Systèmes UNIX. Écosystème du Shell

Sergey Kirgizov

Découvrir et comprendre la multitude de commandes d'UNIX

Aujourd'hui

Littérature

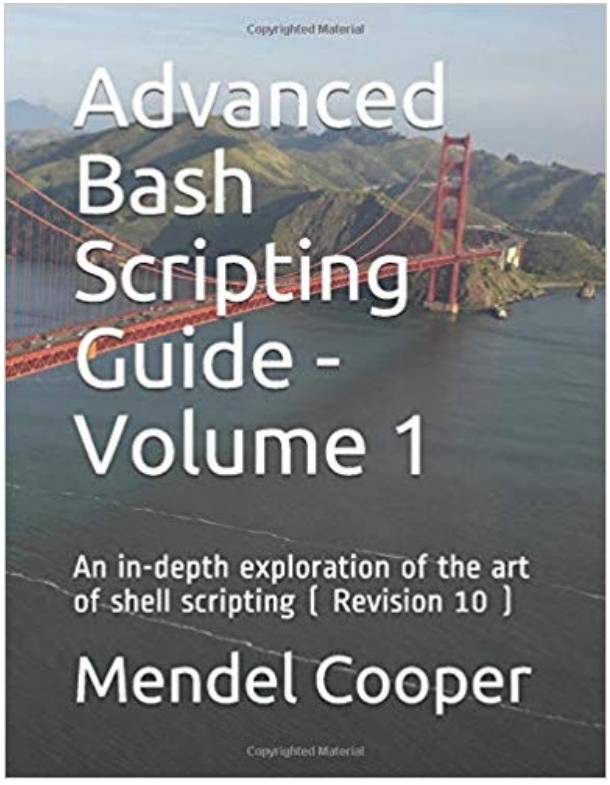
Multitude de commandes UNIX
- Variable d'environnement PATH
- Commandes et leurs paramètres
- Lecture de fichiers
- Connexion avec le monde extérieur
- Recherche
- Modifications
- Délégation

Flux des données

Redirections avancées

Littérature

Littérature : shell



- Advanced Bash-Scripting Guide par Mendel Cooper
  `https://www.tldp.org/LDP/abs/html/`

Littérature : unix man

Unix & Linux Commands Man Page Set	Commands
BSD 2.11	1,352
CentOS 7.0	38,339
Debian 7.7	81,313
FreeBSD 11.0	9,637
HP-UX 10.31 (mods only)	7,209
Linux 2.6	7,299
Minix 2.0	365
NetBSD 6.1.5	7,937
OSF1 5.1 (alpha)	6,070
OSX 10.14 Mojave	13,507
OSX 10.6.2	30,972
OpenDarwin 7.2.1	1,893
OpenSolaris 2009.06	16,086
POSIX 1003.1	3,514
Plan 9	380

`https://www.unix.com/man-page-repository.php`

Il y des mans en français, par exemple :

`http://jp.barralis.com/linux-man/`

Exemple de man

MAN(1) BSD General Commands Manual MAN(1)

NAME

man — display online manual documentation pages

SYNOPSIS

man [**-adho**] [**-t** | **-w**] [**-M** *manpath*] [**-P** *pager*] [**-S** *mansect*] [**-m** *arch*[:*machine*]]
[**-p** [*eprtv*]] [*mansect*] *page* ...
man **-f** *keyword* ...
man **-k** *keyword* ...

DESCRIPTION

The **man** utility finds and displays online manual documentation pages. If *mansect* is provided, **man** restricts the search to the specific section of the manual.

The sections of the manual are:

1. FreeBSD General Commands Manual
2. FreeBSD System Calls Manual
3. FreeBSD Library Functions Manual
4. FreeBSD Kernel Interfaces Manual
5. FreeBSD File Formats Manual
6. FreeBSD Games Manual
7. FreeBSD Miscellaneous Information Manual
8. FreeBSD System Manager's Manual
9. FreeBSD Kernel Developer's Manual

Options that **man** understands:

-M *manpath*
Forces a specific colon separated manual path instead of the default search path. See manpath(1). Overrides the MANPATH environment variable.

Littérature : info

man
C'est une petite brochure : bref, pratique, et très utile.

info
C'est un livre détaillant les commandes, leurs options et les concepts qui les accompagnent.

Comparer par exemple `man ln` et `info ln`.

info

```
File: dir,      Node: Top,      This is the top of the INFO tree.

This is the Info main menu (aka directory node).
A few useful Info commands:

  'q' quits;
  'H' lists all Info commands;
  'h' starts the Info tutorial;
  'mTexinfo RET' visits the Texinfo manual, etc.

* Menu:

Archiving
* Tar: (tar).                   Making tape (or disk) archives.

Basics
* Bash: (bash).                 The GNU Bourne-Again SHell.
* Common options: (inetutils)Common options.
                                Common options.
* Inetutils: (inetutils).       GNU networking utilities.
* Coreutils: (coreutils).       Core GNU (file, text, shell) utilities.
-----Info: (dir)Top, 2657 lines --Top-----------------------------------------
Welcome to Info version 6.7.  Type H for help, h for tutorial.
```

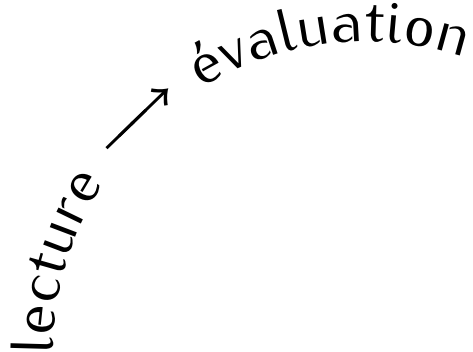
Multitude de commandes UNIX

REPL : read–eval–print loop

Shell d'UNIX vous permet de communiquer avec le système d'exploitation en boucle :

REPL : read–eval–print loop

Shell d'UNIX vous permet de communiquer avec le système d'exploitation en boucle :

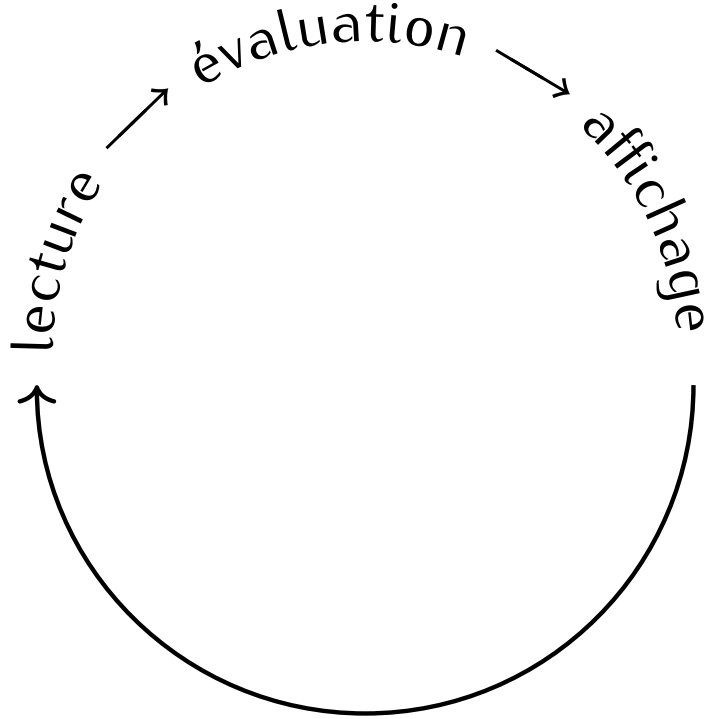


REPL : read–eval–print loop

Shell d'UNIX vous permet de communiquer avec le système d'exploitation en boucle :

REPL : read–eval–print loop

Shell d'UNIX vous permet de communiquer avec le système d'exploitation en boucle :



Où se trouvent les commandes unix ?

Où se trouvent les commandes unix ?

Dans des répertoires dont les noms sont contenus dans la variable d'environnement `$PATH`

Où se trouvent les commandes unix ?

Dans des répertoires dont les noms sont contenus dans la variable d'environnement `$PATH`

Exemple :

```
echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/bin/
```

Commandes et leurs paramètres

Commandes et leurs paramètres

Les *paramètres* sont séparés par des espaces, si vous voulez qu'un paramètre contient un espace, vous devez soit y échapper par un antislash "\", soit utiliser des apostrophes ou des guillemets.

Commandes et leurs paramètres

Les *paramètres* sont séparés par des espaces, si vous voulez qu'un paramètre contient un espace, vous devez soit y échapper par un antislash "\", soit utiliser des apostrophes ou des guillemets.

Les *options* sont des paramètres commençant par le signe moins "-".

- Les *options courtes* ont la forme de "-x" où "x" est un seul caractère.
- Les *options longues* ont la forme de "--bla-bla-bla".

Commandes et leurs paramètres

Les *paramètres* sont séparés par des espaces, si vous voulez qu'un paramètre contient un espace, vous devez soit y échapper par un antislash "\", soit utiliser des apostrophes ou des guillemets.

Les *options* sont des paramètres commençant par le signe moins "-".

- Les *options courtes* ont la forme de "-x" où "x" est un seul caractère.
- Les *options longues* ont la forme de "--bla-bla-bla".

Généralement, l'ordre des options n'a pas d'importance.

Commandes et leurs paramètres

Les *paramètres* sont séparés par des espaces, si vous voulez qu'un paramètre contient un espace, vous devez soit y échapper par un antislash "\", soit utiliser des apostrophes ou des guillemets.

Les *options* sont des paramètres commençant par le signe moins "-".

- Les *options courtes* ont la forme de "-x" où "x" est un seul caractère.
- Les *options longues* ont la forme de "--bla-bla-bla".

Généralement, l'ordre des options n'a pas d'importance.
Les options courtes peuvent être regroupées, par exemple, les trois commandes suivantes sont équivalentes

```
ls -l -a
ls -la
ls -al
```

Commandes et leurs paramètres

Exemple

```
boulangerie -a -b --foo-bar fichier1.txt "tram param"
```

- $0 : nom de la commande, 'boulangerie'
- $1 : une option courte, '-a'
- $2 : une autre option courte, '-b'
- $3 : une longe option '--foo-bar'
- $4 : un parameter, 'fichier1.txt'
- $5 : un autre parameter, 'tram param'

Lecture de fichiers

Lecture : cat

```
CAT(1)                                                          User Commands
                                                CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s) to standard output.

       With no FILE, or when FILE is -, read standard input.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines, overrides -n

       -e     equivalent to -vE

       -E, --show-ends
              display $ at end of each line

       -n, --number
              number all output lines
 Manual page cat(1) line 1/69 36% (press h for help or q to quit)
```

Lecture de premières lignes : head

```
HEAD(1)             User Commands             HEAD(1)

NAME
       head - output the first part of files

SYNOPSIS
       head [OPTION]... [FILE]...

DESCRIPTION
       Print the first 10 lines of each FILE to stan-
       dard output.  With more than one FILE, precede
       each with a header giving the file name.

       With no FILE, or when FILE is -, read standard
       input.

       Mandatory arguments to long options are manda-
       tory for short options too.

       -c, --bytes=[-]NUM
              print the first NUM bytes of each file;
              with the leading '-', print all but the
              last NUM bytes of each file

       -n, --lines=[-]NUM
              print  the  first  NUM lines instead of
              the first 10;  with  the  leading  '-',
              print  all  but  the  last NUM lines of
              each file

       -q, --quiet, --silent
              never print headers giving file names
```

Lecture de dernières lignes : tail

```
TAIL(1)                    User Commands                    TAIL(1)

NAME
       tail - output the last part of files

SYNOPSIS
       tail [OPTION]... [FILE]...

DESCRIPTION
       Print  the last 10 lines of each FILE to standard output.
       With more than one FILE, precede each with a header  giv-
       ing the file name.

       With no FILE, or when FILE is -, read standard input.

       Mandatory  arguments  to  long  options are mandatory for
       short options too.

       -c, --bytes=[+]NUM
              output the last NUM bytes; or use -c +NUM to  out-
              put starting with byte NUM of each file

       -f, --follow[={name|descriptor}]
              output appended data as the file grows;

              an absent option argument means 'descriptor'

       -F     same as --follow=name --retry

       -n, --lines=[+]NUM
              output the last NUM lines, instead of the last 10;
              or use -n +NUM to output starting with line NUM
```

Lecture inversée : tac

```
TAC(1)                          User Commands                          TAC(1)

NAME
       tac - concatenate and print files in reverse

SYNOPSIS
       tac [OPTION]... [FILE]...

DESCRIPTION
       Write each FILE to standard output, last line first.

       With no FILE, or when FILE is -, read standard input.

       Mandatory  arguments  to  long  options are mandatory for
       short options too.

       -b, --before
              attach the separator before instead of after

       -r, --regex
              interpret the separator as a regular expression

       -s, --separator=STRING
              use STRING as the separator instead of newline
```

wc : word count

Compter les lignes dans un fichier.

```
wc -l fichier
```

Compter les octets

```
wc -c fichier
```

Compter les mots

```
wc -w fichier
```

Affichage de gros fichiers

- **more** : affichage page par page
- **less** : affichage page par page et navigation

Connexion avec le monde extérieur

Les données publiques

`https://www.data.gouv.fr/fr/datasets/fichier-des-prenoms-depuis-1900/`

Communication avec les serveurs

Exemple, téléchargement :

```
wget https://www.insee.fr/fr/statistiques/fichier/7633685/nat2022_csv.zip
```

Autres commandes existent : `curl`, `telnet`, `netcat`, `ping`, `etc...`

Recherche

grep : rechercher des lignes contenant le motif spécifié

La commande **grep** permet de filtrer les lignes d'un fichier et de n'afficher que certaines d'entre elles.

Utile pour savoir si un fichier contient un mot donné. Des nombreuses autres applications existent.

Usage habituelle

```
grep motif fichier
```

grep : rechercher des lignes contenant le motif spécifié

La commande **grep** permet de filtrer les lignes d'un fichier et de n'afficher que certaines d'entre elles.

Utile pour savoir si un fichier contient un mot donné. Des nombreuses autres applications existent.

Usage habituelle

```
grep motif fichier
```

Quelques options

- option -v : afficher les lignes qui ne contiennent pas le motif
- option -n : numéroter les lignes résultat
- sans fichier en paramètre : utilisation de l'entrée standard
- option -e : pour préciser plusieurs motifs
- option -E : utiliser les expressions régulières étendues **(CM 6)**.
- option -o : afficher seulement le motif, pas toute la ligne

grep, exemple I

```
➜  ~ grep motif /usr/share/dict/words
leitmotif
leitmotif's
leitmotifs
motif
motif's
motifs
➜  ~
```

grep, exemple II

```
➜  ~ grep -n recurs /usr/share/dict/words
83967:precursor
83968:precursor's
83969:precursors
83970:precursory
89256:recurs
89257:recursion
89258:recursions
89259:recursive
89260:recursively
```

grep, exemple III

```
➜  ~ grep -e dada -e cubis /usr/share/dict/words
cubism
cubism's
cubist
cubist's
cubists
dadaism
dadaism's
dadaist
dadaist's
dadaists
➜  ~
```

Modifications

Éditeurs de text

- nano
- vi
- emacs
- ainsi que d'autres : ed, acme, sam, gedit, kate, joe, ...

sort : trier les lignes d'un fichier texte

Exemple 1 : trier dans l'ordre lexicographique croissant

```
sort fichier.txt
```

sort : trier les lignes d'un fichier texte

Exemple 1 : trier dans l'ordre lexicographique croissant

```
sort fichier.txt
```

Exemple 2 : trier dans l'ordre lexicographique décroissant

```
sort -r fichier.csv
```

sort : trier les lignes d'un fichier texte

Exemple 1 : trier dans l'ordre lexicographique croissant

```
sort fichier.txt
```

Exemple 2 : trier dans l'ordre lexicographique décroissant

```
sort -r fichier.csv
```

Exemple 3 : trier dans l'ordre numérique croissant

```
sort -n fichier.html
```

Exemple 4 : trier dans l'ordre numérique décroissant

```
sort -r -n fichier.xml
```

ou bien

```
sort --reverse --numeric-sort fichier.xml
sort -rn fichier.xml
sort -nr fichier.xml
```

uniq : supprimer les mêmes lignes qui se succèdent

fichier.txt :

```
foo
foo
bar
baz
baz
foo
```

uniq : supprimer les mêmes lignes qui se succèdent

fichier.txt :

```
foo
foo
bar
baz
baz
foo
```

Exemple 1 : `uniq fichier.txt`

```
foo
bar
baz
foo
```

uniq : supprimer les mêmes lignes qui se succèdent

fichier.txt :

```
foo
foo
bar
baz
baz
foo
```

Exemple 1 : `uniq fichier.txt`

```
foo
bar
baz
foo
```

Exemple 2 : `uniq -c fichier.txt`

```
2 foo
1 bar
2 baz
1 foo
```

diff et patch

diff : trouver des différences entre les fichiers

patch : appliquer le résultat de `diff` pour changer un fichier, c'est utile par exemple pour transférer des modifications de fichiers à une autre personne.

Les premières versions de diff ont été écrites par Douglas McIlroy et James W. Hunt. Voici l'article de recherche :
`https://www.cs.dartmouth.edu/~doug/diff.pdf`

En 1985, Larry Wall (créateur du langage Perl, linguiste de formation) a écrit un utilitaire patch.

Larry Wall

https://fr.wikipedia.org/wiki/Larry_Wall 160%

Les trois vertus du programmeur [modifier | modifier le code]

Larry Wall, avec Randal L. Schwartz et Tom Christiansen, dans la seconde édition de *Programming Perl*, a explicité les *Trois Vertus du Programmeur* :

« Les trois principales qualités du programmeur sont la paresse, l'impatience et l'orgueil » (Camel Book)

1. **Paresse** - La qualité qui vous pousse à faire de grands efforts pour réduire le total des dépenses d'énergie. C'est elle qui vous fait écrire des programmes qui font gagner du temps sans effort et que d'autres trouveront utiles, et c'est elle qui vous pousse à documenter ce que vous avez fait pour ne pas avoir à répondre à plein de questions.
2. **Impatience** - L'énervement que vous ressentez lorsque l'ordinateur est paresseux. L'impatience vous pousse à écrire des programmes qui ne répondent pas seulement à vos besoins, mais qui les anticipent même. Ou du moins qui font comme si.
3. **Orgueil** - C'est la qualité qui vous fait écrire et maintenir des programmes desquels personne ne voudra dire du mal.

sed : éditeur de flux pour filtrer et transformer le texte

Exemple 1 : remplacer TATA par TOTO dans chaque ligne du fichier

```
sed 's/TATA/TOTO/g' fichier
```

Exemple 2 : afficher les lignes de 2 à 6 du fichier

```
sed -n '2,6p' fichier
```

Exemple 3 : afficher des lignes paires

```
sed -n '0~2p' fichier
```

Exemple 4 : afficher chaque quatrième ligne commençant par la ligne 7

```
sed -n '7~4p' fichier
```

Sed est très puissant, je vous conseille fortement de lire la documentation : `man sed` **ou bien** `info sed`.

cut : supprimer une partie de chaque ligne d'un fichier

naissances :

```
Maria Paris 1977
Lora Dijon 2344
Margo Moscow 1877
Denis Londres 1987
Kevin New-York 1934
```

cut : supprimer une partie de chaque ligne d'un fichier

naissances :

```
Maria Paris 1977
Lora Dijon 2344
Margo Moscow 1877
Denis Londres 1987
Kevin New-York 1934
```

Exemple : `cut -d " " -f 1,3 naissances`

```
Maria 1977
Lora 2344
Margo 1877
Denis 1987
Kevin 1934
```

paste : regrouper les lignes de différents fichiers

fichierA

```
Maria
Lora
Margo
Denis
Kevin
```

fichierB

```
1977
2344
1877
1987
1934
```

paste : regrouper les lignes de différents fichiers

fichierA

```
Maria
Lora
Margo
Denis
Kevin
```

fichierB

```
1977
2344
1877
1987
1934
```

Exemple : `paste fichierB fichierA`

```
1977    Maria
2344    Lora
1877    Margo
1987    Denis
1934    Kevin
```

join : fusionner les lignes de deux fichiers ayant un champ commun

books.txt

```
Knuth - The Art of Computer Programming
Knuth - Mariages Stables
Knuth - Surreal Numbers
Tanenbaum - Operating Systems: Design and Implementation
Torvalds - Just for fun
```

authors.txt

```
Knuth Donald Ervin
Tanenbaum Andrew
Torvalds Linus
```

join : fusionner les lignes de deux fichiers ayant un champ commun

books.txt

```
Knuth - The Art of Computer Programming
Knuth - Mariages Stables
Knuth - Surreal Numbers
Tanenbaum - Operating Systems: Design and Implementation
Torvalds - Just for fun
```

authors.txt

```
Knuth Donald Ervin
Tanenbaum Andrew
Torvalds Linus
```

Exemple : `join authors.txt books.txt`

```
Knuth Donald Ervin - The Art of Computer Programming
Knuth Donald Ervin - Mariages Stables
Knuth Donald Ervin - Surreal Numbers
Tanenbaum Andrew - Operating Systems: Design and Implementation
Torvalds Linus - Just for fun
```

tr

tr : translation de caractères.
Changer une classe de caractères en une autre.

Exemple 1 : minuscules -> MAJUSCULES

```
cat fichier.txt | tr a-z A-Z
```

Exemple 2 : remplacer des espaces par des retours chariot

```
cat fichier.txt | tr " " "\n"
```

Délégation

xargs : construire et exécuter des lignes de commandes

authors.txt

```
Knuth Donald Ervin
Tanenbaum Andrew
Torvalds Linus
```

La commande

```
cat authors | xargs -I {} mkdir "{}"
```

va créer trois répertoires suivants 'Knuth Donald Ervin', 'Tanenbaum Andrew' et 'Torvalds Linus'.

Flux des données

Entrées sortie d'un processus

Flux d'entrée / sortie par défaut

- Au démarrage d'un processus et durant toute son exécution : trois flux associés par défaut
- permettent échanges/interactions avec l'utilisateur
- Identifiés par un nom ou un numéro :
  - stdin (0) : entrée standard
  - stdout (1) : sortie standard
  - stderr (2) : sortie erreur standard

Entrées sortie d'un processus

stdin (0) : standard input / entrée standard

- Permet de passer des informations vers le processus
- Principale interface de communication utilisateur
- Par défaut : clavier

stdout (1) : standard output / sortie standard

- Permet de passer des informations à l'utilisateur
- Affichage des messages standards
- Par défaut : écran

Entrées sortie d'un processus

stderr (2) : standard error / sortie erreur

- Dédiés aux messages d'erreur à destination de l'utilisateur
- Contient les messages relatifs à un comportement anormal (ex : accès refusé, fichier inexistant, ...)
- Par défaut : écran

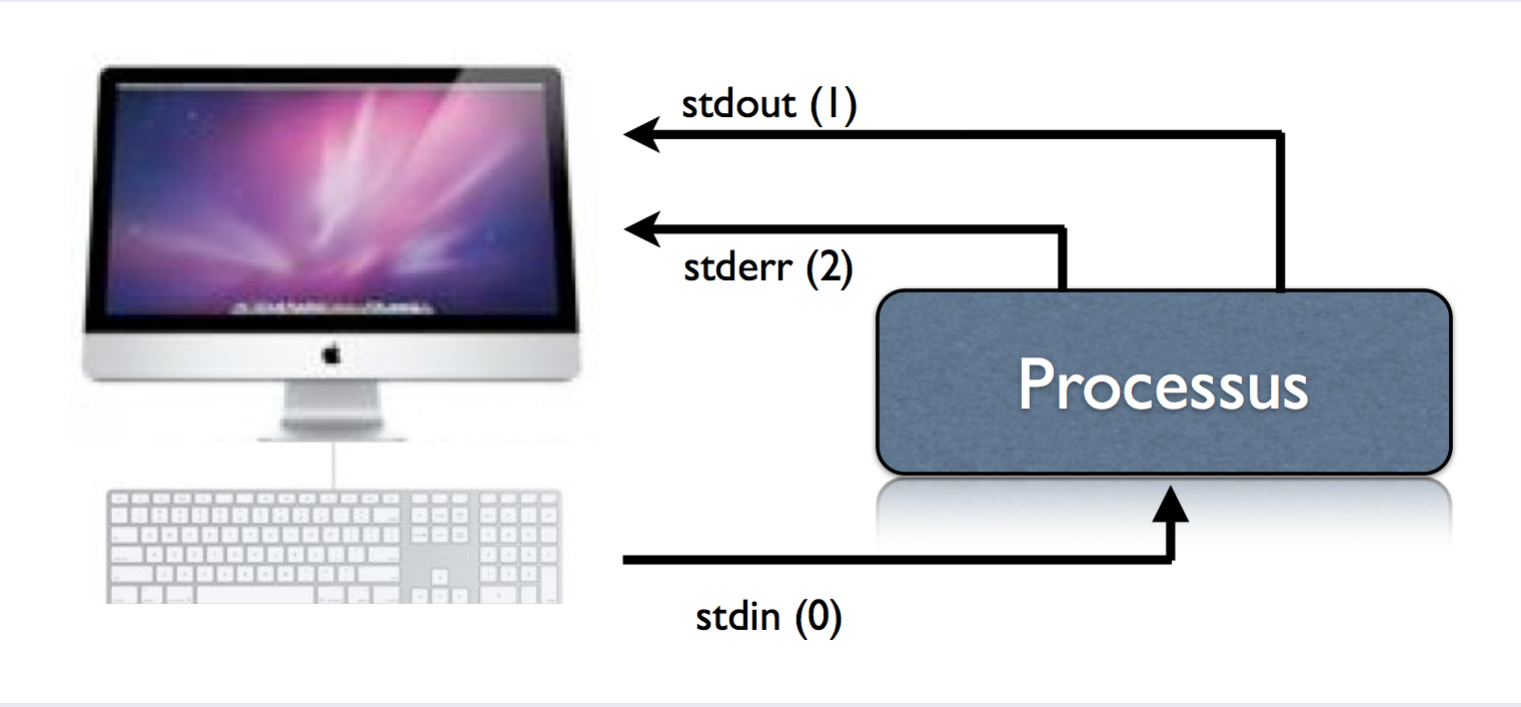
interaction utilisateur-shell

en dialogue standard avec le shell :

- L'utilisateur entre des lignes de commandes à faire exécuter au clavier
- Il reçoit le résultat de certaines commandes à l'écran
- En cas de problème, un message d'erreur est affiché à l'écran

Entrées sortie d'un processus

Schéma usuel (sans redirection des flux)



Flux d'entrée VS arguments

Attention : ne pas confondre flux d'entrée, et arguments / options passés en ligne de commande !

- Le flux d'entrée lu au clavier correspond aux données envoyées au processus lorsque celui ci est déjà actif
- Les arguments sont des éléments que l'on énumère sur la ligne de commande avant de lancer l'exécution du programme, et donc la création du processus
- Par exemple : on peut lancer l'exécution de `ls` ou `cd` en passant des arguments et des options, mais les processus ne lisent rien sur leur entrée standard.

Traitement sur fichier, ou sur flux d'entrée

fichier passés en arguments ou sur le flux d'entrée

- De nombreux programmes effectuent des opérations sur des données contenues dans des fichiers (ex : commandes `cat`, wc)
- Le nom du fichier doit être passé en argument
  Par exemple : `cat fichier.txt` ou `wc fichier.txt`
- Si aucun nom n'est passé en argument :
  - Les programmes considèrent que les données à traiter vont être passées par le flux stdin
  - Lecture de stdin comme s'il s'agissait d'un fichier
  - L'utilisateur saisit directement les données à traiter au clavier
  - Nécessité d'utiliser la combinaison de touches ctrl + d pour indiquer la fin du flux d'entrée à traiter.

Demo : Ecriture sur flux d'entrée avec le clavier

Saisie d'une commande travaillant usuellement sur un fichier

```
1 Galactica:fichiers benoit$ cat -n
```

- Le programme attends que du texte soit rentré
- Saisie de texte sur plusieurs lignes.
- A chaque retour à la ligne, la ligne est affiché avec le numéro de la ligne correspondante. [2]

```
1 hello
2      1  hello
3 Ceci est un simple test
4      2  Ceci est un simple test
5 pour illustrer l'ecriture sur un flux d'entree
6      3  pour illustrer l'ecriture sur un flux d'entree
```

- Fin de saisie avec `ctrl + d` pour arrêter l'écriture.

2. cat -n n'a besoin que d'une lecture ligne à ligne. S'il avait eu besoin de lire tout le fichier avant, il aurait attendu que toutes les données aient été entrées

Redirection des flux d'entrée/sortie par défaut

Redirection de flux pour combiner la puissance des outils UNIX

- Une des principales forces d'UNIX
- Possibilité de rediriger les flux depuis une autre source ou vers une autre destination (selon leur nature).
- Communications inter-processus
- Lecture du flux d'entrée depuis un fichier
- Ecriture du flux de sortie standard et/ou erreur vers un fichier
- Utilisation d'un flux de sortie d'un processus comme flux d'entrée d'un autre processus
- Duplication / redéfinition de flux
- ...

Opérateur de redirection de flux vers /depuis un fichier

Utilisation :

```
Programme (opérateur) fichier
```

Liste des opérateurs de redirection vers / depuis un fichier

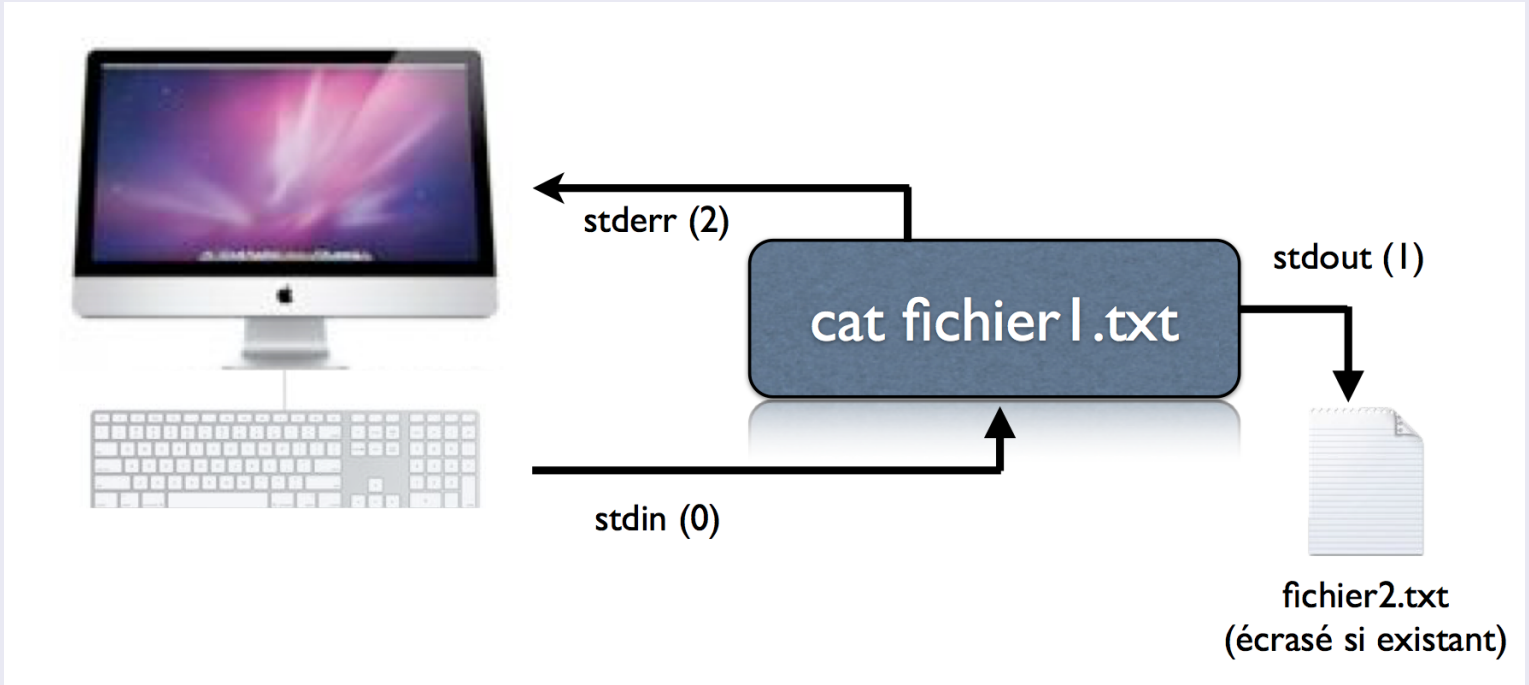
>	redirection de stdout dans un fichier. Le contenu du fichier est écrasé.
>>	redirection de stdout dans un fichier. Le flux est rajouté à la suite du fichier si le fichier n'était pas vide.
2 >	redirection de stderr dans un fichier. Le contenu du fichier est écrasé.
2 >>	redirection de stderr dans un fichier. Le flux est rajouté à la suite si le fichier n'était pas vide.
<	lecture de stdin à partir d'un fichier.

Exemples de redirection d'un flux de sortie

Analyse de la commande suivante :

```
1 cat fichier1.txt > fichier2.txt
```

Conséquence sur la redirection des flux



`fichier2.txt` sera une copie parfaite de `fichier1.txt`

Exemples de redirection d'un flux de sortie

Redirection de flux successives avec concaténation :

```
1  $ echo "bonjour" > fichierResult
2  $ echo "nous somme le " >> fichierResult
3  $ date >> fichierResult
4
5  $ cat fichierResult
6  bonjour
7  nous somme le
8  Mar 18 aoÃ» 2015 23:04:45 CEST
```

Sur les secondes et troisièmes instructions, le résultat de chaque commande est ajouté à `fichierResult`

Rediriger le flux de sortie d'un processus vers le flux d'entrée d'un autre

Problème :

Comment numéroter les lignes d'un fichier `fichier.txt`, et enregistrer seulement les lignes comprises entre 16 et 20 dans un fichier de sortie ?

Philosophie UNIX : Diviser pour régner

Idée : découper l'opération en 3 sous-opérations, utiliser à chaque étape le résultat de l'étape précédente comme donnée d'entrée

1. Un premier programme `cat -n` numérote les lignes du fichier
2. Un second programme `head` affiche les 20 premières lignes numérotées du fichier
3. Un troisième programme `tail` affiche les 5 dernières lignes de ces 20 premières lignes

Solution 1 (mauvaise) : passer par des fichiers temporaires

```
Programme1 > fichierTemporaire
Programme2 < fichierTemporaire
```

Utilisation d'un fichier temporaire comme buffer

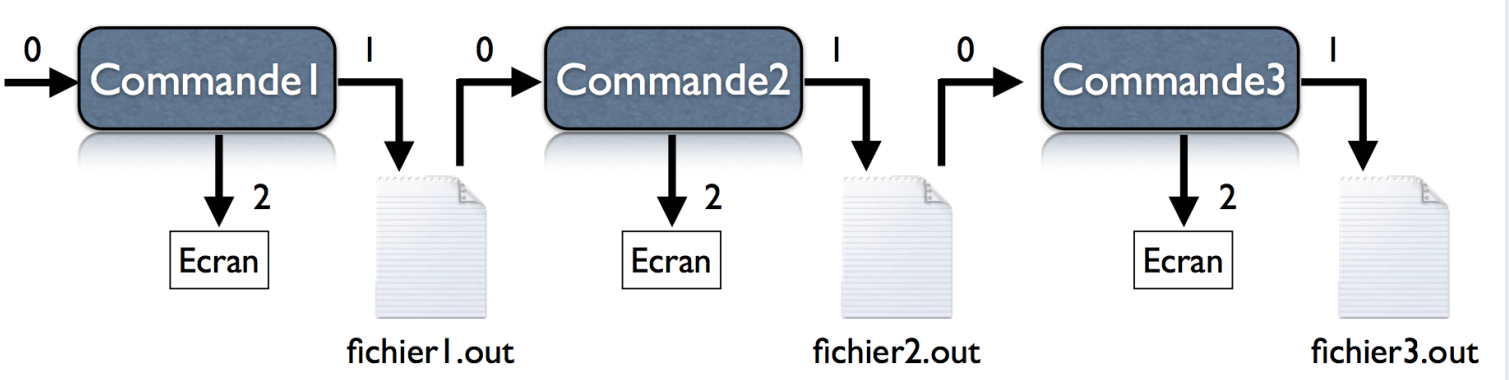
- Enregistre le flux de sortie de Programme1, et l'utilise comme flux d'entrée pour Programme2
- Fonctionne, mais n'est pas efficace
- Il faut attendre que l'exécution de Programme1 soit terminée pour lancer celle de Programme2.

Solution 1 (mauvaise) : passer par des fichiers temporaires

Analyse de la suite de commandes suivante :

```
1 commande1 > fichier1.out
2 commande2 < fichier1.out > fichier2.out
3 commande3 < fichier2.out > fichier3.out
```

Conséquence sur la redirection des flux



Numérotation des lignes de fichier, et enregistrement du résultat dans fichierTemp1

```
1 cat -n fichier > fichierTemp1
```

Récupération des 20 premieres lignes de fichierTemp1. Enregistrement dans fichierTemp2

```
1 head -n 20 < fichierTemp1 > fichierTemp2
```

Récupération des 5 dernières lignes de fichierTemp2 et affichage

```
1 Tail -n 5 < fichierTemp2
```

Le résultat correspond bien aux lignes 16 à 20 de `fichier`

Solution 2 (bonne) : utiliser les tubes

Programme1 | Programme2

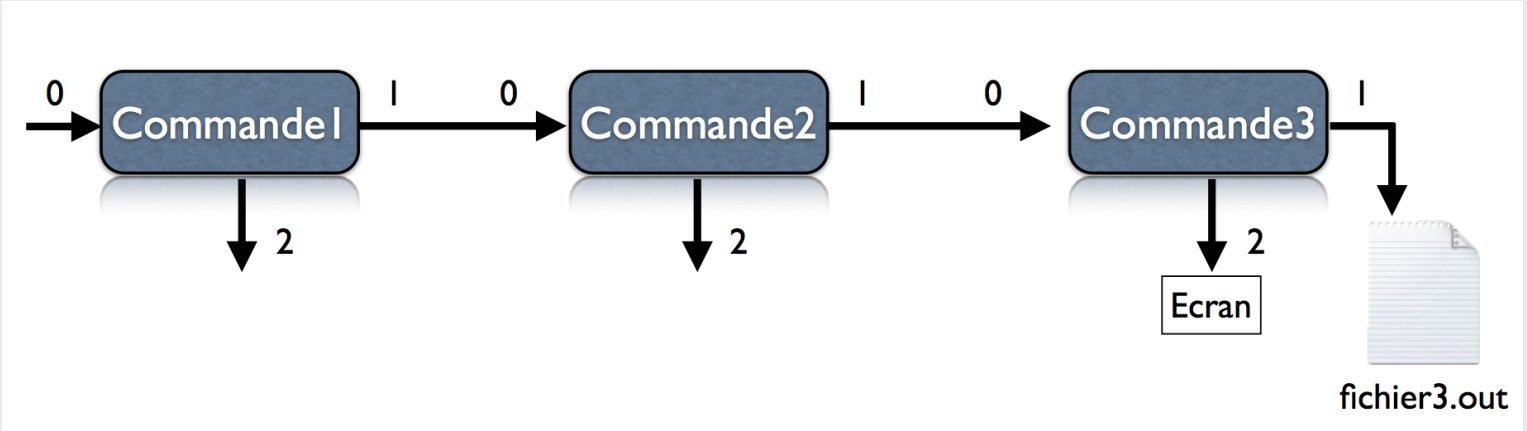
- Redirige le flux de sortie standard de Programme1 vers l'entrée standard de Programme2
- Programmes exécutés en parallèle
- Synchronisation assurée par UNIX

Solution 2 (bonne) : utiliser les tubes

Analyse de la commandes suivante :

```
1  commande1 | commande2 | commande3  > fichier3.out
```

Conséquence sur la redirection des flux



Problème :

Comment numéroter les lignes d'un fichier `fichier.txt`, et enregistrer seulement les lignes comprises entre 16 et 20 dans un fichier de sortie ?

Solution au problème

```
cat -n fichier.txt | head -n 20 | tail -n 5 > fichierSortie.txt
```

Notion de filtre

Les filtres sont des commandes qui peuvent traiter des données lues sur l'entrée standard, et renvoyer le résultat sur la sortie standard
Généralement, on peut préciser un nom de fichier pour remplacer l'entrée standard cat, head, tail, sort, uniq sont des exemples de filtres ls, cd, pwd mkdir sont des exemples de commandes qui ne sont pas des filtres
L'utilisation de tubes n'a évidemment d'intérêt que si la commande positionnée à droite d'un tube est un filtre

Tubes, pipes, pipelines

Le concept a été inventé par Malcolm Douglas McIlroy.
La notation “|” a été inventé par Ken Thompson.

`http://cs.bell-labs.co/who/ken/`
`https://www.cs.dartmouth.edu/~doug/`
`https://en.wikipedia.org/wiki/Pipeline_(Unix)`

Redirections avancées

Descripteur de fichier

0 — `/dev/stdin`, `/proc/PID/fd/0`
1 — `/dev/stdout`, `/proc/PID/fd/1`
2 — `/dev/stderr`, `/proc/PID/fd/2`

Pour les autres fichiers auxquels notre programme aura accès, le système va créer les autres descripteurs, par exemple :

3 — `/proc/PID/fd/3`

Guide avancé d'écriture des scripts Bash
`https://abs.traduc.org/abs-5.3-fr/ch19.html`

Collage des entrées-sorties

`i>&j`

Redirige le descripteur de fichier i vers j. Toute sortie vers le fichier pointé par i est envoyée au fichier pointé par j.

`2>&1`

Redirige stderr vers stdout. Les messages d'erreur sont envoyés à la même place que la sortie standard.

Exemples à tester chez vous

- `cat fichier-existant | less`
- `cat fichier-non-existant | less`
- `cat fichier-non-existant 2>&1 | less`

Substitution de paramètres

Substituer un paramètre par un résultat d'un autre commande

- `command1 $(command2 arg1 arg2 ...)`
- `command1 `command2 arg1 arg2 ...``

Plus qu'un flux

`<(command2 arg ...)` sera remplacé par le nom d'un fichier temporaire contenant la sortie standard de "`command arg ...`"

Exemple

```
command1 <(command2 arg ...) <(command 4 arg ...)
```

Merci Benoît !

Ce cours est en partie fondé sur le cours Benoît Darties.
`https://benoit.darties.fr/`

Questions ?