chen
Diallo
Dupont
Nadia
-- etape 2 : tr a-z → A-Z
BRUNO
CHEN
DIALLO
DUPONT
NADIA
-- etape 3 : sed prefixe
Fiche03 BRUNO
Fiche03 CHEN
Fiche03 DIALLO
Fiche03 DUPONT
Fiche03 NADIA
```

3. Résultat : la sortie de l'étape 1 sert d'entrée à l'étape 2, celle de l'étape 2 sert d'entrée à l'étape 3 (l'outillage des tubes est détaillé en fiche 04).

13 Exercice 4 — `cut -c` et la position des colonnes

13.1 Consigne

1. À partir d'une ligne de forme `DT:12`, extraire uniquement le premier caractère du premier champ.
2. Expliquer pourquoi `cut -c` est fragile ici.

13.2 Correction pas à pas

1. Attendu : la lettre `D`.
2. Vérification :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
printf '%s\n' 'DT:12' | cut -c1
printf '%s\n' 'DT:12' | cut -c2
```

```
D
T
```

3. Résultat : `cut -c` compte les caractères, pas les champs. `cut -c1` donne le premier caractère (`D`) ; `cut -c2` donne le deuxième (`T`). Si une ligne change de longueur ou de préfixe, la même commande extrait autre chose : préférence à `-d`/`-f` dès qu'il y a un séparateur.

14 Exercice 5 — lire une ligne de commande

14.1 Consigne

1. À partir de `notes.passwd`, extraire ce qu'est chaque compte, c'est-à-dire son identifiant de connexion, avec une seule commande.
2. Extraire le programme lancé à la connexion (le dernier champ), puis demander ce même champ et l'identifiant de connexion ensemble avec `-f7,1` : la consigne dit « dans l'ordre inverse », et l'on va constater ce que `cut` répond vraiment.
3. Trouver l'identifiant numérique associé au compte `etudiante`.
4. Obtenir la liste des shells utilisés, sans les trier, puis retrouver l'ordre complet des comptes trié par identifiant numérique croissant.

14.2 Correction pas à pas

1. Attendu : trois lignes, `bruno`, `etudiante`, `prof`.
2. Extraire le login :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
cut -d: -f1 notes.passwd
```

```
bruno
etudiante
prof
```

3. Extraire le champ 7 (le shell) :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
cut -d: -f7 notes.passwd
```

```
/bin/zsh
/bin/bash
/bin/zsh
```

4. Piège : `cut` rend toujours les champs **dans l'ordre du fichier**, jamais dans l'ordre écrit après `-f`. `-f7,1` renvoie donc `bruno:/bin/zsh`, comme `-f1,7` : inverser la liste n'inverse pas la sortie. `rev` ne conviendrait pas non plus : il inverse les **caractères** de la ligne entière, pas l'ordre des champs. Réordonner deux champs demande un autre outil — `awk`, explicitement hors programme dans cette unité. Avec `cut` seul, le seul repli honnête est de lire les champs en deux passes, `-f7` puis `-f1`. Vérifions `cut` :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
cut -d: -f7,1 notes.passwd
```

```
bruno:/bin/zsh
etudiante:/bin/bash
prof:/bin/zsh
```

5. Retrouver l'identifiant numérique de `etudiante` avec `cut -d: -f1,3`, qui affiche le login et l'identifiant côte à côte :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
cut -d: -f1,3 notes.passwd
```

```
bruno:1000
etudiante:1001
prof:100
```

6. L'identifiant numérique de `etudiante`, en partant de sa ligne complète : on garde d'abord la ligne qui commence par ce nom, puis on découpe son 3 champ.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
grep '^etudiante:' notes.passwd | cut -d: -f3
```

```
1001
```

7. Lister les shells utilisés. `sort` range les lignes, `uniq` supprime les doublons adjacents (deux commandes détaillées en **fiche 04**, employées ici sans réexplication) :

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
echo '-- sans tri : les lignes dans l ordre du fichier'
cut -d: -f7 notes.passwd
echo '-- avec sort | uniq'
cut -d: -f7 notes.passwd | sort | uniq
```

```
-- sans tri : les lignes dans l ordre du fichier
/bin/zsh
/bin/bash
/bin/zsh
-- avec sort | uniq
/bin/bash
/bin/zsh
```

8. Trier les comptes par identifiant numérique croissant, avec le séparateur `-t:` et la clé `-k3` (le 3 champ). `sort` accepte donc un séparateur de champ explicite : l'option n'est pas reprise dans le tableau de la fiche 04, retenez-la ici.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
sort -t: -k3 -n notes.passwd
```

```
prof:x:100:100:prof:/home/prof:/bin/zsh
bruno:x:1000:1000:bruno:/home/bruno:/bin/zsh
etudiante:x:1001:1001:etudiante:/home/etudiante:/bin/bash
```

9. Le nom du programme lancé à la connexion est le **dernier** champ : on l'a extrait ci-dessus avec `-f7`, parce que chaque ligne de `notes.passwd` compte exactement 7 champs. Quand le nombre de champs change, `-f7` tombe faux : il faut soit compter les champs, soit changer d'outil.

[!WARNING] `cut -c` découpe par colonnes et n'est pas une méthode fiable pour lire le contenu d'un fichier avec des séparateurs.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
echo 'coupe par champs'
printf '%s\n' 'bruno:x:1000' | cut -d: -f3
echo 'coupe par colonnes'
printf '%s\n' 'bruno:x:1000' | cut -c3
echo 'convertit les caractères'
printf '%s\n' 'bruno:x:1000' | tr ':' '#'
```

```
coupe par champs
1000
coupe par colonnes
u
convertit les caractères
bruno#x#1000
```

15 Exercice 6 — lire une valeur saisie

15.1 Consigne

Écris un petit script qui demande un identifiant, lit la réponse au clavier, et affiche un message différent si la valeur est vide ou absente.

> [!NOTE] Tu n'as pas de terminal interactif dans ce site publié : une commande comme `read` attend une frappe et ne se déroule pas ici. Écris le script dans un fichier (par exemple `scripts/capture.sh`), teste-le chez toi dans un vrai terminal, et joins-le à ton rendu.

15.2 Correction pas à pas

1. Attendu : une ligne selon l'entrée (`etudiant` donne un message d'accueil, une entrée vide ou absente donne un message d'erreur).
2. Le corrigé ci-dessous lit une valeur au clavier avec `read`. `[[ -z "$v" ]]` vaut « vrai » quand `$v` est vide. Nous, nous l'avons testé en ligne de commande en lui injectant une valeur (`<<<`), sans ouvrir de vrai terminal interactif :

```
read -r identifiant
if [[ -z "$identifiant" ]]; then
  echo "Erreur : identifiant vide."
else
  echo "Bienvenue $identifiant"
fi
```

3. Vérification : on rejoue ce même script dans les trois cas (valeur injectée, ligne vide, aucune entrée du tout avec `< /dev/null`) :

Le motif `cat > fichier <<'MARQUEUR'` employé ci-dessous est un **ici-document** : tout ce qui suit, jusqu'à une ligne `MARQUEUR` écrite seule, part dans l'entrée standard de `cat` qui l'écrit dans le fichier. C'est la **fiche 05** qui l'explique en détail ; ici nous l'employons sans réexplication.

```
cd "${WORK:?}/fiche03-sandbox" || { echo "ERREUR : dossier absent ${WORK}/
fiche03-sandbox — exécuter d'abord la cellule « Nettoyer et transformer avec tr
et sed »"; exit 1; }
mkdir -p scripts
cat > scripts/capture.sh <<'CAPTURE'
read -r identifiant
if [[ -z "$identifiant" ]]; then
  echo "Erreur : identifiant vide."
else
  echo "Bienvenue $identifiant"
fi
CAPTURE
echo '-- avec une valeur'
bash scripts/capture.sh <<< 'etudiant'
echo '-- avec une ligne vide'
bash scripts/capture.sh <<< ''
echo '-- sans aucune entrée'
bash scripts/capture.sh < /dev/null
```

```
-- avec une valeur
Bienvenue etudiant
-- avec une ligne vide
Erreur : identifiant vide.
-- sans aucune entrée
Erreur : identifiant vide.
```

4. Résultat : les deux derniers cas affichent `Erreur : identifiant vide.` et le script s'arrête proprement.

16 Récapitulatif des commandes de la fiche

Besoin	Commande
Extraire une colonne par séparateur	`cut -d: -f1`
Extraire une colonne par position	`cut -c1`

Besoin	Commande
Supprimer des caractères	`tr -d '0-9'`
Compresser des répétitions	`tr -s ' '`
Changer la casse	`tr 'a-z' 'A-Z'`
Remplacer un motif	`sed 's/anomalie/erreur/g'`
Préfixer une ligne	`sed 's/^/Fiche03 /'`
Lister sans doublons	`sort | uniq`

17 Auto-vérification

17.1 Consigne

Compare chaque sortie obtenue à la sortie attendue ci-dessous. Coche au fur et à mesure. Les sorties attendues viennent des fichiers `notes.txt`, `notes.passwd` et `contacts.csv` fabriqués par la cellule de démarrage.

17.2 Correction pas à pas

Chaque test compare, avec `[[ ... == ... ]]`, la sortie réellement produite par la commande à la sortie attendue écrite mot pour mot. `$(cmd)` remplace une commande par ce qu'elle affiche. Une ligne est affichée par test raté, plus une ligne de comptage.

```
cd "${WORK:?}" || { echo "ERREUR : dossier absent ${WORK}"; exit 1; }
ok=0
total=7

attendu='Bruno
Chen
Diallo
Dupont
Nadia'
obtenu=$(cut -d: -f1 notes.txt)
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO cut -d: -
f1 notes.txt"

attendu='B
C
D
D
N'
obtenu=$(cut -c1 notes.txt)
```

```
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO cut -c1 notes.txt"

attendu='1000
1001
100'
obtenu=$(cut -d: -f3 notes.passwd)
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO cut -d: -
f3 notes.passwd"

attendu='bruno:/bin/zsh
etudiante:/bin/bash
prof:/bin/zsh'
obtenu=$(cut -d: -f7,1 notes.passwd)
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO cut -d: -
f7,1 notes.passwd"

attendu='abc'
obtenu=$(tr -d '0-9' <<< 'a1b2c3')
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO tr -d '0-9'"

attendu='a b c'
obtenu=$(tr -s ' ' <<< 'a   b   c')
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO tr -s ' '"

attendu='erreur détectée, erreur confirmée'
obtenu=$(sed 's/anomalie/erreur/g' <<< 'anomalie détectée, anomalie confirmée')
[[ "${obtenu}" == "${attendu}" ]] && ok=$((ok + 1)) || echo "KO sed s/anomalie/
erreur/g"

echo "Auto-vérification fiche03 : ${ok}/${total} tests OK"
```

```
Auto-vérification fiche03 : 7/7 tests OK
```

Les sept sorties attendues, une par test :

- `cut -d: -f1 notes.txt` : `Bruno`, `Chen`, `Diallo`, `Dupont`, `Nadia` ;
- `cut -c1 notes.txt` : `B`, `C`, `D`, `D`, `N` ;
- `cut -d: -f3 notes.passwd` : `1000`, `1001`, `100` ;
- `cut -d: -f7,1 notes.passwd` : `bruno:/bin/zsh`, `etudiante:/bin/bash`, `prof:/bin/zsh` (dans l'ordre du fichier, pas dans l'ordre demandé) ;
- `tr -d '0-9'` sur `a1b2c3` : `abc` ;
- `tr -s ' '` sur `a   b   c` : `a b c` ;

- `sed 's/anomalie/erreur/g'` sur `anomalie détectée, anomalie confirmée` : `erreur détectée, erreur confirmée` (les deux occurrences, grâce à `g`).