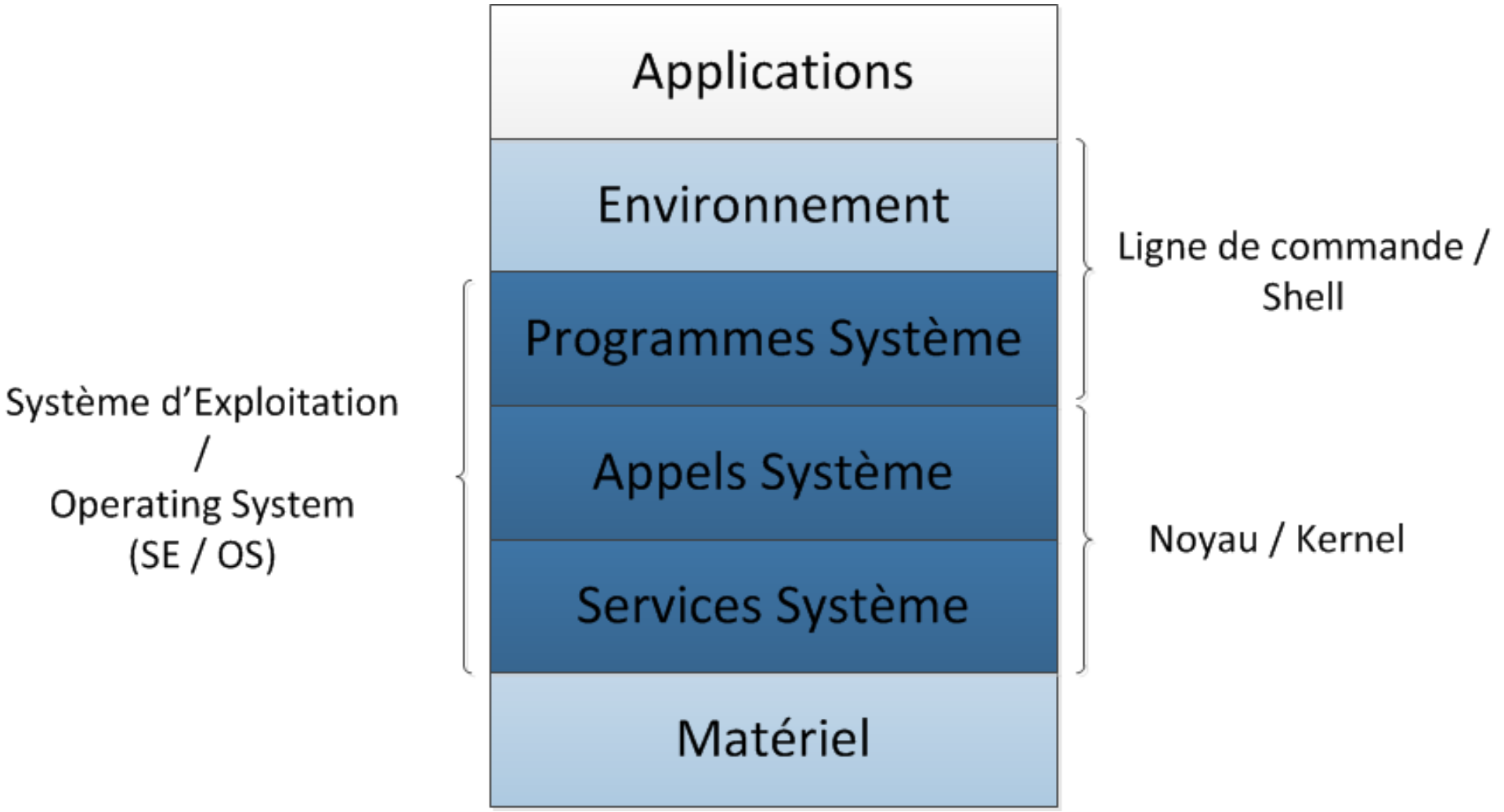
Architecture des Ordinateurs et Systèmes d'Exploitation

Partie 4 : Commandes
Cours (suite) & TD

Fabrice BOISSIER & Elena KUSHNAREVA
2017/2018

fabrice.boissier@gmail.com
elena.kushnareva@malix.univ-paris1.fr

Rappel OS


Applications
Environnement
Programmes Système
Appels Système
Services Système
Matériel
Système d'Exploitation / Operating System (SE / OS)
Ligne de commande / Shell
Noyau / Kernel

Rappel Shell

- Tout script commence par :
  `#! /bin/sh`

- if, then, elif, else, fi
- case, in, esac
- while, do, done
- until, do, done
- for, in, do, done

- mode maths

Rappel Shell

- Double quote : " (sur AZERTY :) (touche 3)
  - Contient du texte et des variables remplacées
    `"Coucou $nom !"` => `"Coucou BOISSIER !"`
- Simple quote : ' (touche 4)
  - Contient du texte et des variables « non » remplacées
    `'Coucou $nom !'` => `'Coucou $nom !'`
- Back quote : ` (AltGr + 7)
  - Contient une commande qui sera exécutée dans un sous shell
    `` `ls -a` `` => `" . .. file1"`

Rappel Shell

$0	Commande/Shell actuel
$-	Paramètres du shell actuel
$#	Nombre de paramètres positionnels
$*	Liste des paramètres en un seul mot
$@	Liste des paramètres en plusieurs mots
$?	Valeur de retour de la dernière commande
$!	PID du dernier processus en background
$$	PID du shell courant

Rappel Shell

- Multiples variables d'environnement existantes :

Prompt : PS1, PS2, PS3, PS4

PATH, HOME, USER, PWD, HOSTNAME, LANG, EDITOR, SHELL, SHLVL, …

Builtins Shell

- Redirections : < > << >> |
- test / [
- echo
- exit
- set / unset
- export (setenv)
- alias / unalias
- read
- readonly
- return
- eval
- let
- type
- =
- cd
- source
- jobs
- fg / bg
- fc
- break / continue
- kill
- command
- dirs / pushd / popd
- times
- wait

Liste rapide d'outils classiques

- Commandes/Outils de base :
  - man, sh, bash, tcsh, exit, echo
  - cd, ls, cp, mv, rm, mkdir, rmdir
  - vi, vim, emacs, nano, ed

- Outils :
  - find, date,
  - cat, cut, paste, tr, tee, sort, mktemp, basename, …
  - diff, head, tail, more, wc

- Outils avancés :
  - grep, expr, sed, awk, ed

Références Bibliographiques

- FreeBSD Handbook
- Linux Man Pages
- The Open Group
  Single UNIX Specification / SUSv4

Usage des Manuels

- Commande `man`
- Plusieurs sections :
  - 1 commande/programme
  - 2 appel système (syscall)
  - 3 fonction C (subroutine)
  - 4 fichiers spéciaux
  - 5 format de fichier
  - 7 macros et conventions
  - 8 commande de maintenance (super user)

Usage des Manuels

- Par défaut : man 1

`man passwd` = `man 1 passwd`

`man 5 passwd` = format du fichier passwd

`man printf` != `man 3 printf`

`man read` != `man 2 read`

Usage des Manuels

- Pour lire le manuel de la commande : `ls`

```
man ls
man 1 ls
```

- Pour lire le manuel de l'appel system : `read`

```
man 2 read
```

- Pour lire le manuel de la fonction C : `printf`

```
man 3 printf
```

Editeurs de Texte

- Modifier des fichiers texte « brut »
  - Modification en mode ASCII, UTF-8, …
  - Mais PAS dans l'optique de créer un document visuellement joli…
  - Objectif : créer un fichier de données

- Sur UNIX, plusieurs éditeurs de textes majeurs :
  - Emacs
  - Vi / Vim
  - Nano / Pico

Emacs

- Modification directe : on tape du texte, celui-ci est écrit dans le document… usage « classique » du clavier

- Sauvegarde : Ctrl + X Ctrl + S
- Quitter : Ctrl + X Ctrl + C
- Ouvrir un fichier : Ctrl + X Ctrl + F

- Nombreuses extensions en *scheme* (dialecte LISP)
- Editeur « relativement » lourd

```
emacs file1
emacs -nw file1
```

Emacs

- Outillage accessible par : Alt + … ou Ctrl + …
  - Meta + … ou Ctrl + …
  - Touche « Meta » souvent remplacée par « Alt » ou « ESC »
  - Commandes auto-complétée par « Tab »

Meta + X `delete-trailing-whitespace`

Emacs

- Si modification de fichiers existants, création de fichiers de sauvegarde…
  - Fichier précédent = nom + ~
  - Exemple :
    - `file1` existe
    - On modifie avec emacs le fichier file1
    - On sauvegarde et on quitte
    - `file1` est la version à jour + `file1~` a été créé
  - On salit très « très » vite ses répertoires…
    - Penser à un `rm *~` de temps en temps…

Vi / Vim

- Deux modes de fonctionnement :
  - Mode commande
  - Mode édition/insertion

- Vi disponible sur tous les UNIX quels qu'ils soient
- Editeur très léger, mais complet
  - Descendant/Compatible avec `ed`

```
vi file
```

Vi / Vim

- Mode commande :
  - Mode de démarrage
  - Déplacement avec flèches OU avec `H J K L`
  - Insertion de commandes
  - Passage en mode écriture avec : `a` ou `i` ou `A` ou `I`
    - a = append / ajout « après » le caractère courant
    - A = append sur ligne suivante
    - i = insertion / ajout « à partir » du caractère courant
    - I = insertion sur ligne suivante

Vi / Vim

- Mode commande :
  - Suppression d'un caractère : `x` ou `X`
    - x = supprime le caractère à droite
    - X = supprime le caractère à gauche
  - Suppression d'une ligne : `dd`
  - Supprimer tout jusqu'à la fin de la ligne : `D`

Vi / Vim

- Mode commande :
  - Ajout d'une ligne : `o` ou `O`
    - o = ajout d'une ligne « après » la ligne courante
    - O = ajout d'une ligne « avant » la ligne courante
  - Remplacer des caractères : `r` ou `R`
    - r = remplacer LE caractère suivant
    - R = remplacer plusieurs caractères suivant

Vi / Vim

- Mode commande : (! = forcer)
  - Sauvegarder :

    :w

    :w!
  - Quitter :

    :q

    :q!
  - Sauvegarder & Quitter :

    :wq

    :wq!

Vi / Vim

- Mode insertion :
  - On tape son texte…
  - Repasser en mode commande :
    - touche ESC

Bon courage !

Nano

- Modification directe : on tape du texte, celui-ci est écrit dans le document… usage « classique » du clavier

- Editeur très « très » léger
- Commandes avec Ctrl (^ = Ctrl)
  - Quitter : Ctrl + X (^X)
  - Sauvegarder : demandé lorsque l'on quitte

Commande script

- script

- Permet d'enregistrer tout ce qui s'affiche dans le terminal (commande, erreur, sortie standard, …)
- Par défaut, fichier `typescript` utilisé
- Se termine par appuie de *Ctrl + D* ou `exit`

```
script
ls
ls -1234
```

Déplacement Arborescence

- pwd (Print Working Directory)

- Affichage du nom du dossier courant
- Built-in du shell, qui affiche la variable d'environnement PWD

```
mkdir dossier1
pwd
cd dossier1
pwd
```

Déplacement Arborescence

- cd (Change Directory)

- Déplacement de dossier en dossier
- Built-in du shell, qui modifie la variable d'environnement PWD
- `cd` tout seul ramène au home directory (HOME)
- `cd -` ramène au dossier précédent

```
mkdir dossier1
cd dossier1
cd ..
cd /
cd
```

Manipulation Dossiers

- mkdir (Make Directory)

- Créer un répertoire

```
pwd
cd dossier1
mkdir dossier1
cd dossier1
pwd
```

Manipulation Dossiers

- mkdir (Make Directory)

- mkdir -p crée toute l'arborescence si nécessaire
- mkdir -m crée le dossier avec les droits précis

```
mkdir -p test/deep/dir
mkdir -m 0777 free_dir
```

Manipulation Dossiers

- rmdir (Remove Directory)

- Supprime un répertoire vide

```
mkdir dir
rmdir dir
```

Manipulation Fichiers

- touch

- Met à jour la date de modification du fichier
- Par effet de bord : cela crée le fichier s'il n'existait pas

```
ls
touch file1
ls
```

Manipulation Fichiers

- ls (List Segments)

- Affiche le contenu du dossier courant

```
ls
mkdir Dossier1
ls
cd dossier1
ls
```

Manipulation Fichiers

- ls (List Segments)

- ls -a affiche les fichiers « cachés »
  (commençant par un . (point))
- ls -l affichage des propriétés des fichiers
  (droits, propriétaire, groupe, taille, date de modification)
- ls -t trier par date de modification

```
ls -a
ls -la
```

Manipulation Fichiers

- cp (Copy)

- Copie un fichier ou un répertoire (et son contenu)

```
touch file1
ls
cp file1 file_copy
ls
```

Manipulation Fichiers

- cp (Copy)

- cp -r copie récursivement un répertoire

```
mkdir -p test/deep/dir
touch test/deep/dir/file
cp -r test dir2
```

Manipulation Fichiers

- rm (Remove)

- Supprime un fichier

```
touch file1
rm file
```

Manipulation Fichiers

- rm (Remove)

- rm -i demande une confirmation avant de supprimer
- rm -f force la suppression sans message
- rm -r supprime de façon récursive
  (utile pour supprimer des dossiers)

```
touch file1
rm -i file
mkdir -p test/deep/dir
touch test/deep/dir/file
rm -rf test
```

!!! ATTENTION !!!
NE JAMAIS FAIRE
`rm -rf /`

Manipulation Fichiers

- mv (Move)

- Déplace un fichier ou un dossier
- Par effet de bord : renomme un fichier ou dossier

```
touch file1
mv file1 file
```

Manipulation Fichiers

- mv (Move)

- mv -i demande une confirmation avant d'écraser un fichier existant
- mv -f force l'écrasement de fichier si nécessaire

```
touch file1 file2
mv -f file1 file2
```

Manipulation Fichiers

- ln (Link)

- Crée un lien symbolique (symlink) ou matériel (hardlink) vers un fichier ou dossier
- Sur Windows, il s'agit des raccourcis

```
touch file1
ln file1 link_to_file
```

Manipulation Fichiers

- ln (Link)
- ln crée un lien matériel/hardlink
  (crée une entrée dans le dossier qui pointe vers l'i-node du fichier et augmente un compteur de lien sur le fichier : si le compteur atteint 0 suite à des rm, le fichier est supprimé du disque.)
- ln -s crée un lien symbolique
  (un i-node indépendant du fichier est créé pour pointer vers le fichier : si le fichier est détruit, mais pas le symlink, celui-ci renverra une erreur lorsque l'on essayera de l'utiliser)

```
touch file1
ln -s file1 symlink_to_file
ln file1 hardlink_to_file
```

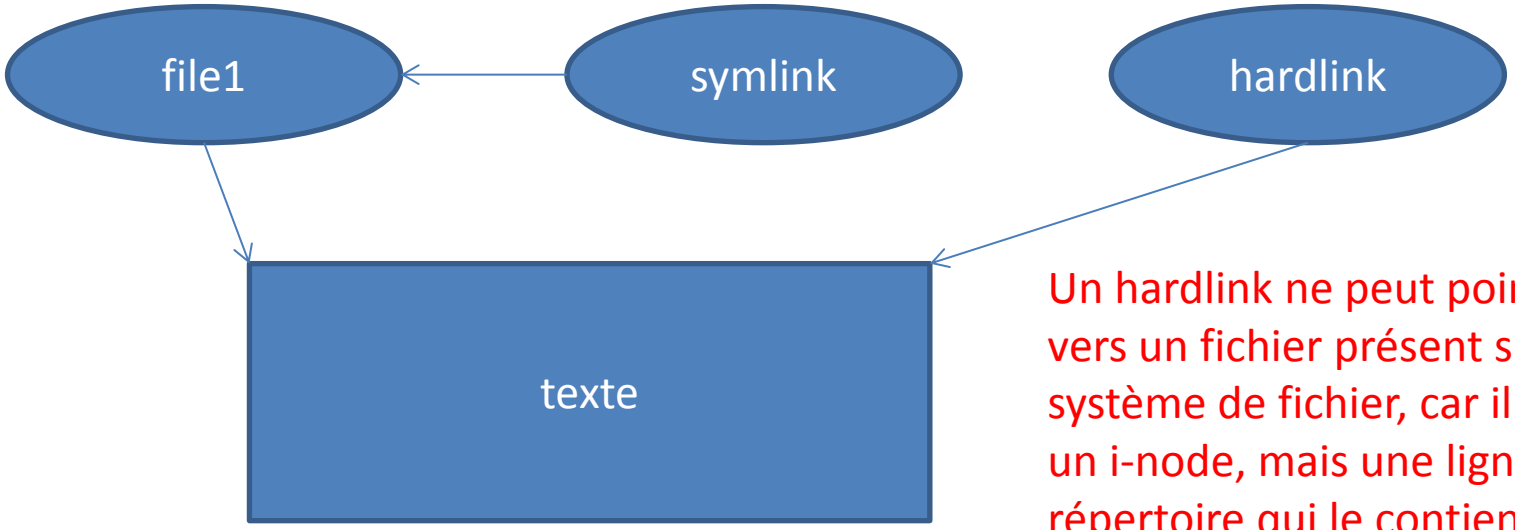
Manipulation Fichiers

```
echo "texte" > file 1

ln -s file1 symlink

ln file1 hardlink
```

Un symlink dispose de son propre i-node, et contient dans son bloc de données le chemin du fichier pointé
ex : ../../file2
ou : /usr/home/file2



Un hardlink ne peut pointer QUE vers un fichier présent sur le même système de fichier, car il n'est pas un i-node, mais une ligne dans le répertoire qui le contient

Types Fichiers

- file

- Permet d'identifier un fichier selon son en-tête (le « magic number », les premiers caractères), ou sa structure

```
echo "#! /bin/sh" > file1
file file1

echo "Coucou" > file1
file file1

file /bin/ls
```

Espace Disque

- df (Disk Free)

- Affiche l'espace libre restant de la partition courante

```
cd
df
cd /
df
```

Espace Disque

- df (Disk Free)

- df -h affiche l'espace libre restant dans un format « humain »
  (avec des unités adaptées (Go ou To))
- df -a affiche l'espace libre de toutes les partitions et disques reliés à la machine
- df -l affiche l'espace libre des partitions/disques locaux
- df -i affiche l'espace libre en terme d'inodes

```
df -ah
df -i
df -l
```

Espace Disque

- du (Disk Usage)

- Affiche la taille d'un/des fichier(s) ou dossier(s) à partir du dossier courant

```
cd
du
mkdir dir1
du dir1
cd dir1
du
```

Espace Disque

- du (Disk Usage)

- du -h affiche l'espace utilisé dans un format « humain » (avec des unités adaptées (Go ou To))
- du -a détaille les fichiers, pas seulement les dossiers
- du -L entre dans les liens symboliques
- du -P n'entre PAS dans les liens symboliques
- du -s affiche un total pour chaque argument
- du -x effectue la recherche sur un seul système de fichiers
- du --block-size=*SIZE* affiche les tailles selon un multiple de *SIZE*

```
du -ah .
du -a --block-size=1024
du -s --block-size=1024 /bin /var
```

Copie Données

- dd

- Copie des données depuis une source vers une destination
- Possibilité de limiter ce que l'on copie (une section précise)
- Possibilité de convertir le format de blocs

- !!! Attention !!!
  Possibilité de détruire ses partitions et ses fichiers

Copie Données

- dd

- if=*FILE* Input File / Le fichier en entrée ou la source à lire
- of=*FILE* Output File / Le fichier de sortie ou le device où écrire
- bs=*SIZE* Block Size / Transfert par blocs de taille *SIZE*
- count=*NB* copie *NB* blocs exactement
- skip=*NB* saute les *NB* premiers blocs lors de la lecture du if=
- seek=*NB* saute les *NB* premiers blocs lors de l'écriture sur of=
- ibs=*SIZE* lit depuis if= par blocs de taille *SIZE*
- obs=*SIZE* écrit sur of= par blocs de taille *SIZE*
- conv=*CONV* convertit depuis un format vers un autre
- iflag=*FLAGS* paramètres pour le programme (synchro I/O, symlinks…)

```
dd bs=10K count=100K if=/dev/cdrom of=./file1
dd conv=ucase if=./file1 of=./file2
dd conv=ebcdic if=./script.sh of=./ICEGENER.JCL
```

Droits Fichier & Dossiers

- chmod (Change Mode)

- Modifie les droits des fichiers ou dossiers
- Fonctionne avec les droits en chiffres ou en lettres

```
chmod 750 file1
chmod -R 750 dir1
chmod u=rwx,g=r,o= file1
chmod a=rx prog1
chmod u+w,g-w file1
```

Droits Fichier & Dossiers

- umask

- Modifie le umask, c'est-à-dire les permissions par défaut lors de la création de fichiers ou dossiers pour la session en cours
- Fonctionne avec les droits en chiffres ou en lettres
  - En chiffres, le umask est inversé sauf sur la première propriété (umask 0022 équivaut à chmod 0755)

```
umask
umask -S
umask 0022
umask a-r
```

Droits Fichier & Dossiers

- chgrp (Change Group)

- Modifie le groupe d'appartenance de fichiers ou dossiers

```
chgrp www file1
chgrp daemon file1 file2
chgrp -R www dir1
```

Droits Fichier & Dossiers

- chown (Change Owner)

- Modifie le propriétaire d'un fichier ou dossier
- Peut aussi modifier le groupe

```
chown user file1
chown user:daemon file1 file2
chown -R www:www dir1
```

Droits Fichier & Dossiers

- chown (Change Owner)

- chown -h modifie uniquement un lien symbolique, pas sa cible

```
touch file1
ln -s symlink file1
chown -h www:www symlink
```

Droits Avancés

- getfacl (Get File Access Control List)

- Affiche les droits ACL des fichiers et dossiers
- Parfois aussi : `fs getacl`

```
getfacl file1
```

```
fs getacl file1
```

Droits Avancés

- setfacl (Set File Access Control List)

- Modifie les droits ACL des fichiers et dossiers
- Parfois aussi : `fs setacl`

```
setfacl -m u:user:r file1
setfacl -x g:staff file1
setfacl -m m::rx file1
```

Droits Avancés

- setfacl (Set File Access Control List)

- setfacl -m modifie les droits
- setfacl -x supprime une entrée (group/user)

```
setfacl -m u:user:r file1
setfacl -x g:staff file1
setfacl -m m::rx file1

getfacl file1 | setfacl --set-file=- file2
```

Recherche Commande

- which

- Permet de savoir où se trouve le programme qui sera appelé par la commande
- Si which est un programme : ne peut pas différencier les built-ins des programmes

```
which which
which ls
which echo
```

Recherche Commande

- whereis

- Liste les endroits où se trouvent les programmes, les sources, et les manuels d'une commande
- whereis ne peut différencier les built-ins des programmes

```
whereis ls
whereis which
whereis whereis
```

Recherche Commande

- whereis

- whereis -b recherche les binaires uniquement
- whereis -s recherche les sources uniquement
- whereis -m recherche les manuels uniquement

```
whereis -b ls
whereis -m which
```

Basename

- basename

- Affiche/Extrait uniquement le nom du fichier (ou dossier) à partir du chemin complet

```
basename /bin/ls
basename ../../home/user
```

Basename

- basename

- basename -a gère plusieurs chemins
- basename -s *SUFFIX* supprime du nom affiché le suffixe associé

```
basename -a /bin/ls /etc/passwd
basename -s .log /var/log/user.log
basename -s log /var/log/user.log
```

Dirname

- dirname (Directory Name)

- Affiche/Extrait uniquement le nom du chemin contenant le fichier (ou dossier)
- N'affiche pas le chemin absolu (ce n'est pas pwd)

```
dirname /bin/ls
dirname ../../home/user
```

Find

- find

- Recherche des fichiers dans l'arborescence
- En pratique, il liste ce qui correspond à un pattern précis
- S'accorde « parfaitement » avec `grep` et un pipe

```
find .
find /bin /home
find . -name 'E*'
```

Find : timestamps

- find tests

- find . -atime *n* cherche les fichiers accédés dans les *n* * 24h
- find . -amin *n* cherche les fichiers accédés dans les *n* mins
- find . -mtime *n* cherche les fichiers modifiés dans les *n** 24h
- find . -mmin *n* cherche les fichiers modifiés dans les *n* mins

```
find . -atime -1
find . -mmin -60
find . -atime +36
```

Find : timestamps

- find tests

- find . -ctime *n* cherche les fichiers dont les statuts ont été modifiés dans les *n** 24h
- find . -cmin *n* cherche les fichiers dont les statuts ont été modifiés dans les *n* mins

```
find . -cmin -60
find . -cmin 3
```

Find : owners

- find tests

- find . -uid *u* cherche les fichiers qui appartiennent à l'UID *u*
- find . -user *u* cherche les fichiers qui appartiennent à l'utilisateur *u*
- find . -nouser cherche les fichiers dont l'UID n'existe pas (l'utilisateur a disparu, par exemple)

```
find . -user matt
find . -uid 1003
```

Find : owners

- find tests

- find . -gid *g* cherche les fichiers qui appartiennent au GID *g*
- find . -group *g* cherche les fichiers qui appartiennent au groupe *g*
- find . -nogroup cherche les fichiers dont le GID n'existe pas (le groupe a disparu, par exemple)

```
find . -group daemon
find . -nogroup
```

Find : file type

- find tests

- find . -type *t* chercher les fichiers selon leur type *t* :

b	c	d	p	f	l	s
Fichiers spéciaux (mode block)	Fichiers spéciaux (mode char)	Répertoires	Tubes nommés (named pipes)	Fichiers normaux	Liens symboliques	Sockets

```
find . -type d
find . -type p
```

Find : depth

- find expression

- find . -maxdepth *L* — descend au maximum à une profondeur *L*
  find . -mindepth *L* — ignore tout ce qui n'est pas au moins à une profondeur *L* minimum
- find . -mount
  find . -xdev — ne descend pas dans d'autres partitions (ou FS) différentes

```
find . -maxdepth 4
find . -xdev
```

Find

- find

```
find . -maxdepth 4 -type d -uid 1003   \
-atime -1 -name `E*`
```

Find : exec

- find exec

- find . -exec *utility args {} \;* exécute le programme *utility* avec les arguments *args* sur chacun des résultats

- find . -exec *utility args {} +* exécute le programme *utility* avec les arguments *args* sur tous les résultats concaténés en une ligne

```
find . –exec echo {} \;
find . –exec echo {} +
```

Find : exec

- find exec

```
find . –exec echo {} \;
find . –exec echo {} +

find -atime -1 -exec echo -e {} {} \\n \;

find . -name "*~" -exec ls {} \;
find . -name "*~" -exec rm -f {} \;
```

Grep

- grep (Global Regular Expression print)

- Filtre du texte selon un pattern (voir regexp)
- Recherche dans des fichiers ou des flux
- RegExp classiques ou étendues (`grep -E`) possibles
- Famille de `grep` : `egrep`, `fgrep`, `rgrep`, …

```
grep "chaine" fichier1.txt
find . | grep filename
```

Grep : recherche dans des fichiers

- grep

- grep -i recherche majuscules ou minuscules
- grep -r recherche récursive dans dossiers (-R pour symlinks)
- grep -c affiche seulement le nombre de lignes qui matchent
- grep -v inverse la sélection (affiche ce qui ne matche pas)

```
grep -ri "chaine" dir1
grep -rciv "LOL"  *
```

Grep : recherche dans des fichiers

- grep

- grep -w pattern recherché forme exactement un mot
- grep -x pattern recherché forme exactement une ligne
- grep -m *N* stoppe la lecture du fichier après *N* matchs
- grep -n affiche le numéro de ligne où le match se produit

```
grep -iw -m 10 "alias" .bashrc
grep -riwn "LOL"  *
```

Grep : recherche dans des fichiers

- grep

- grep -a lit les fichiers binaires comme du texte
- grep -I ignore les fichiers binaires

- grep --exclude=*GLOB* ignore les fichiers qui respectent le pattern *GLOB*
- grep --exclude-dir=*GLOB* ignore les dossiers qui respectent le pattern *GLOB*

```
grep -rI --exclude-dir="*dir*" "text"  .
```

Locate

- locate

- Recherche des fichiers ou dossiers selon leur nom
- Utilise une base de données régulièrement mise à jour pour chercher les noms (donc utilitaire très rapide)
- Attention si la base n'est pas à jour…
- Fonctionne en mode globbing (défaut) ou regexp (option)

```
locate filename
locate filenam
```

Locate

- locate

- locate -i insensible à la casse (majuscule/minuscule)
- locate -l *N* limite à *N* résultats max
- locate --basename recherche uniquement le basename
- locate --regex *EXPR* recherche en suivant une regexp simple (désactive le globbing)

```
locate -i -l 10 .log
locate -i --regex "\.log$"
locate -i --regex "^.*log$"
```

Locate : updatedb

- locate / updatedb

- /usr/libexec/locate.updatedb
- updatedb force la mise à jour de la base de données de locate
  (le programme change de nom selon l'OS)

```
updatedb
/usr/libexec/locate.updatedb
```

ps

- ps (Process Status)

- Affiche l'état des processus en cours pour l'utilisateur courant
- Attention : les paramètres ne se passent pas toujours de la même manière selon l'UNIX utilisé
- Colonnes classiques : PID, TTY, TIME, CMD
  - PID : numéro du processus
  - TTY : TeleTYpe / terminal auquel le processus est rattaché
  - TIME : temps processeur consommé depuis le lancement (user + kernel)
  - CMD : commande utilisée pour lancer le processus

ps

ps

- ps (Process Status)

- ps -a affiche les processus de tous les utilisateurs
- ps -u affiche des informations supplémentaires
  OU affiche les processus d'un utilisateur précis
- ps -x affiche les processus qui ne sont pas rattachés à un terminal

```
ps -auuser
ps -aux
ps aux
```

ps

- ps (Process Status) SUS & POSIX options

- ps -e affiche tous les processus
- ps -u affiche les processus d'un utilisateur précis
- ps -f affiche plus d'informations sur les processus
- ps -l affiche encore plus d'informations

```
ps -ef
ps -el
ps -eu user
```

ps

- ps (Process Status)

- Quelques infos sur les colonnes :
  - State : Running, Sleep, Idle, Disk wait, Lock waiting, sTopped, Zombie
  - PRI : priorité (voire nice & renice)
  - PGID : process group id, id du groupe de processus (ex : un enchaînement de commandes va créer un groupe à partir du 1ère programme, le PGID sera le PID de ce programme)
  - SID : session id, le PID du programme qui a ouvert une session (le shell en général)
  - EUID : effective UID, l'UID utilisé par le processus pour accéder au système de fichier (EUID, EGID)
  - RUID : real UID, l'UID utilisé par le processus pour émettre des signaux

kill

- kill

- Envoie un signal à un processus (PID)
- Certains signaux sont rattrapables, et d'autres non
- Permet de tuer, stopper, reprendre le processus en cours
- …ou juste d'envoyer un signal qui sera traité de façon particulière

```
kill -9 1010
kill -s SIGKILL 1010
```

kill

- kill

- kill -l affiche les noms et numéros des signaux
- kill -l *NUM* affiche le nom du signal numéro *NUM*
- kill -s *NAME* envoie le signal *NAME* (nom complet)
- Si -1 est utilisé à la place d'un PID, cela envoie à tous les processus de l'utilisateur courant
  - Attention en root… -1 fait rebooter/éteindre la machine

```
kill -l 30
kill -9 -1
```

kill

- kill

- Signaux impossibles à rattraper :
  - SIGKILL (9) : tuer/terminer le processus immédiatement
  - SIGSTOP(17) : stopper immédiatement le processus

- Pour relancer un processus stoppé, utiliser le signal :
  - SIGCONT (19) : Continue

- Signaux dont l'usage est à définir dans le code source soi-même :
  - SIGUSR1(30), SIGUSR2(31)

nice

- nice

- Lance un programme avec une priorité précise dans le scheduler
  - Utile si le CPU est surchargé
- Plus la valeur est faible, plus le processus est prioritaire
  - Seul root peut mettre une priorité négative
- Incrémente la priorité par l'argument donné (par défaut : 10)
  - Un programme lancé sans nice a une priorité de 0

```
nice date
nice -n 10 date
nice -n -5 date
```

renice

- renice

- Modifie la priorité d'un processus dans le scheduler
- Plus la valeur est faible, plus le processus est prioritaire
  - Seul root peut mettre une priorité négative
- Incrémente la priorité par l'argument donné

```
renice -n 10 4322
```

renice

- renice

- renice -p modifie un processus PID (défaut)
- renice -g modifie un groupe de processus PGID
- renice -u modifie les processus d'un utilisateur précis (UID)

```
renice -n 10 4322
renice -g -n 30 610
renice -u -n 15 1003
```

nohup

- nohup (No Hangup)

- Ne tue pas une tâche qui fonctionne, même si le login shell est fermé
  - Utile si un gros job doit être lancé et qu'il prendra beaucoup de temps
- En pratique :rend la commande immune au signal SIGHUP (signal qui est transmis lorsque l'on se logout ou que l'on ferme le terminal hébergeant la tâche)

```
nohup bash
```

sleep

- sleep
- Attend un certain temps avant de reprendre l'exécution
- Décompte en secondes

```
sleep 1
sleep 5
```

time

- time

- Mesure le temps d'exécution d'une commande
- Le temps est écrit sur STDERR, pas STDOUT

- Real : temps total réel d'exécution
- User : temps CPU consommé en userland
- Sys : temps CPU système consommé (en kernelland)

```
time ls
```

time

- time
- time -p écrit le temps dans un format précis

```
time -p ls
```

date

- date

- Affiche ou met à jour l'heure

```
date
```

date

- date

- date -u affiche l'heure universelle (UTC+0/GMT+0)
- date +FORMAT affiche l'heure dans le format voulu :
  - %Y année (entière)
  - %y année (dizaine et unité)
  - %m mois (1 - 12)
  - %d jour du mois (1 - 31)
  - %H heure (24h)
  - %M minute
  - %S seconde

date

- date

- date +FORMAT affiche l'heure dans le format voulu :

%j	jour dans l'année (1 - 366)
%W	numéro de semaine (0 - 53)
%u	num du jour de la semaine (1 - 7)
%B	nom du mois
%A	nom du jour de la semaine
%t	tabulation
%n	nouvelle ligne
%%	symbole %

date

- date

```
date -u

date +%Y-%m-%d%t%Hh%Mm%Ss
date +%Y-%m-%d--%H:%M:%S
date +%Y-%m-%d%t%Hh%Mm%Ss
date +%Y-%W-%u--%j
```

date

- date

- Sur UNIX, le temps est calculé en secondes depuis « l'Epoch », c'est-à-dire le 1er janvier 1970 à 00h00:00
- Le « timestamp » est le nom donné à une date décomptée en secondes depuis « l'Epoch ». Il est possible de convertir les dates et obtenir le timestamp : `%s` donne le nombre de secondes écoulées depuis l'Epoch.

```
date +%s
date --date="Oct 1 09:00:00 BST 2009" +%s
```

touch (retour)

- touch

- touch -t *FORMAT* met le timestamp à *FORMAT :*
  [[*CC*]*YY*]*MMDDhhmm*[.*SS*]
  CC siècle, YY année,
  MM mois [01-12],DD jour [01-31],
  hh heure [00, 23], mm minute [00, 59],
  .SS seconde [00, 60]

```
touch -t 10211430 file1
```

touch (retour)

- touch

- touch -c ne crée pas de fichier s'il n'existe pas
- touch -a change l'heure d'accès
- touch -m change l'heure de modification
- touch -r *file* copie les caractéristiques de *file*

```
touch -t 10211430 file1
touch -t 10211430 -a file1
touch -r reffile file1
```

cron

- cron (Vixie Cron)

- Planificateur de tâches sur UNIX
- Lit un fichier `crontab` puis exécute aux heures indiquées les commandes
- Souvent, uniquement contrôlable par `root`
- Est lancé automatiquement par le système
- Lit un fichier `crontab` dans lequel se trouve des tâches planifiées

`cron`

crontab

- crontab

- Fichier contenant les tâches planifiées (comme une « table »)
- Une « table » par utilisateur
- Permet de modifier les tâches planifiées
- Les tâches lancées par cron/crontab utilisent le shell de l'utilisateur cron
  - Penser à vérifier la syntaxe que ce shell accepte (sh ou csh)

`crontab`

crontab

- crontab

- crontab -l affiche les tâches prévues
- crontab -r supprime la table courante
- crontab -e édite la table avec l'éditeur texte par défaut
- crontab -u *USER* gère les tâches de l'utilisateur *USER*

```
crontab -l
crontab -e
crontab -u user -l
```

crontab

- crontab

- Format : * * * * * commande à exécuter

minute(0-59) heure(0-23) jour(1-31) mois(1-12) jour_semaine(0-6)

- jour_semaine : du dimanche au samedi, parfois, 7 = dimanche)
- * = tous les

ex : effacer la error_log tous les jours à 00:01

```
1 0 * * * printf > /var/log/apache/error_log
```

crontab

- Exécuter script.sh tous les samedi (6e jour semaine) à 23h45

```
45 23 * * 6 /home/user/script.sh
```

- Exécute le 1er et le 15 de chaque mois, et tous les lundi

```
0 0 1,15 * 1 /home/user/script.sh
```

- Exécute seulement le lundi

```
0 0 * * 1 /home/user/script.sh
```

- Seulement une date par an

```
0 12 14 2 * /home/user/script.sh
```

crontab

- crontab

Une fois par an le 1er janvier à minuit	`0 0 1 1 *`
Une fois par mois le 1er jour à minuit	`0 0 1 * *`
Une fois par semaine le dimanche à minuit	`0 0 * * 0`
Une fois par jour à minuit	`0 0 * * *`
Une fois par heure (début heure)	`0 * * * *`

at

- at

- Exécute des commandes en mode différé, sans terminal de rattachement
- Copie les variables d'environnement (ENVVARS), le dossier courant (CWD), le masque de création de fichiers (UMASK), et quelques autres options « lors » de l'inscription de la tâche
- Les utilisateurs doivent être déclarés dans at.allow ou at.deny
  - Fichier `at.deny` vide = tout le monde peut utiliser la commande `at`

```
at now
```

at

- at

- at -f *FILE* lit le fichier *FILE* qui contient des commandes
- at -m envoie un mail à l'utilisateur une fois les tâches effectuées
- at -l liste les jobs en attente de l'utilisateur courant
- at -r *AT_ID* supprime le job en attente n° *AT_ID*
- at -q *QUEUE* envoie le job dans la file *QUEUE*
  file `a` par défaut, file `b` pour batch

```
at -f file1 -q a
```

at

- at

- at *timespec* date au format :
  midnight, noon, now
  today, tomorrow
  2pm + 1 week, 2pm next week, …
  (supporte : minutes, hours, days, weeks, months, years)

```
at -f file1 now
```

at

- at
- at -t *time_arg* date au format : [[*CC*]*YY*]*MMDDhhmm*[.*SS*]
  CC siècle, YY année, MM mois [01-12],
  DD jour [01-31], hh heure [00, 23],
  mm minute [00, 59], SS seconde [00, 60]

```
at -f file1 -t 201710302342.00
at -f file1 -t 201710302342
at -f file1 -t 10302342
at -t 10302342
```

batch

- batch

- Équivalent à la commande : `at -q b -m now`
- Lance des tâches immédiatement, si le processeur n'est pas trop chargé
- Comme la commande `at`, il va lire l'entrée standard par défaut
  - Mais batch ne fait « que » lire l'entrée standard
  - Termine la lecture par un `Ctrl + D` ou `EOF`

```
batch
```

yes

- yes

- Ecrit indéfiniment « yes »
- Ecrit indéfiniment un texte précis
- (Ctrl C pour quitter)

```
yes
yes haha hehe hoho hihi huhu hyhy
```

mktemp

- mktemp (make temporary)

- Créer un fichier (ou dossier) avec un nom aléatoire
- Permet de créer un fichier qui n'existe pas
- Renvoie le nom du fichier créé sur STDOUT
- Utilise un template (remplace les XXXX par des caractères)
- Ne pas oublier de le supprimer avant la fin du traitement

```
mktemp
mktemp FichierXXXXXX
```

mktemp

- mktemp (make temporary)

- mktemp -d crée un répertoire plutôt qu'un fichier
- mktemp -q échoue silencieusement (ne fait pas crasher le script ni n'écrit sur STDERR)
- mktemp -p *DIR* utilise *DIR* comme préfixe du chemin
- mktemp -t renvoie un chemin complet vers le fichier (sans -p, utilise le dossier /tmp)

```
mktemp -p ~ -t temporary.XXXXX
TMP_FILE=`mktemp -t temporary.XXXXX`
```

xargs

- xargs

- Construit une ligne de commande à partir d'une commande (en argument), et d'une liste de paramètre sur l'entrée standard
  - Espaces, tabulations, retour à la ligne, EOF comme séparateurs
- Utile pour rediriger des flux
- Parfois, ligne de commande trop « longue », xargs aide

```
find /path -type f -print | xargs rm
```

xargs

- xargs

- xargs -t écrit les traces d'exécution sur STDERR
- xargs -p demande confirmation de chaque ligne construite
- xargs -x quitte si la taille de la commande est trop longue
- xargs -I *CHAR* place les arguments dans la ligne à la place du/des caractère(s) *CHAR*

```
find /path -type f -print | xargs rm
```

```
… | xargs -I {} -t mv dir1/{} dir2/{}
```

xargs

- xargs

- xargs -n *NUM* place *NUM* arguments maximum par ligne de commande effectuée (cmd arg1 arg2 …)
- xargs -L *NUM* utilise au plus *NUM* lignes pour construire la ligne de commande (implique -x)
- xargs -s *NUM* utilise au plus *NUM* caractères pour construire la ligne de commande

```
echo {0..9} | xargs -n 2
```

cat

- cat (Concatenate)

- Concatène des fichiers et les imprime
- Utilisation quasi exclusivement avec des redirections et pipes
- Redirige aussi l'entrée standard

- !!! ATTENTION !!! Aux caractères non imprimables contenus dans des fichiers : ceux-ci peuvent modifier le comportement du terminal

```
cat file1
```

cat

- cat (Concatenate)

- cat - lit STDIN, et renvoie ligne à ligne vers STDOUT jusqu'à lire un Ctrl + D ou un EOF
- cat (même chose que `cat –`)

```
cat
cat > file1
cat > file2 << EOF
cat > file3 << fin
```

cat

- cat (Concatenate)
- cat -u copie les caractères immédiatement/sans buffering (fait des read non bloquants)

```
mkfifo foo
cat -u foo > /dev/tty13 &
cat -u > foo
```

sort

- sort

- Trie le contenu des fichiers
- Fusionne le contenu de fichiers
- Vérifie qu'un fichier est trié

- Tri numérique, alphanumérique, inverse

```
echo -e "3\nB\nC\nx\n4\nz\ny\n1\n2\nA" > file1
sort file1
```

sort : tri

- sort
- sort -n tri numérique (valeur de la chaîne)

```
echo -e "03\n-0\n-1\n0\n42\n008\n  13" > file2
sort file2
sort -n file2
```

sort : tri

- sort

- sort -i ignore les caractères non imprimables
- sort -f ignore la casse (lowercase/uppercase)
- sort -r inverse l'ordre
- sort -o *OUTPUT* écrit la sortie triée dans le fichier *OUTPUT*

```
sort -i -f -r file1
sort -o fileout -i file1
```

sort : tri

- sort

- sort -b ignore les caractères blancs en début de colonne
- sort -d caractères blancs et alphanumériques utilisés (impossible à utiliser avec -i ou -n)

```
echo -e " C\nA\n   B" > file3
sort file3
sort -b file3
```

sort : tri

- sort
- sort -u ne garde qu'une seule occurrence en cas de doublon

```
echo -e "E\nc\nD\nD\nb\na\nA\nB" > file4
```

```
sort -u file4
```

```
sort -f -u file4
```

sort : tri

- sort

- sort -k *NUM* trie selon la colonne *NUM*
- sort -k *X,Y* trie selon les caractères entre la colonne *X* et *Y*
- sort -t'*CHAR*' le caractère *CHAR* est utilisé comme délimiteur au lieu de espace

cas spécial : caractères échappés… pour tab : `-t $'\t'`

```
echo -e "A|3\nB|1\nC|4\nD|2" > file5
sort -t'|' -k2 file5
```

sort : tri

- sort

- sort -k *NUM[type]* trie selon la colonne *NUM*
- sort -k *X[type],Y[type]* trie selon les caractères entre la colonne *X* et *Y*

- *[type]* décrit le type de colonne : b, d, f, i, n, r
- Ordre des `-k` donne priorité des colonnes comparées

```
echo -e "G|2|w\nG|1|x\nC|2|y\nD|2|z" > file6
sort -t'|' -k2n -k1,1f file6
```

sort : vérification

- sort

- sort -c teste si le fichier est ordonné
  envoie une erreur sur STDERR en cas de problème
  valeur de retour à 0 si tout est OK
- sort -C comme `sort -c`, mais n'écrit rien sur STDERR
  il faut juste comparer la valeur de retour

```
sort -c file1
sort -C file1
```

sort : fusion

- sort

- sort -m fusionne/merge plusieurs fichiers en un
  les fichiers DOIVENT être triés avant
  `sort -c` permet de vérifier si le tri est bon

```
echo -e "D\nJ\nA\nC" > merge1
echo -e "E\nC\nH\nG" > merge2
sort -c merge1
sort -o merge1 merge1
sort -o merge2 merge2
sort -c merge1
sort -m -o merge_out merge1 merge2
```

uniq

- uniq

- Supprime les lignes successives en double
- Ne supprime QUE les lignes adjacentes similaires !
  - Faire un tri avant si nécessaire pour organiser ?...
  - …mais `sort -u` fait déjà ça ?
- Écrit sur la sortie standard ou dans un fichier

```
uniq file
uniq file file_out
uniq - file_out
```

uniq

- uniq

- uniq -d ne copie que les lignes en doublon (mais une seule fois)
- uniq -u ne copie que les lignes qui ne sont pas en doublon
- uniq -c ajoute le nombre de répétitions devant chaque ligne

```
echo -e "AAAA\nBB\nBB\nC\nD\nD\nD\nE" > file7
uniq -d file7
uniq -c file7
uniq -c -u file7
```

uniq

- uniq

- uniq -f *FIELDS* saute les *FIELDS* premiers champs avant de faire la comparaison des lignes
- uniq -s *CHARS* saute les *CHARS* premiers caractères avant de faire la comparaisons des lignes
  Si utilisé avec -f : les *CHARS* premiers caractères « après les champs ignorés » sont ignorés

```
echo -e "A BB\nB BB\nB C\nC C\nD C\nD AA\nD A\nE A\nE AA\nF AA\nG" > file8
uniq -f 1 -s 1 file8
```

comm

- comm

- Cherche et affiche les lignes communes entre deux fichiers
  - Attention, les fichiers doivent être triés
- La sortie affichera 3 colonnes :
  - Les lignes seulement dans le 1er fichier
  - Les lignes seulement dans le 2e fichier
  - Les lignes communes aux deux fichiers

```
echo -e "A\nB\nB\nC\nC\nC\nD\nE\nF" > file10
echo -e "B\nC\nC\nC\nE\nF\nG" > file11
comm file10 file11
```

comm

- comm

- comm -1 n'affiche pas les lignes uniques au 1er fichier
- comm -2 n'affiche pas les lignes uniques au 2e fichier
- comm -3 n'affiche pas les lignes communes au deux fichiers

```
echo -e "A\nB\nB\nC\nC\nC\nD\nE\nF" > file10
echo -e "B\nC\nC\nC\nE\nF\nG" > file11
comm file10 file11
comm -1 -2 file10 file11
comm -3 file10 file11
```

tee

- tee

- Duplique l'entrée standard (se termine par Ctrl + D ou HereDoc)
  - Vers la sortie standard
  - Optionnellement vers un/des fichier(s)
- Pas de bufferisation/Copy directe caractère par caractère

```
tee
tee outfile1 outfile2
tee outfile << fin
```

tee

- tee

- tee -a écrit en mode append dans le/les fichiers
  rappel : « mode append » signifie que l'on ajoute en fin de fichier
- tee -i ignore le signal SIGINT

```
echo "test" > file
tee -a file
tee -i file
```

tr

- tr (translate/transliterate)

- Translate des caractères
- Modifie les caractères de l'entrée standard avant de les mettre sur la sortie standard
  - Substitution ou suppression

```
tr aeiouy vvvvvv
echo "toto" | tr o A
echo "Je m'appelle Lolo" | tr aeiouy bfjqvz
```

tr

- tr (translate/transliterate)

- tr -d supprime tout caractère qui appartient à la première chaîne passée en paramètre
- tr -s supprime les doublons de caractères qui se suivent
- tr -sd *DEL DOUB* Supprime tous les caractères *DEL*, et supprime les doublons *DOUB*

```
echo "aaaabbbccc" | tr -s a
echo "aaaabbbccc" | tr -d bc
echo "aaaabbbccc" | tr -sd a bc
```

tr

- tr (translate/transliterate)

- tr -c complément des caractères de la 1ère chaîne
- tr -C complément des valeurs de la 1ère chaîne

```
echo "aaaabbbccc" | tr -c a z

echo "aaaabbbccc" | tr -d a
echo "aaaabbbccc" | tr -cd a
```

tr

- tr (translate/transliterate)

- Classes reconnues :
  `[:alnum:] [:alpha:] [:digit:] [:xdigit:]`
  `[:blank:] [:space:] [:lower:] [:upper:]`
  `[:punct:] [:print:] [:graph:] [:cntrl:]`

```
echo "Ceci est    1 phrase" | tr -s [:blank:]
echo "aaaabbbccc" | tr -d bc
```

cut

- cut

- Extrait des octets, des caractères, ou des champs

- Extraction selon des numéros de colonnes
- Extraction selon des numéros de champs
  - Supprime les lignes qui ne contiennent pas de délimiteur

```
cut -d : -f 1,7 /etc/passwd
cut -c 2,4 -
```

cut

- cut

- cut -c *list* coupe les caractères à la/les colonnes *list*
- cut -b *list* coupe les octets désignés par *list*
- cut -f *list* coupe les champs désignés par *list*
  (séparés par TAB)

```
echo "Toto" | cut -c 3
echo "Toto" | cut -c 2-8
echo "Toto" | cut -c 2,4
echo "Toto" | cut -b 1
```

cut

- cut -f

- cut -s supprime les lignes qui n'ont pas de délimiteur
- cut -d *delim* choix des délimiteurs dans *delim*
- cut -f *list* coupe les champs désignés par *list*
  (séparés par TAB, par défaut)

```
echo -e "user1:val\nBLOB\nuser3:val" > file1
cut -f 1 -d ":" file1
cut -f 2 -d ":" file1
cut -f 1 -d ":" -s file1
```

paste

- paste

- Concatène des lignes d'un fichier, et écrit le résultat sur la sortie standard
- Les '\n' sont remplacés par des TAB ('\t')

```
echo -e "user1:val\nBLOB\nuser3:val" > file1
echo -e "Test1\n\nTest2\n\n\n\n" > file2

paste file1 file2
paste file1 -
```

paste

- paste

- paste -d *delim* change le délimiteur de sortie
  (accepte aussi : '\n', '\t', '\\', '\0')
- paste -s concatène toutes les lignes de chaque fichier en une ligne (avec le délimiteur si nécessaire), sépare chaque fichier par un '\n'

```
echo -e "Tata\nToto\nTiti\nTutu\n" > file3
paste -d "#" file1 file2
paste -s -d "#" file1 file2 file3
```

join

- join

- Similaire à la jointure en base de données
- Colle/Mélange deux fichiers en un selon un critère commun
  - Utilise la première colonne des 2 fichiers pour faire la jointure
  - Puis copie les données des autres champs
- Les fichiers doivent déjà être triés sur la clé (cf *sort*)

```
echo -e "1 Fabrice\n1 Elena\n2 Ali" > file1
echo -e "1 Archi\n2 BDD\n3 POO" > file2
join file1 file2
```

join

- join

- join -1 *NUM* désigne avec *NUM* la colonne sur laquelle la jointure se produit pour le premier fichier
- join -2 *NUM* désigne avec *NUM* la colonne sur laquelle la jointure se produit pour le deuxième fichier

```
echo -e "Fabrice 1\nElena 1\nAli 2" > file3
echo -e "Archi 1\nBDD 2\nPOO 3" > file4
join -1 2 -2 2 file3 file4
```

join

- join
- join -t *CHAR* utilise le caractère CHAR comme séparateur de colonne dans les deux fichiers

```
echo -e "1:Fabrice\n1:Elena\n2:Ali" > file5
echo -e "1:Archi\n2:BDD\n3:POO" > file6
join -t : file5 file6

echo -e "1 Fabrice\n1 Elena\n2 Ali" | join - file6
echo -e "1:Archi\n2:BDD\n3:POO" | join file5 -
```

join

- join

- join -a *NUM* ajoute les lignes sans correspondance du fichier numéro *NUM*
- join -v *NUM* affiche sur la sortie standard uniquement les lignes sans correspondance du fichier numéro *NUM*

```
join -a 1 file1 file2
join -a 1 file2 file1
join -v 2 file1 file2
join -v 1 -v 2 file1 file2
```

join

- join

- join -o *LIST* modifie la sortie par le format décrit dans *LIST*
  1.x désigne la colonne x du fichier 1
  2.y désigne la colonne y du fichier 2
  0 désigne la colonne de jointure
- join -e *STR* remplace les lignes vides de -o par *STR*

```
echo -e "1 Fabrice B\n1 Elena K" > file5
echo -e "1 Archi OS\n2 BDD DB\n3 POO" > file6
join -a 2 -e AH -o 1.2,0,2.2,2.3 file5 file6
```

wc

- wc (Word Count)
- Compte le nombre de mots, lignes, caractères, ou octets dans un ou des fichiers
  - Les mots sont séparés par des espaces par défaut
- Affiche dans l'ordre, pour chaque fichier, le nombre de :

  Retours à la ligne '\n', Mots, Octets, Nom du Fichier

  (si plusieurs fichiers : un total est affiché à la fin)

```
wc file1 file2
wc -
```

wc

- wc (Word Count)

- wc -c écrit le nombre d'octets contenus
- wc -m écrit le nombre de caractères contenus
- wc -l écrit le nombre de retours à la ligne '\n'
- wc -w écrit le nombre de mots (séparés par espace)

```
wc -mlw file1 file2
wc -w file1
```

head

- head

- Affiche les premières lignes d'un fichier
  - Par défaut les 10 premières lignes

- head -n *NUM* affiche les *NUM* premières lignes du fichier (ou ignore les *NUM* dernières)

```
head file1
head -n 13 file2
head -n -13 file2
head -
cat file3 | head -n 3
```

tail

- tail

- Affiche les derniers caractères ou lignes d'un fichier
  - Par défaut les 10 dernières lignes

- Affiche (les derniers)/tous les caractères ou lignes d'un fichier en ignorant les N premiers caractères ou lignes

```
tail file1
tail -
cat file1 | head -n 20 | tail -n 5
```

tail

- tail
- tail -c *NUM* affiche les *NUM* derniers caractères OU ignore les *NUM* premiers caractères
- tail -n *NUM* affiche les *NUM* dernières lignes OU ignore les *NUM* premières lignes

NUM : précédé de + signifie « ignorer les NUM premiers »
précédé de – ou rien signifie « afficher les NUM derniers »

```
tail -n 3 file1
tail -c +15 file2
tail -n +2 file1
```

tail

- tail

- tail -f FILE ne quitte pas à la fin du fichier, mais se met en attente de changements sur le fichier.
  Fonctionne sur des fichiers classiques, mais aussi sur des fifo (mkfifo).
  Ne fonctionne PAS sur l'entrée standard si celle-ci est un pipe ou une fifo.

=> utile pour scruter des logs en temps réel

```
tail -f service.log
```

more

- more
- Affiche le contenu des fichiers dans le terminal en mode page par page (comme les manuels)
- Si la sortie standard de more n'est pas un terminal (mais un pipe…), l'intégralité sur contenu est copiée sans modification

```
more file1
cat file1 | more | cat -
```

more

- more

- more -s si plusieurs lignes consécutives sont vides, elles sont regroupées en une seule ligne vide
- more -c « nettoie » l'écran et écrit à partir du début de l'écran
- more -u affiche les « backspaces » et traite les « \r »
- more -n *NUM* affiche *NUM* lignes par page

```
more -c file
more -u file
more -n 10 file
```

less

- less
- Pas dans le standard SUS (mais utile…)
- Agit comme more… mais supporte de revenir en arrière (il ne fait pas qu'avancer dans le fichier)

```
less file
```

split

- split

- Découpe un fichier en 0 ou plus fichiers
- Tous les fichiers sont de la même taille
- Par défaut, découpe toutes les 1000 lignes
- Par défaut, les fichiers de sorties seront de la forme :
  - xaa, xab, xac, … , xba, xbb, … xzz (676 fichiers max)

```
split file1
split -
echo "lol" | split
```

split

- split

- Si les 676 fichiers en sortie ne suffisent pas à contenir tout le fichier d'origine, split renvoie une erreur et laisse les 676 fichiers en place

- Si le fichier d'entrée est vide, aucun fichier n'est créé en sortie, et aucune erreur n'est renvoyée

```
rm file1 ; touch file1
split file1
```

split

- split

- split -a *LEN* les noms de fichiers de sortie auront un suffixe de longueur *LEN*
- split -b *NUM* découpe le fichier tous les *NUM* octets
  NUM k : découpe tous les 1024 octets
  NUM m : découpe tous les 1024 * 1024 o
- split -l *NUM* découpe le fichier toutes les *NUM* lignes

```
split -a 8 -l 42 file
split -b 38k file
```

csplit

- csplit

- Découpe un fichier en plusieurs selon des critères
- Tous les fichiers ne seront PAS de la même taille
- En cas d'erreur, tous les fichiers générés sont supprimés
- Les noms de fichiers en sortie sont de la forme :
  - Préfixe « xx » suivi de 2 chiffres
  - xx00, xx01, xx02, …

```
csplit file 3 10
csplit - 2 3
```

csplit

- csplit

- csplit -k — ne supprime pas les fichiers en cas d'erreur
- csplit -s — n'affiche pas la taille des fichiers créés
- csplit -f *PREFIX* — modifie le préfixe par *PREFIX* (« xx » est remplacé par *PREFIX*)
- csplit -n *LEN* — modifie la taille maximale du suffixe en *LEN* (par défaut, 2 chiffres sont utilisés)

```
csplit -k -s file 2 3
csplit -s -f "pref" -n 4 file 1 3
```

csplit

- csplit

- csplit [options] FILE ARG…

ARG peut contenir plusieurs types d'arguments :

- Nombres — copie jusqu'à la ligne en question, mais sans l'inclure (puis change de fichier)
- /regexp/[offset] — copie jusqu'à atteindre la regexp, mais sans inclure la ligne matchant la regexp (puis change de fichier)
- %regexp%[offset] — saute (ne copie pas) jusqu'à atteindre la regexp
- {Nombre} — répète « Nombre » fois la dernière opération
- {*} — répète aussi souvent que possible la dernière opération

csplit

- csplit

```
echo -e "AAH\nBEEH\nBla\nBlo\nBlu\nCEEH\n" > file2

csplit file2 /B/
csplit file2 /B/ {1}
csplit file2 /B/ {2}

csplit file2 %B%
csplit file2 %B% {1}
csplit file2 %B% {2}
```

csplit

- csplit

```
echo -e "Titre\nChap 1\nBla\nChap 2\nBla\n" > file3

csplit file3 /Chap/ {*}
csplit file3 %Chap% {*}
csplit file3 %Chap% {1}

echo -e "Titre\nChap 1\nBla\n  Chap 2\nBla\n" > file4
csplit file4 /^Chap/ {*}
csplit file4 /^Chap$/ {*}
csplit file4 /^Chap.*$/ {*}
```

diff

- diff (differences)

- Compare le contenu de deux fichiers, et affiche les différences

```
echo -e "Ligne 1\nLigne 2\nLigne 3\nLigne 4" > file1
cat file1 > file2
diff file1 file2

echo -e "Ligne 1   \nLignne 2\nLige 3\nLigne 4" > file2
diff file1 file2
```

diff

- diff

- diff -b transforme les espaces des fin de ligne en un seul ‘\n’
- diff -c extrait 3 lignes autour des modifications dans chaque fichier
- diff -C *NUM* extrait *NUM* lignes autour des modifications dans chaque fichier

```
diff -b file1 file2
echo -e "L1\nL2\nL3\nL4\nL5\nL6\nL7\nL8\nL9" > file3
echo -e "L1\nL2\nL3\nL4\nC5\nL6\nL7\nL8\nL9" > file4
diff -c file3 file4
```

diff

- diff

- diff -u extrait 3 lignes autour des modifications d'un fichier
- diff -U *NUM* extrait *NUM* lignes autour des modification du fichier de référence

```
diff -c file3 file4
diff -C 1 file3 file4
diff -u file3 file4
diff -U 1 file3 file4
```

diff

- diff

- diff -e affiche les modifications dans un format compréhensible par l'utilitaire ed
- diff -f affiche les modifications dans un format proche de -e, mais dont le résultat n'est pas exploitable par l'utilitaire ed (les commandes sont dans l'ordre inverse)

```
diff -e file3 file4
diff -f 1 file3 file4
```

diff

- diff

- diff -r effectue une comparaison récursive dans deux dossiers. Les fichiers de même nom dans les deux dossiers sont comparés, et les fichiers existant dans un seul des deux dossier sont nommés. Diff détecte les boucles infinies de dossiers (liens arrières).

```
mkdir dir1 dir2
touch dir1/f1 dir2/f1 dir1/f2 dir2/f2 dir2/f3
echo "test" > dir1/f2
diff -r dir1 dir2
```

ed

- ed (Editor)

- Éditeur de texte en ligne de commande (CLI)
  - Traitement ligne par ligne
- Pas d'interface … sauf une ligne de commande
- Mode commandes et mode insertion (comme vi)
- Pas adapté à l'édition par pipes, éditeur interactif !…
  - …supporte « non officiellement » les commandes par heredoc

```
ed file
```

ed

- ed (Editor)

1. On demande à passer en mode insertion
2. On écrit son texte
3. On finit le mode insertion en entrant une ligne ne contenant qu'un seul point
4. On effectue des commandes (sauvegarder, …)

```
ed file
```

ed

- ed (Editor)

- ed -s n’affiche pas le nombre d’octets écrits lors des modes e, E, r, w, ou !
- ed -p STR utilise STR comme prompt lors du mode commande (par défaut : aucun prompt)

```
ed -p "CMD>"
ed -p "CMD>"  -s file
```

ed

- ed (Editor)

- Mode commandes :

q quitte ed
Q quitte ed sans écrire les changements

w sauvegarde le fichier
w *FILE* sauvegarde dans le fichier *FILE*

ed

- ed (Editor)

- Mode commandes :

H inverse le mode d'affichage des erreurs (par défaut, aucun message n'est affiché)

h affiche les explications de la dernière erreur

u annule la dernière commande

!*CMD* exécute la commande *CMD* dans le shell courant

ed

- ed (Editor)

- Mode commandes :

f *FILE* met le nom par défaut des fichiers à *FILE*

e *FILE* édite le fichier nommé *FILE*, le « curseur » est placé en fin de fichier. Si aucun nom n'est donné le nom par défaut est utilisé

E *FILE* comme e, mais, les changements non écrits sont perdus sans demande à l'utilisateur

ed

- ed (Editor)

- Mode commandes :

.	ligne courante
1	aller à la ligne 1
$	aller à la dernière ligne
36	aller à la ligner 36
-2	revenir 2 ligne en arrière
+4	aller 4 lignes plus loin

ed

- ed (Editor)

- Mode commandes :

(.)i ajoute du texte (passage en mode insertion) avant la ligne courante. Une fois l'insertion terminée, le « curseur » est mis sur la dernière ligne insérée. (insertion)

(.)a ajoute du texte à partir de la ligne courante (append)

(.,.)c écrase la ligne (remplace la ligne par ce que l'on écrit)

(.,.)d supprime la/les lignes

ed

- ed (Editor)

- Mode commandes :

(.,.+1)j joint les adresses et les fusionne en une seule ligne. La ligne courante devient la ligne créée.

(.,.)p affiche les lignes sélectionnées (mode page par page si le contenu est trop long : appuyer sur enter pour avancer). Le « curseur » est mis sur la dernière ligne affichée.

(.,.)m(.) déplace les lignes vers l'adresse sélectionnée (après). Le « curseur » est mis à la dernière ligne écrite.

ed

- ed (Editor)

- Mode commandes :

(.,.)n affiche les lignes sélectionnées avec leur numéro de ligne. Place le « curseur » sur la dernière ligne affichée.

(.,.)l affiche les lignes sélectionnées de façon non-ambiguë (mode page par page si le contenu est trop long : appuyer sur enter pour avancer). Le « curseur » est mis sur la dernière ligne affichée.

ed

- ed (Editor)

- Mode insertion :

. finir l'insertion de texte (passage en mode commandes)

(comment insérer un point seul ? Faire substitution :)

a

x.

.

s/x//

ed

- Scénario 1 (on crée le fichier MyFile et on entre du texte) :

```
bash$ ed -p ">" MyFile

> a
Coucou
Ceci est un test
.
> w
> q
```

ed

- Scénario 2 (on édite le fichier MyFile, en y ajoutant du texte à la fin):

```
bash$ ed -p ">" MyFile

> a
Test suivant
.
> w
> q
```

ed

- Scénario 3 (on édite le fichier MyFile, et on regarde les lignes):

```
bash$ ed -p ">" MyFile

> p
Coucou
> 2p
Ceci est un test
> $
> p
Test suivant
>1,$n
```

ed

- Scénario 4 (on édite le fichier MyFile, et on ajout des lignes au milieu):

```
bash$ ed -p ">" MyFile

> $
> -1
> i
Ajout !
.
> wq
```

ed

- Scénario 5 (on édite le fichier MyFile, et on remplace une ligne):

```
bash$ ed -p ">" MyFile

> 2
Ajout !
> 2c
Nouvelle ligne
.
> wq
```

expr

- expr (Expressions)

- Évalue des expressions arithmétiques
- Évalue aussi des opérations sur des chaînes de caractères
- Le résultat est affiché sur la sortie standard

```
expr
expr 4 + 3
expr 6 \* 7
```

expr

- expr (Expressions)

- Mode arithmétique

Opérandes comprises (attention au shell qui interprète) :
`( ) | & = > >= < <= != + - * / % :`

```
expr 1 + 3                  expr 8 \* 3
expr 1 - 3                  expr 8 / 3
          expr 8 % 3
```

expr

- expr (Expressions)

- Mode arithmétique

Opérandes comprises (attention au shell qui interprète) :
`( ) | & = > >= < <= != + - * / % :`

```
expr 1 = 3                    expr 1 != 3
expr 1 \< 3                   expr 1 \<= 3
expr 1 \> 3                   expr 1 \>= 3
```

expr

- expr (Expressions)

- Mode arithmétique

Opérandes comprises (attention au shell qui interprète) :

```
( ) | & = > >= < <= != + - * / % :
```

```
expr \( 1 \< 3 \) \& \( 1 \> 3 \)
expr \( 1 \< 3 \) \| \( 1 \> 3 \)
```

expr

- expr (Expressions)

- Mode arithmétique

Ne fonctionnent PAS :

```
expr 1+2                (pas d'espace)
expr "1 + 2"            (chaîne de caractères)
expr 1 + (2 * 3)        (parenthèses pas échappées)
```

expr

- expr (Expressions)

- Mode chaîne de caractères

Pattern Matching : (match oui ou non)

```
expr "MaChaine" : "M.*C.*"
expr "MaChaine" : "M*C*"
expr "MaChaine" : "Z"
```

expr

- expr (Expressions)

- Mode « récent » (hors standard)

Longueur d'une chaîne :

```
expr length "MaChaine"
expr length "Ma Chaine"
```

expr

- expr (Expressions)

- Mode « récent » (hors standard)

Recherche de la 1ère occurrence d'une lettre :

```
expr index "MaChaine" "m"
expr index "Metalman" "m"
expr index "Metalman" "M"
expr index "Metalman" "man"
expr index "Metalman" "a"
```

expr

- expr (Expressions)

- Mode « récent » (hors standard)

Pattern matching : (matching & extraction)

```
expr match "46" "[0-9]*"
expr match "46" "\([0-9]*\)"

expr match "Chaine 46" "[^0-9]*[0-9]*"
expr match "Chaine 46" "[^0-9]*\([0-9]*\)"
```

expr

- expr (Expressions)

- Mode « récent » (hors standard)

Pattern matching : (matching & extraction)

```
expr match "Chaine 46" "[^0-9]*[0-9]*[^0-9]*"
expr match "Chaine 46" "[^0-9]*[0-9]*[^0-9]\+"

expr match "Chaine 46." "[^0-9]*[0-9]*[^0-9]*"
expr match "Chaine 46." "[^0-9]*[0-9]*[^0-9]\+"
```

sed

- sed (Stream Editor)

- Éditeur de texte par script (ou avec pipes)
  - Travaille avec des RegExp
- Écrit le résultat des modifications sur la sortie standard
- Travaille parfaitement bien avec Awk et autres outils de flux
- (Complémentaire à ed)

```
sed
echo "lolilol" | sed -e "s/lol/mdr/g"
```

sed

- sed (Stream Editor)

- sed -e *CMD* applique les commandes *CMD*
- sed -f *FILE* applique les commandes contenues dans le fichier *FILE*
- sed -n supprime la sortie par défaut. N'affiche que les lignes explicitement sélectionnées
- sed -E utilise les nouvelles RegExp
- sed -i écrit directement dans le fichier

```
sed –i "s/mto/mot/g" text_file.txt
```

sed

- sed (Stream Editor)

- Similaire à ed, peut travailler à des lignes (adresses) précises :

```
echo -e "ABC\nEFG\nIJK" | sed -ne "2,3p"
echo -e "ABC\nEFG\nIJK" | sed -e "2d"
echo -e "ABC\nEFG\nIJK" | sed -e "2w file.txt"
```

- Enchaînement de commandes (non ordonnées) :

```
echo -e "ABC\nEFG\nIJK" | sed -e "1h ; 3,4g"
echo -e "ABC\nEFG\nIJK" | sed -e "3p;1p;2p"
```

sed

- sed (Stream Editor)

Quelques commandes (cf le manuel pour plus d'infos) :

- *[adr]*p affiche la ligne (fonctionne bien avec l'option -ne) (P)
- *[adr]*d supprime la ligne (D)
- *[adr]*h copie la ligne dans le buffer « hold » (H)
- *[adr]*g colle depuis le buffer « hold » vers la ligne (G)
- *[adr]*x échange le contenu du buffer « hold » avec la ligne
- *[adr]*w *FILE* écrit la ligne dans le fichier *FILE*

sed

- sed (Stream Editor)
- */pattern/*
- Recherche le *pattern* et effectue la commande demandée

```
echo -e "ABC\nEFG\nIJK" | sed -e "/EF/ d"
```

sed

- sed (Stream Editor)

- **s/***pattern***/***replacement***/***flags*

Remplace le *pattern* par *replacement* selon les *flags* activés

Flags possibles :

- **g** transforme autant de fois que nécessaire (global)
- **p** écrit sur la sortie standard les modifications faites
- **w** *file* écrit le *replacement* dans *file* autant de fois qu'il a été écrit
- *n* (nombre) transforme l'occurrence *N* du *pattern* en *replacement*

sed

- sed (Stream Editor)

```
echo -e "ABC\nEFG\nIJK\nMNO\nQRS" > FILE1
echo -e "ABCE\nEFGE\nABCA" > FILE2

sed -e "s/EFG\|MNO/Test/p" FILE1
sed -ne "s/EFG\|MNO/Test/p" FILE1
sed -e "s/E/z/1" FILE2
sed -e "s/E/z/g" FILE2
sed -e "s/E/z/w out.txt" FILE2
```

sed

- sed (Stream Editor)

- **s/***pattern***/***replacement***/***flags*

Remplace le *pattern* par *replacement* selon les *flags* activés

Usages spéciaux :

- **&** remplace la chaîne reconnue par le *pattern*
- **\(\)** crée une *back-reference* sur cette partie du *pattern*
- ***n*** (nombre) rappelle la *back-reference* numéro *N*

sed

- sed (Stream Editor)

```
echo -e "ABC\nEFG\nIJK\nMNO\nQRS" > FILE1
echo -e "ABCE\nEFGE\nABCA" > FILE2

sed -e "s/E\|J/-&-/g" FILE1
```

On peut indiquer avant le “s” la ligne où l’on souhaite travailler :

```
sed -e "s/ABC/LOL/g" FILE2
sed -e "3s/ABC/LOL/g" FILE2
```

sed

- sed (Stream Editor)

```
echo -e "AxB\nDxC\nExB" > FILE3

sed -e "s/\(.\)x\(.\)/\1mul\2/g" FILE3

# pour mieux voir :
# "s / \( . \) x \( . \) / \1 mul \2 / g"
```

sed

- sed (Stream Editor)

- **y/***characters***/***replacement***/**

Remplace chaque *character* par son équivalent dans *replacement* (fonctionne exactement comme la commande *tr*)

```
echo -e "ABC\nEFG\nIJK" | sed -e "y/AFK/MDR/"
```

awk

- awk (Aho, Weinberger, Kernighan)

- Langage de traitement de fichiers par colonnes
  - Utile pour les extractions selon des séparateurs
- Interprète les fichiers comme des séquences d'enregistrements
  - Enregistrement/Record = ligne
  - Champ/Field = colonne
  - 1 Record = N Fields
- Gère séparateurs de lignes et colonnes différents (entrée/sortie)

`awk`

awk

- awk

- awk -F *IFS* change l'IFS (Input/Entrée)
- awk -v *ASSIGN* assigne des variables avant le lancement
- awk -f *FILE* lit un script awk dans le fichier *FILE*

(attention : le code doit être entre ' pour éviter que le shell interprète)

```
awk -F ":"   '{ print $3 }'   /etc/passwd
```

IFS changé pour « : » | code/script awk | fichier traité par awk

awk

- awk

Séparateur de sortie (OFS) matérialisé par une virgule dans le code

Quelques variables :

- $n colonne numéro N ($1, $2, … comme en shell)
- $0 l'enregistrement complet (la ligne complète)
- $NF dernière colonne

```
awk -F ":" '{ print $NF,$3,$1 }' /etc/passwd
```

awk

- awk

Quelques variables (cf le manuel pour plus d'infos) :

- FS / OFS (Input)/Output Field Separator
- RS / ORS (Input)/Output Record Separator
- NF Number of Fields (total pour cette ligne)
- NR Number of Records (numéro de ligne)

```
awk -F ":" '{ print NF,NR,$3,$1 }' /etc/passwd
awk -F ":" -v OFS=" V " '{ print $3,$1 }' /etc/passwd
awk -F ":" -v ORS="//" '{ print $3,$1 }' /etc/passwd
```

awk

- awk

Quelques fonctions (cf le manuel pour plus d'infos) :

- length(s) calcule la taille de la variable en paramètre
- tolower(s) passe le paramètre en minuscules
- toupper(s) passe le paramètre en majuscules
- index(s,t) donne la position dans *s* d'un caractère de *t*
- substr(s, m[, n]) découpe une chaîne dans *s* depuis la position *m* jusqu'à la fin ou jusqu'à *n*

```
awk -F ":" '{ print NR,$1,length($1) }' /etc/passwd
```

awk

- awk

Le langage gère : if, else, for, while, do, break, continue, function, return, exit, next, …

(fonction int(x) pour obtenir un entier)

```
echo '{ if ($1 == "root")
  { print "admin" }
  else { print $1 }
}'                                  > script_awk

awk -F ":" -f script_awk /etc/passwd
```

awk

- awk

```
echo '{
if ($1 == "root")
  { NewUID = 0;
    NewLogin = "admin"; }
else
  { NewUID = $3 + 42;
    NewLogin = $1; }
print "Utilisateur :",NewLogin,NewUID
}'                                > script_awk

awk -F ":" -f script_awk /etc/passwd
```

awk

- awk

Pattern matching : /str/

```
echo '/root/ { print "admin : "$0 }
      ! /root/ { print "user : "$0 }
'                                    > script_awk2

awk -F ":" -f script_awk2 /etc/passwd
```

awk

- awk

(Opérateurs : ~, !, ||, &&, … +, -, *, %, /, ++, --, <, >, <=, >=, …)

```
echo ' { if ($0 !~ /root/) { print "user : "$0 }
        else { print "admin : "$0 }
}'                                      > script_awk3

awk -F ":" -f script_awk3 /etc/passwd
```

awk

- awk

Pattern matching : /str/

```
echo '/ro[^0-9]*/ { print "Ro : "$0 }
      ! /ro[^0-9]*/ { print $0 }
'                                    > script_awk4

awk -F ":" -f script_awk4 /etc/passwd
```

last

- last

- Indique l’horaire de la dernière connection de l’utilisateur

```
last
last user
```

strings

- strings

- Extrait les chaînes de caractères imprimables de fichiers
- Utile pour visualiser rapidement les phrases imprimables d'un programme

```
strings /bin/ls
strings script.sh
strings prog.c
```

bc

- bc (Basic Calculator)

- Calculette de base
- Peut lire des scripts complets…
- …ou passer par des pipes

```
bc
echo "scale=2; 5 * 7 /3;" | bc
bc -l <<< "5*7/3"
```

ar

- ar (Archiver)

- Rassemble des fichiers en un seul…
- …utilisé aujourd'hui pour créer des bibliothèques (*libraries*) statiques

```
ar -cr libutil.a func1.o func2.o

ar -r archive.ar file1 file2 file3
ar -x archive.ar
```

ar

- ar

- ar -r ajoute/remplace des fichiers dans une archive
- ar -x extrait les fichiers d'une archive
- ar -t affiche le contenu de l'archive
- ar -v active le mode « verbose » (verbeux)

```
ar -rv my_archive.ar file1 file2
ar -t my_archive.ar
ar -xv my_archive.ar
```

ar

- ar

- ar -m déplace un fichier dans l'archive
- ar -a *POS* ajoute un fichier dans l'archive dans la position suivante du fichier nommé *POS*
- ar -b *POS* ajoute un fichier dans l'archive dans la position précédente du fichier nommé *POS*

```
ar -r -a file1 my_archive.ar file3
ar -m -a file2 my_archive.ar file3
ar -m -b file2 my_archive.ar file1
```

ar

- ar

- ar -u met a jour un fichier dans l'archive
  (si avec -r : met à jour si fichier différent)
- ar -d supprime le(s) fichier(s) de l'archive

```
ar -d my_archive.ar file3
ar -r -u my_archive.ar file2
```

ar

- ar
- ar -s regénère la table des symboles de l'archive

```
ar -cr libutil.a func1.o func2.o func3.o
ar -m -a func1.o libutil.a func3.o
ar -s libutil.a
```

tar

- tar (Tape Archiver)

- Créer une archive vers le lecteur de bande magnétique
  - Comportement par défaut… inutile aujourd'hui
- Créer une archive « pour » du matériel en accès séquentiel…
  - …utile pour réduire la taille au minimum
- L'archive est constituée de blocs
- Les archives peuvent être automatiquement compressées !

```
tar -cvf my_archive.tar file1 file2
tar -xvf my_archive.tar
```

tar

- tar (Tape Archiver)

- tar -f *NAME* accède au un fichier *NAME*, et non pas au lecteur de bandes de la machine
- tar -c écrit vers l'archive
- tar -v active le mode « verbose » (verbeux)
- tar -t affiche le contenu de l'archive

```
tar -cvf my_archive.tar file1 file2
tar -tf my_archive.tar
```

tar

- tar (Tape Archiver)

- tar -x extrait les fichiers d'une archive (détecte la compression si nécessaire)
- tar -z compresse l'archive avec gzip
- tar -j compresse l'archive avec bzip2

```
tar -xvf my_archive.tar
tar -cvzf my_archive.tar.gz file1 file2
tar -cvjf my_archive.tar.bz2 file1 file2
tar -xvf my_archive.tar.bz2
```

pax

- pax (Portable Archive Exchange)

- Archivage du standard POSIX (tar et cpio ont trop d'options non standards dans les différents UNIX)
- Modes « list » (~~r/w~~), « read » (r), « write » (w), « copy » (r/w)
- Créer des archives tar avec un attribut supplémentaire
- Peut lire des archives tar classiques

```
pax -w -f my_archive.tar file1 file2
pax -r -f my_archive.tar
```

pax

- pax (Portable Archive Exchange)

- pax -f travaille sur un fichier comme archive
  (si cette option est seule, elle affiche le contenu de l'archive)
- pax -r « lit » / extrait une archive (read)
- pax -w « écrit » / créer une archive (write)

```
pax -w -f my_archive.tar file1 file2
pax -f my_archive.tar
pax -r -f my_archive.tar
```

pax

- pax (Portable Archive Exchange)

- pax -x *FORMAT* désigne le format de l'archive
  cpio : archive au format cpio
  ustar : standard de tar
  pax : standard pax

```
pax -w -f archive.tar -x ustar file1 file2
pax -w -f archive.pax -x pax file1 file2
```

pax

- pax (Portable Archive Exchange)

- pax -p *OPTION* conserve ou non les droits des fichiers lors du mode « read »

  a : ne conserve pas l'heure d'accès au fichier

  m : ne conserve pas l'heure de modification

  o : conserve l'UID et GID

  p : conserve le sticky bit (SETUID, …), et d'autres

  e : conserve toutes les propriétés précédentes

```
pax -r -f archive.tar -p amo
```