Programmation C

Fiche technique nº 1 : aide-mémoire shell

Ce document reprend l'aide mémoire du stage « Environnement de travail (niveau 2) », accessible sous moodle.

---

1 Pourquoi apprendre le shell ?

Jusqu'ici, vous avez surtout utilisé un système par interface graphique : clics, glisser-déposer, fenêtres. Cela fonctionne, mais a un coût caché. En effet, chaque action répétitive demande d'utiliser la souris (ce qui est lent) et d'effectuer des allers-retours entre fenêtres (ce qui est également lent). De plus, il est difficile d'enregistrer ces actions et de les combiner pour les automatiser : une interface graphique n'est généralement pas programmable de cette manière. Le shell permet de répondre à ces inconvénients. Son apprentissage demande un petit investissement de départ, pour un gain de temps énorme ensuite.

Prenons un exemple concret : vous voulez renommer 200 fichiers `photo (1).jpg`, `photo (2).jpg`, … en `img_1.jpg`, `img_2.jpg`, etc. À la souris, cela nécessite de répéter 200 fois une opération de renommage, ce qui peut représenter un travail long et pénible, avec un risque d'erreur non négligeable. En shell, une commande ou un petit script peut traiter l'intégralité du dossier en une seconde, de façon fiable et reproductible. Ceci n'est qu'un exemple parmi d'autres : c'est la même chose pour chercher tous les fichiers `.c` contenant un motif particulier, supprimer tous vos fichiers temporaires de plus d'une semaine, synchroniser un répertoire sur une machine distante, télécharger une vidéo YouTube, ou relancer une même série de manipulations chaque soir. À la souris, ces tâches sont longues et fastidieuses. Avec le shell, elles peuvent souvent être exécutées en quelques secondes.

On peut comparer l'apprentissage du shell à celui du vélo. Les premières séances peuvent être frustrantes : on doit apprendre à garder l'équilibre, et l'on peut même avoir l'impression de perdre du temps par rapport à la marche. Mais une fois l'équilibre acquis, on va bien plus vite et bien plus loin, pour un effort moindre. La situation est analogue avec le shell : les premières commandes demandent un effort de mémorisation, mais elles deviennent rapidement des réflexes qui vous feront gagner un temps considérable. Cela vous sera utile pendant tout votre cursus en informatique, en particulier lors des épreuves sur machines. Parfois d'ailleurs, le shell est indispensable, car certaines fonctionnalités n'ont aucune interface graphique.

Un dernier argument pour vous convaincre d'apprendre le shell : vous pouvez au départ vous contenter des quelques commandes les plus utiles, et en ajouter au fur et à mesure. Autrement dit, il n'est pas obligatoire de tout apprendre d'un coup pour commencer à gagner du temps. Bon shell !

2 Aide-mémoire et récapitulatif pour une prise en main du shell

Ce document récapitule les commandes les plus utiles pour démarrer avec le shell. Il donne des exemples pour chacune d'elles. Une fois assimilé, vous pourrez accomplir l'essentiel de votre travail quotidien en ligne de commande, c'est-à-dire sous le shell.

Conseil nº 1 : Ne cherchez pas à tout mémoriser. Apprenez quelques commandes, puis utilisez la commande `man`, l'option `--help` ou `--aide` de la plupart des commandes, l'historique et la complétion automatique pour retrouver le reste.

Conseil nº 2 : Le plus important est de **pratiquer**. Dès qu'une manipulation vous semble répétitive, demandez-vous si le shell pourrait la faire à votre place.

Rappel sur la forme des commandes. Avant de présenter les commandes elles-mêmes, on rappelle leur syntaxe générale (voir la section sur les commandes de base du stage environnement de travail). Une commande est composée d'un **nom de commande**, éventuellement suivi d'**arguments**, le tout séparé par des espaces :

```
$ commande argument1 argument2...
```

Parmi ces arguments, certains sont des **options** : elles modifient le comportement de la commande plutôt que de désigner une donnée sur laquelle agir. Les options ont deux styles, *court* et *long*. Une option courte s'écrit avec un seul tiret suivi d'une lettre (`-x`), parfois suivie d'une valeur (`-x valeur`) ; une option longue s'écrit avec deux tirets suivis d'un mot (`--optionx`), éventuellement avec une valeur après un signe égal (`--optionx=valeur`). Prenons comme exemple la commande `grep` qui permet d'afficher les lignes d'un fichier contenant un motif. Dans la ligne de commande,

```
$ grep -n "mon motif" fichier.py
```

`grep` est le nom de la commande, `-n` est une option (elle demande d'afficher les numéros des lignes trouvées), tandis que `"mon motif"` et `fichier.py` sont deux arguments ordinaires : le motif recherché et le fichier concerné. Au lieu de `-n`, on peut utiliser la version longue `--line-number`. C'est une question de goût. L'une est plus rapide à taper, l'autre peut-être plus parlante et plus facile à mémoriser :

```
$ grep --line-number "mon motif" fichier.py
```

Sans les guillemets autour de `mon motif`, le shell aurait compris qu'on lui donne 4 arguments : `-n`, `mon`, `motif` et `fichier.py`, puisque les espaces séparent les arguments. Les guillemets permettent de regrouper plusieurs mots en un seul argument malgré les espaces qu'ils contiennent : `"mon motif"` compte donc comme un unique argument, le motif recherché par `grep`. Enfin, on rappelle que les options courtes se combinent : au lieu d'écrire `ls -l -a`, on peut simplement écrire `ls -la`.

Rappel sur les noms de fichiers. Dans cette fiche, « **dossier** » et « **répertoire** » désignent exactement la même chose. Les noms de fichiers ou de répertoires peuvent être indiqués sous forme **absolue**, en partant de la racine du système (`/net/cremi/mon_login/projet/main.c`), ou sous forme **relative**, en partant du répertoire courant (`../projet/main.c` pour remonter d'un niveau et descendre dans `projet/`). On peut aussi utiliser le raccourci `~` pour désigner le répertoire personnel.

2.1 Navigation et manipulation de fichiers

Commande	Exemple	Effet
pwd	`pwd`	Affiche le chemin absolu du dossier dans lequel on se trouve.
cd	`cd Documents`	Descend dans le sous-dossier `Documents`.
cd	`cd ..`	Remonte au dossier parent.
cd	`cd ../Documents`	Entre dans le dossier `Documents` du dossier parent.
cd	`cd`	Retourne au dossier personnel.
cd	`cd -`	Retourne au dernier dossier visité.
ls	`ls`	Affiche les noms des fichiers et dossiers du répertoire courant (sauf ceux dont les noms commencent par `.`).
ls	`ls -l`	Idem, avec détails (permissions, taille, date, propriétaire).
ls	`ls -la`	Idem, en incluant les fichiers cachés (commençant par `.`).
ls	`ls -lh`	Affiche les tailles dans un format lisible (Ko, Mo…).
ls	`ls -lt`	Trie les fichiers du plus récent au plus ancien.
ls	`ls -lS`	Trie les fichiers du plus grand au plus petit.
ls	`ls -laht`	Combine les options `l`, `a`, `h` et `t` précédentes.
mkdir	`mkdir projet`	Crée le dossier `projet`.
mkdir	`mkdir -p a/b/c`	Crée toute l'arborescence `a/b/c`.
rmdir	`rmdir tmp`	Supprime le dossier `tmp` s'il est vide.

Commande	Exemple	Effet
touch	`touch notes.txt`	Crée un fichier vide `notes.txt` s'il n'existe pas, ou actualise sa date de modification s'il existe.
cp	`cp a.txt b.txt`	Crée une copie de `a.txt` nommée `b.txt`. ⚠ Si `b.txt` existait déjà, il est écrasé.
cp	`cp a.txt dossier/`	Copie `a.txt` dans `dossier/` en conservant son nom. ⚠ Si `dossier/a.txt` existait déjà, il est écrasé.
cp	`cp -r src/ dst/`	Copie récursivement tout le contenu de `src/` vers `dst/`. Crée `dst` s'il n'existe pas. ⚠ Écrase les fichiers de `dst` de même nom que ceux de `src`.
cp	`cp -i fichier.txt dossier/`	Copie en demandant confirmation avant d'écraser un fichier existant.
mv	`mv old.txt new.txt`	Renomme `old.txt` en `new.txt`. ⚠ Si `new.txt` existait déjà, il est écrasé.
mv	`mv fichier.txt dossier/`	Déplace `fichier.txt` dans `dossier/`. ⚠ Si `dossier/fichier.txt` existait déjà, il est écrasé.
mv	`mv *.txt dir/`	Déplace tous les fichiers `.txt` du dossier courant vers `dir/`. ⚠ Écrase les fichiers de `dir` de même nom qu'un des fichiers déplacés.
rm	`rm fichier.txt`	Supprime définitivement `fichier.txt`. ⚠ Action irréversible ! .
rm	`rm -i *.txt`	Supprime chaque fichier `.txt`, après confirmation individuelle.
rm	`rm -r dossier/`	Supprime `dossier/` et tout son contenu, récursivement. ⚠ Action irréversible !.
find	`find . -name '*.py'`	Cherche tous les fichiers et dossiers dont le nom se termine par `.py` à partir du dossier courant.
find	`find . -type f -name '*.c'`	Idem pour les fichiers `.c`, en ne gardant que les fichiers ordinaires (pas les dossiers).
find	`find . -type d -name "build"`	Cherche les dossiers nommés `build`.
ln	`ln -s /chemin/cible mon_lien`	Crée un lien symbolique `mon_lien` pointant vers le fichier ou répertoire `/chemin/cible`.
rename.ul	`rename.ul .jpeg .jpg *.jpeg`	Renomme toutes les extensions `.jpeg` en `.jpg` dans le dossier courant.
mmv	`mmv "photo (*).jpg" "img_#1.jpg"`	Renomme `photo (1).jpg`, `photo (2).jpg`… en `img_1.jpg`, `img_2.jpg`…

⚠ Attention à `cp`, `mv` et surtout `rm`

Les fichiers supprimés avec `rm` ne sont normalement pas placés dans la corbeille. De même, les commandes `cp` et `mv` écrasent leur destination, par défaut. Utiliser l'option `-i` de ces trois commandes fait qu'elles demanderont confirmation à l'utilisateur en cas de suppression ou d'écrasement d'un fichier. Ajoutez les lignes suivantes à la fin du fichier `~/.bashrc` de configuration du shell pour que ces commandes vous demandent toujours confirmation en cas d'écrasement ou de suppression d'un fichier :

```
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'
```

Exemple : renommage de fichiers en masse

Pour l'exemple des 200 photos du niveau 2 du stage « Environnement de travail », plusieurs utilitaires permettent de le faire, comme `rename` et `mmv` (inclus dans le tableau ci-dessus). Il faut les installer au préalable sur votre système. Au CREMI, `rename.ul` est installé.

2.2 Historique et aide

Commande	Exemple	Effet
man	`man grep`	Ouvre le manuel détaillé de la commande `grep`.
man	`man man`	Ouvre le manuel détaillé de la commande `man`.
man	`man printf`	Ouvre le manuel détaillé de la commande `printf` du shell.
man	`man 3 printf`	Ouvre le manuel détaillé de la fonction `printf` de la bibliothèque C (les fonctions de la bibliothèque standard C sont en section 3).
info	`info gcc`	Ouvre le manuel complet et hypertexte du compilateur `gcc`.
info	`info info`	Ouvre le manuel complet et hypertexte du système `info`.
commande --help	`grep --help`	Affiche un résumé des options disponibles pour `grep`.
history	`history`	Affiche la liste numérotée des commandes précédemment tapées.
↑ / ↓	—	Rappelle la commande précédente ou suivante dans l'historique.
ctrl + R	—	Ouvre une recherche incrémentale dans l'historique Taper ctrl + R permet de remonter dans cet historique.
→|	—	Complète le nom en cours de saisie, ou propose les possibilités.
clear ou ctrl + L	`clear`	Efface l'affichage à l'écran (sans toucher à l'historique des commandes).

À retenir

Vous n'avez pas besoin de mémoriser toutes les commandes. Les touches →|, ctrl + R, les commandes `man`, `info` et les options `--help` ou `--aide` permettent de retrouver rapidement ce que vous avez oublié.

2.3 Motifs de fichiers (*globbing*)

Le shell peut remplacer certains motifs par les noms de fichiers qui leur correspondent.

Commande	Exemple	Effet
*****	`ls *.c`	Liste tous les fichiers dont le nom se termine par `.c`.
*****	`cp *.c sauvegarde/`	Copie dans `sauvegarde` tous les fichiers dont le nom se termine par `.c`.
?	`ls ?.txt`	Liste les fichiers dont le nom fait exactement un caractère suivi de `.txt`.
[abc]	`ls [abc].txt`	Liste `a.txt`, `b.txt` ou `c.txt`, s'ils existent.
[0-9]	`ls img_[0-9].jpg`	Liste `img_0.jpg` à `img_9.jpg` (ceux qui existent).

Attention

Les symboles `*`, `?`, `[` et `]` sont interprétés par le shell lui-même. Ce n'est pas une fonctionnalité de `ls` ou de `cp`. Par conséquent, ce mécanisme fonctionne avec toute commande. Si on ne veut pas que les caractères `*`, `?`, `[` et `]` soit interprétés de cette façon spéciale par le shell, on les fait précéder du caractère `\` (comme dans `\*`) ou on les met entre guillemets (comme dans `"*"` ou `'*'`).

2.4 Afficher et rechercher du contenu dans des fichiers

Commande	Exemple	Effet
cat	`cat main.c`	Affiche le contenu de `main.c` dans le terminal.
cat	`cat a.txt b.txt > c.txt`	Concatène `a.txt` et `b.txt` dans un nouveau fichier `c.txt`.
less	`less rapport.log`	Ouvre `rapport.log` en mode consultation, navigable avec les flèches, l'espace et `b`. Sous `less`, tapez `h` pour une aide détaillée.
head	`head fichier.txt`	Affiche les 10 premières lignes de `fichier.txt`.
head	`head -n 20 data.csv`	Affiche les 20 premières lignes de `data.csv`.

Commande	Exemple	Effet
tail	`tail -n 5 fichier.txt`	Affiche les 5 dernières lignes de `fichier.txt`.
tail	`tail -f server.log`	Affiche les dernières lignes de `server.log`, puis suit le fichier en temps réel au cas où il grandirait.
grep	`grep erreur log.txt`	Affiche les lignes de `log.txt` contenant le motif « erreur ».
grep	`grep -i erreur log.txt`	Idem, sans tenir compte des majuscules/minuscules.
grep	`grep -iw erreur log.txt`	Idem, mais le motif `erreur` doit former un mot. Ainsi, les motifs `terreur` et `erreurs` ne compteront pas.
grep	`grep -rn TODO src/`	Cherche le motif `TODO` récursivement dans tous les fichiers de `src/`, en affichant les numéros des lignes trouvées.
wc	`wc -l fichier.txt`	Affiche le nombre de lignes de `fichier.txt`.
sort	`sort fichier.txt`	Affiche les lignes de `fichier.txt` triées par ordre alphabétique.
sort	`sort -n nombres.txt`	Trie les lignes en interprétant leur contenu comme des nombres. Ainsi, `09` est considéré comme plus grand que `7`, alors qu'en ordre alphabétique, `09` est plus petit que `7` (car `0` vient avant `7`).
diff	`diff v1.c v2.c`	Affiche les différences ligne à ligne entre `v1.c` et `v2.c`.

2.5 Rediriger et enchaîner les commandes

Un flux est un endroit où un programme peut lire ou écrire. Les commandes utilisent en général trois flux :

- l'**entrée standard** (notée `stdin` en C), habituellement associée au clavier. C'est le flux où elles lisent.
- la **sortie standard** (notée `stdout` en C), habituellement associée à l'écran. C'est le flux où elles écrivent.
- la **sortie d'erreur** (notée `stderr` en C), habituellement associée à l'écran. C'est le flux où elles écrivent leurs messages d'erreur.

On peut se demander l'intérêt d'avoir un flux où écrire les messages « normaux », et un autre où écrire les messages d'erreur, d'autant que ces deux flux sont associés à l'écran par défaut. Autrement dit, par défaut, les messages normaux et les messages d'erreur d'une commande se mélangent à l'écran. L'intérêt est pourtant réel : il vient du fait que le shell est capable, comme on va le voir dans cette section, de rediriger les flux, c'est-à-dire de changer les associations par défaut (clavier pour la lecture et écran pour l'écriture). Le shell peut par exemple faire en sorte que le flux de sortie soit associé à un fichier `resultats.txt` au lieu de l'écran, alors que la sortie d'erreur reste inchangée. La commande pour laquelle le shell fait cette modification aura toutes ses écritures normales (par `printf(...);` pour un programme C) redirigées vers le fichier `resultats.txt`. Par contre, les écritures réalisées par `fprintf(stderr, ...);` resteront affichées à l'écran.

Commande	Exemple	Effet
>	`ls > liste.txt`	Redirige le résultat de `ls` dans `liste.txt`. Crée `liste.txt` s'il n'existe pas, écrase son contenu précédent s'il existe.
>>	`echo "log" >> journal.txt`	Ajoute la ligne « log » à la fin de `journal.txt`, sans effacer le reste.
<	`sort < data.txt`	Trie le contenu de `data.txt`, utilisé comme entrée de `sort`.
2>	`prog 2> erreurs.log`	Enregistre les messages d'erreur de `prog` dans `erreurs.log`.
|	`ls -l | grep ".c"`	Liste les fichiers, puis ne garde que les lignes contenant `.c`. Ce qu'écrit `ls` (sa sortie) est redirigé vers ce que lit `grep` (son entrée).

⚠ Attention à la redirection >

La redirection `>` **écrase un fichier existant**. Par exemple, si le fichier `fic.txt` existe, la commande `echo Hello > fic.txt` écrasera le fichier existant. Vous pouvez faire que le shell refuse ce type d'écrasement, en ajoutant dans le fichier `~/.bashrc` de configuration du shell la ligne `set -o noclobber`. Dans ce cas, la commande `echo Hello > fic.txt` échouera si `fic.txt` existe déjà. Dans cette même situation, si on veut vraiment écraser `fic.txt`, il faudra utiliser `echo Hello >| fic.txt`.

Le symbole `|` (pipe) est particulièrement puissant car il permet de **chaîner des commandes simples** pour construire des traitements plus complexes.

2.6 Processus

Commande	Exemple	Effet
ps	`ps aux`	Affiche tous les processus du système avec leurs détails.
top	`top -o' %MEM'`	Ouvre un moniteur interactif classant les processus par consommation mémoire.
htop	`htop`	Ouvre un moniteur interactif, plus convivial que `top`, classant les processus par consommation CPU. On peut ensuite changer ce critère de classement.
&	`long_calcul &`	Exécute `long_calcul` sans bloquer le terminal.
ctrl + Z	—	Met le programme en pause, récupérable ensuite avec `fg` ou `bg`.
jobs	`jobs`	Affiche les tâches en arrière-plan ou suspendues de ce shell.
fg	`fg %1`	Replace la tâche numéro 1 au premier plan.
bg	`bg %1`	Relance en arrière-plan la tâche numéro 1, précédemment suspendue.
kill	`kill 1234`	Demande au processus 1234 de se terminer proprement.
kill -9	`kill -9 1234`	Force l'arrêt immédiat du processus 1234, sans possibilité de nettoyage.
ctrl + C	—	Arrête immédiatement le programme en cours d'exécution.

Conseil

Préférez `kill PID` à `kill -9 PID`. Le premier demande normalement au programme de se terminer proprement, alors que `kill -9` doit plutôt être réservé aux cas où cela ne fonctionne pas.

Remarque

Le fait que ctrl + C soit utilisé pour envoyer un signal explique pourquoi il n'est pas utilisé sous le shell pour copier, alors que c'est le raccourci standard pour copier dans la plupart des applications. Pour copier sous le shell, on utilise ctrl + ⇧ + C (et ctrl + ⇧ + V pour coller). Certains programmes sont « immunisés » contre le ctrl + C, en premier lieu le shell lui-même (pour éviter de le terminer par erreur). Pour le terminer, vous pouvez taper `exit` ou ctrl + D, qui simule une fin de fichier au terminal.

2.7 Permissions

Commande	Exemple	Effet
chmod	`chmod +x script.sh`	Rend `script.sh` exécutable.
chmod	`chmod 0644 fic.txt`	Donne lecture/écriture au propriétaire, lecture seule au groupe et aux autres.
chown	`sudo chown user fic.txt`	Affecte l'utilisateur `user` comme propriétaire du fichier `fic.txt`.
sudo	`sudo commande`	Exécute `commande` avec les droits administrateur.

Les permissions sont notées `r` (lecture), `w` (écriture) et `x` (exécution). Elles sont indiquées lorsqu'on lance `ls -l`, séparément pour le propriétaire, le groupe et les autres utilisateurs. Dans l'exemple suivant, le propriétaire (`joe`) a les droits en lecture, écriture exécution (`rwx`), le groupe (`staff`) a les droits en lecture et en exécution mais pas en écriture (`r-x`), et les autres n'ont aucun droit (`---`).

```
$ ls -lh a.out
-rwxr-x--- 9,4k joe staff 2026-09-01 17:04 a.out
```

Le premier symbole de la ligne (ici `-`) indique la nature du fichier (ici, un fichier ordinaire). On aurait eu un `d` pour un dossier. Par ailleurs, sur un dossier, les permissions sont interprétées de la façon suivante : la permission `r` désigne celle de pouvoir lister le dossier (par exemple par `ls`). La permission `x` désigne celle de pouvoir y ajouter ou supprimer d'autres fichiers ou dossiers. Enfin, la permission `x` désigne celle de pouvoir traverser le dossier (par `cd`).

2.8 Réseau

Commande	Exemple	Effet
ping	`ping ombrage.emi.u-bordeaux.fr`	Vérifie que la machine `ombrage.emi.u-bordeaux.fr` répond sur le réseau.
ssh	`ssh monlogin@ombrage.emi.u-bordeaux.fr`	Ouvre une session sur une machine distante — Ouvre une session distante sur la machine `ombrage` du CREMI avec le compte `monlogin`.
ssh	`ssh monlogin@ombrage.emi.u-bordeaux.fr ps`	Lance une commande sur une machine distante — Lance la commande `ps` sur la machine `ombrage` du CREMI avec le compte `monlogin`.
scp	`scp fichier.txt monlogin@ombrage.emi.u-bordeaux.fr:./dossier/`	Envoie `fichier.txt` dans `./dossier/` sur la machine distante.
curl	`curl -O https://exemple.fr/data.csv`	Télécharge `data.csv` et l'enregistre sous son nom d'origine.
wget	`wget https://exemple.fr/data.csv`	Télécharge `data.csv` dans le dossier courant.

2.9 Archives et compression

Commande	Exemple	Effet
tar	`tar cfz archive.tar.gz dir/`	Crée l'archive compressée `archive.tar.gz` à partir de `dir/`.
tar	`tar xfz archive.tar.gz`	Extrait le contenu de l'archive dans le dossier courant.
zip	`zip -r archive.zip dir/`	Crée `archive.zip` en compressant récursivement `dir/`.
unzip	`unzip archive.zip`	Extrait le contenu de `archive.zip` dans le dossier courant.
gzip	`gzip fichier.txt`	Remplace `fichier.txt` par `fichier.txt.gz` compressé.
gunzip	`gunzip fichier.txt.gz`	Restaure `fichier.txt` à partir de l'archive `.gz`.
bzip2	`bzip2 fichier.txt`	Remplace `fichier.txt` par `fichier.txt.bz2` compressé.
bunzip2	`bunzip2 fichier.txt.bz2`	Restaure `fichier.txt` à partir de l'archive `.bz2`.