Introduction aux systèmes d'exploitation - M2 CCI

Language de commandes

Nicolas Louvet (Univ. Lyon 1)

d'après un cours pompé sur Laure Gonnord
qui avait pompé elle-même sur Julien Forget

1. L'interpréteur de commandes
   - Commandes pour l'arborescence de fichier
   - Notions sur les I/O
   - Vers les scripts

2. Quelques commandes utiles
   - Le retour des fichiers
   - Accès distant
   - Les filtres

Langages de commandes

Objectif : Apprendre à se servir d'un langage de commandes (Shell) :

- Utilisation de la ligne de commande ;
- Commandes classiques ;
- Scripts (dans la partie suivante).

Rôle de l'interpréteur de commandes

Interface (textuelle) entre l'utilisateur et le système :

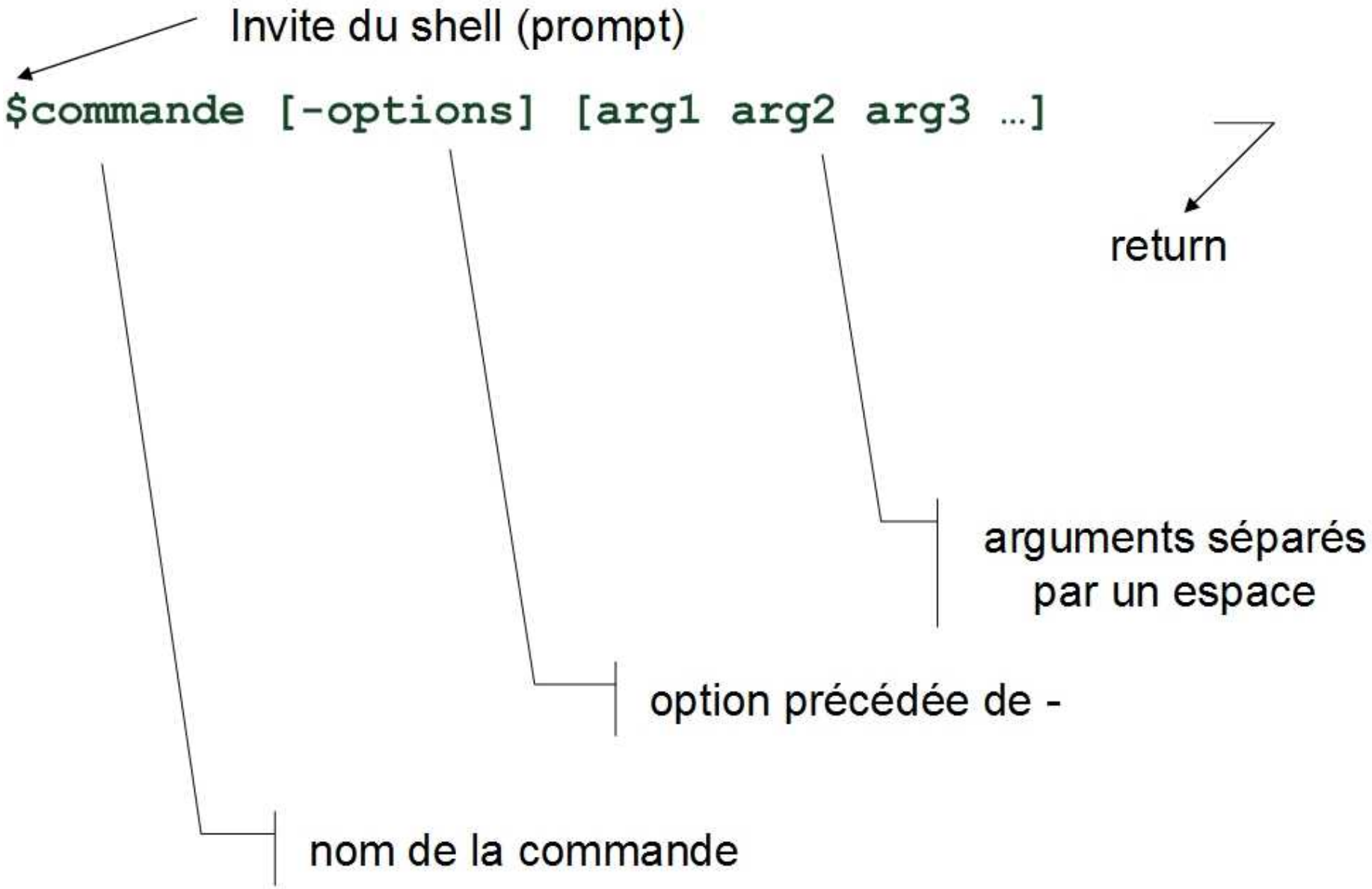
- Exécution de programmes et de commandes ;
- Contrôle de l'environnement ;
- Redirection des entrées/sorties ;
- Gestion de variables ;
- Traitement des caractères spéciaux ;
- Fournit un langage de programmation (script shell).

Choix d'un interpréteur de commandes

- Bourne Shell et dérivés (UNIX) :
  - `sh` : bourne shell (shell original)
  - `bash` : bourne again shell
  - `ksh` : korn shell
  - `zsh` : Z shell
- C Shell et dérivés (UNIX) :
  - `csh` : C shell, développé par Berkeley
  - `tcsh` : C shell amélioré
- Microsoft :
  - DOS : `command.com`
  - Windows : `cmd.exe`.

Format d'une commande

Format général d'une commande shell :



Quelques exemples avant de passer à la suite

```
laure@meert:~/Documents/1314work/enseignement/SystemeCCI$ ls
Cours  plan_indicatif.txt  ProgSyst  SystJF  TP

laure@meert:~/Documents/1314work/enseignement/SystemeCCI$ echo 'coucou'
coucou

laure@meert:~/Documents/1314work/enseignement/SystemeCCI$ cd TP

laure@meert:~/Documents/1314work/enseignement/SystemeCCI/TP$ ls *.pdf
hierarchie.pdf  tp1.pdf  tp2.pdf  tp3-4.pdf
```

Les caractères spéciaux du shell

Chemins :

- `~` désigne le répertoire `home` de l'utilisateur
- `.` désigne le répertoire courant.
- `..` désigne le répertoire parent.

Expressions : les jokers ou wildcards

- `?` 1 caractère quelconque
- `*` 0, un ou plusieurs caractères
- `[ ]` caractère parmi un ensemble
- `[^ ]` caractère en dehors d'un ensemble

Les jokers en shell : exemple

Gestion des jokers/wildcards sous UNIX :

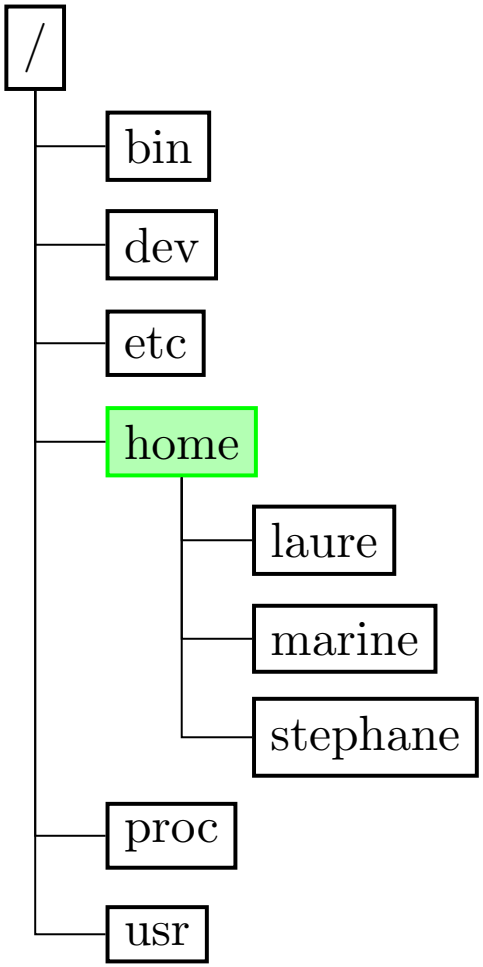
- l'expression est d'abord étendue par le shell,
- puis la commande est lancée sur le résultat.

```
laure@meert:~/$ ls /usr/bin/z*
/usr/bin/zdump                /usr/bin/zipcloak   /usr/bin/zipsplit
/usr/bin/zeitgeist-daemon     /usr/bin/zipgrep    /usr/bin/zsoelim
/usr/bin/zenity               /usr/bin/zipinfo    /usr/bin/zxpdf
/usr/bin/zip                  /usr/bin/zipnote
laure@meert:~/$ ls /usr/bin/z[a-g]*
/usr/bin/zdump  /usr/bin/zeitgeist-daemon  /usr/bin/zenity
laure@meert:~/$ ls /usr/bin/z[^a-g]*
/usr/bin/zip        /usr/bin/zipgrep    /usr/bin/zipnote
/usr/bin/zsoelim    /usr/bin/zipcloak   /usr/bin/zipinfo
/usr/bin/zipsplit   /usr/bin/zxpdf
```

Exo : Qu'affiche la commande suivante ?

```
laure@meert:~/tmp$ ls /usr/bin/z[a-g]???[a-z]
```

Arborescence Linux typique



- / : la racine
- `/bin` : commandes système
- `/etc` : configuration
- `/home` : répertoires perso ("home") des utilisateurs

▶ Le reste sera découvert plus tard.

S'informer, se déplacer dans l'arborescence

- Afficher des renseignements sur les fichers du répertoire courant
  - `ls [-al]`

    option -a : ne pas ignorer les entrées débutant par .
    option -l : utiliser le format long d'affichage

- Afficher des renseignements sur des *fichiers spécifiques*
  - `ls [-al] <fich1> <fich2> ... <fichn>`

- Afficher des renseignements des *répertoires spécifiques*
  - `ls -d <rep1> <rep2> ... <repn>`

- Afficher des renseignements sur le *contenu* de répertoires
  - `ls <rep1> <rep2> ... <repn>`

- Changer de répertoire courant
  - `cd <rep>`

Création/renommage de fichier/répertoire

- Renommer un fichier
  - `mv <fich1> <fich2>`
- Création de répertoire(s)
  - `mkdir [-p] <rep>[/<rep>]`

    option -p : crée les répertoires parents si besoin
- Renommer un répertoire
  - `mv <rep1> <rep2>`

Déplacement de fichiers

- UNIX : `mv [-ifu] <fic1> ... <fic2>`
  - option -i (*interactive*) : demander confirmation avant d'écraser
  - option -u (*update*) : écrase si le fichier source est plus récent que la cible
  - option -f (*force*) : ne pas demander de confirmation avant d'écraser

Copie

- Copie de plusieurs fichiers vers un répertoire
  - `cp [-ifu] <fich1> ... <fichn> <rep>`
- Copie récursive de répertoire
  - `cp [-ifu] -r <rep1> ... <repn> <rep>`
- Dans les deux cas
  - option -i (*interactive*) : demander confirmation avant d'écraser
  - option -f (*force*) : ne pas demander de confirmation avant d'écraser
  - option -u (*update*) : écrase si le fichier source est plus récent que la cible

Suppression répertoires / de fichiers

- Destruction de répertoires
  - `rmdir [-if] -r <rep1> ... <repn>`
  - `rm -rf <rep1> ... <repn>`

- UNIX : `rm [-irf] <fich1> ... <fichn>`
  - -i : confirmation avant destruction
  - -r : récursif
  - -f : force la destruction des fichiers sans autorisation d'écriture

Suppression de fichier : exemple

- UNIX : `rm [-if] <fich1> ... <fichn>`

```
laure@meert:~/tmp$ ls -ail
total 16
3548989 drwxrwxr-x 2 laure laure 4096 Aug  19 15:17 .
3545532 drwxrwxr-x 4 laure laure 4096 Aug  19 15:15 ..
3549043 -rw-rw-r-- 1 laure laure    7 Aug  19 15:16 add.txt
3549044 -rw-rw-r-- 1 laure laure  172 Aug  19 15:17 atagada.txt
laure@meert:~/tmp$ rm -i add.txt
rm: remove regular file 'add.txt'? n
laure@meert:~/tmp$ rm -if add.txt
laure@meert:~/tmp$ ls
atagada.txt
laure@meert:~/tmp$
```

Déplacement / suppression de rep : exemple

```
laure@meert:~/tmp2$ mkdir diressai
laure@meert:~/tmp2$ mkdir diressai
mkdir: cannot create directory ‘diressai’: File exists
laure@meert:~/tmp2$ ls
afile.txt  diressai
laure@meert:~/tmp2$ ls -l
total 8
-rw-rw-r-- 1 laure laure    7 août  20 09:34 afile.txt
drwxrwxr-x 2 laure laure 4096 août  20 09:34 diressai
laure@meert:~/tmp2$ ls diressai
laure@meert:~/tmp2$ mv afile.txt diressai
laure@meert:~/tmp2$ ls diressai
afile.txt
laure@meert:~/tmp2$ rmdir diressai
rmdir: failed to remove ‘diressai’: Directory not empty
laure@meert:~/tmp2$ rm -rf diressai
laure@meert:~/tmp2$ ls
```

Flux d'entrée/sortie

Sous UNIX un processus est créé avec 3 *flux* ($\simeq$ canaux de communications) :

- *Entrée standard* (stdin, numéro 0) : lecture des données d'entrée. Utilisé par exemple par `scanf` en C ;
- *Sortie standard* (stdout, numéro 1) : écriture des données de sortie. Utilisé par exemple par `printf` en C ;
- *Sortie erreur* (stderr, numéro 2) : écriture des données d'erreur. Utilisé par exemple par `perror` en C.

Flux d'entrée/sortie - exemples

Démos :

- stdout avec la commande `ls`
- stdin/stdout avec la commande `cat`
- stderr avec la commande `ls` (avec un répertoire inexistant)

```
laure@meert:~/tmp$ ls /bin/x*
```

Redirection des entrées/sorties

Il est possible de changer la destination/source des flux d'entrée/sortie :

sh	effet
>	redirige `stdout` vers un fichier
<	redirige un fichier vers `stdin`
2 >	redirige `stderr` vers un fichier
>>	redirige `stdin` en fin d'un fichier
2 >>	redirige `stderr` en fin de fichier
> &	redirige `stdout` et `stderr`
2 > &1	redirige `stderr` sur `stdout`

Redirection : exemple

```
laure@meert:~/tmp$ ls /bin/y* > bin_ycommands
laure@meert:~/tmp$ cat bin_ycommands
/bin/ypdomainname
laure@meert:~/tmp$ ls /bin/x* > bin_xcommands
ls: cannot access /bin/x*: No such file or directory
laure@meert:~/tmp$ ls /bin/x* 2> bin_xcommands
laure@meert:~/tmp$ cat bin_xcommands
ls: cannot access /bin/x*: No such file or directory
laure@meert:~/tmp$ ls /bin/k* >> bin_ycommands
laure@meert:~/tmp$ cat bin_ycommands
/bin/ypdomainname
/bin/kbd_mode
/bin/keyctl
/bin/kill
```

Séquence de commandes

- *Séquence simple*, le " ;"

```
laure@meert:~/tmp$ echo "Commandes en k :" ; ls /bin/k*
Commandes en k :
/bin/kbd_mode
/bin/keyctl
/bin/kill
```

- *Séquence avec redirection*, le "|" (pipe). La sortie d'une commande est redirigée vers l'entrée de la suivante

```
laure@meert:~/tmp$ ls /bin/k* | wc -l
3
```

(`wc -l` compte le nombre de lignes d'un texte)

- Possible d'enchaîner plusieurs séquences :
  `ls -la /usr/bin | sort | more`

Les variables

- Affectation :
  - `<var>=<valeur>`
- Désigner la valeur d'une variable :
  - `${<var>}` ou `$<var>`
- Affecter une variable d'environnement :
  - `export <var>=<valeur>`

Les variables d'environnement

Quelques variables d'environnement Unix :

- `HOME` répertoire racine de l'utilisateur
- `SHELL` nom du shell utilisé par l'utilisateur
- `PATH` répertoire des commandes
- `UID, USER` identité de l'utilisateur
- `MANPATH` chemin pour trouver les pages du manuel
- `PS1, PS2` invites utilisées par le shell
- `...`

Démos

```
laure@meert:~/tmp$ a=3 ; echo $a
3
```

```
laure@meert:~/tmp$ echo $USER
laure
```

(export sera vu en tp)

La variable `PATH`

Exécution d'une commande en UNIX :

- Recherche dans les répertoires spécifiés par la variable `PATH`.
- Si commande trouvée, alors exécution ;
- Si la commande n'est pas dans un répertoire du `PATH`, alors on doit spécifier le chemin où elle se trouve.
  Ex : `/chemin/commande` ou `./commande`

Démo PATH

```
laure@meert:~/$ echo $PATH
/home/laure/bin:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/home/laure/bin:
/usr/local/java/jdk1.7.0_09/bin:/home/laure/bin
```

(sans les sauts de ligne)

Autres fonctionnalités notoires du shell

- Complétion :
  - Appuyer sur ⇆ (tabulation) pendant la saisie d'une commande ;
  - Appuyer deux fois si plusieurs complétions possibles.

- Historique :
  - ↑ remonter, ↓ redescendre dans l'historique ;
  - L'historique des commandes
    - ★ peut être affiché avec la commande `history`
    - ★ `!n`+ Entrée lance la commande `n` de l'historique
  - Chercher une commande dans l'historique :
    - ★ Ctrl + R <chaine> : une commande contenant <chaine> ;
    - ★ A nouveau Ctrl + R : cherche la commande précédente ;
    - ★ Entrée : exécute la commande sélectionnée ;
    - ★ Ctrl + G : interrompt la recherche.

Plan

1. L'interpréteur de commandes
   - Commandes pour l'arborescence de fichier
   - Notions sur les I/O
   - Vers les scripts

2. Quelques commandes utiles
   - Le retour des fichiers
   - Accès distant
   - Les filtres

Manuel d'une commande

- `man [section] <commande>`
  - Section 1 : commandes utilisateur
  - Section 2 : appels système
  - Section 3 : fonctions diverses
  - Section 4 : format de fichiers
  - Section 5 : divers

  Pour sortir de `man` utiliser

- `whatis <mot>`.
  - recherche des pages de manuel dont le nom correspond

- `apropos <mot>`.
  - recherche les pages de manuel en rapport avec un sujet donné

- Le man décrit (entre autre) :
  - La manière d'appeler la commande (Synopsis) ;
  - Le rôle de la commande et toutes ses options.

Vous ne connaissez pas une commande ? Utilisez le man ! !

Plan

1. L'interpréteur de commandes
   - Commandes pour l'arborescence de fichier
   - Notions sur les I/O
   - Vers les scripts

2. Quelques commandes utiles
   - Le retour des fichiers
   - Accès distant
   - Les filtres

Affichage

- Afficher le contenu d'un fichier : `cat <fich>`
- Afficher page par page : `more <fich>`
  espace pour changer de page, Q pour quitter
- Afficher page par page : `less <fich>`
  ↑ , ↓ pour se déplacer, Q pour quitter
- Afficher le contenu de plusieurs fichiers : `cat <fich1> ... <fichn>`
- Afficher un texte : `echo <chaîne>`

touch

`touch [-option] [-r référence | -t date] fichier`

- modifie la date d'accès et la date de modification d'un fichier
  - option `-r` : utilise la date d'un fichier de référence
  - option `-t` : date à spécifier
- les fichiers n'existant pas sont créés, leur contenu est vide
  $\Rightarrow$ souvent utilisé pour créer des fichiers vides

```
laure@meert:~$ ls -al touch_essai
ls: cannot access touch_essai: No such file or directory
laure@meert:~$ touch touch_essai
laure@meert:~$ ls -la touch_essai
-rw-rw-r-- 1 laure laure 0 Aug  19 14:53 touch_essai
```

Saisie d'un fichier au clavier

- Utilisation des redirections :
  `cat > <fich>`
  `<texte frappé au clavier>`
  Ctrl + D
- Ou encore : `echo "chaine" > <fich>`
  pour créer un fichier ne contenant que la chaine spécifiée

▶ Doit être réservé à des cas particuliers ; on n'écrit pas de programme comme cela, on utilise un éditeur de texte.

Manipulation de nom de fichier

- `basename` permet d'éliminer le chemin d'accès et le suffixe d'un nom de fichier :

```
basename <nom_de_fichier> [ suffixe ]
```

- `dirname` permet d'éliminer le dernier composant délimité par un slash du nom de fichier :

```
dirname <nom_de_fichier>
```

```
laure@meert:~/tmp$ basename /usr/include/stdio.h
stdio.h

laure@meert:~/tmp$ basename /usr/include/stdio.h .h
stdio

laure@meert:~/tmp$ dirname /usr/include/stdio.h
/usr/include
```

find

`find [chemin] [expression...]`

- rechercher des fichiers dans une hiérarchie de répertoires
- tout plein autres options...
- retenir : `find [chemin] -name "fichier"` (pour chercher un fichier)

```
laure@meert:~$ find /etc/ -type d -group root -print 2> /dev/null | tail
    -n 5
/etc/network/if-post-down.d
/etc/network/if-down.d
/etc/rc6.d
/etc/ssh
/etc/sudoers.d

laure@meert:~$ find . -mtime 0 -type f
./.procmail/log
./irclogs/freenode/#lautre.net.log
./.lbdb/m_inmail.list
./.lbdb/m_inmail.list.dirty
./.spamassassin/bayes_toks
./.spamassassin/bayes_seen
./MailLaure/.mbox/cur/1376994860.30985_0.meert:2,S
```

Plan

1. L'interpréteur de commandes
   - Commandes pour l'arborescence de fichier
   - Notions sur les I/O
   - Vers les scripts
2. Quelques commandes utiles
   - Le retour des fichiers
   - Accès distant
   - Les filtres

ssh (*remote login*)

`ssh [-l username] host`

- login sur un ordinateur hôte (distant) de *manière sécurisée*

```
laure@meert:~$ ssh euler.local
The authenticity of host 'euler.local (192.168.1.43)' can't be
    established.
RSA key fingerprint is df:54:06:55:f2:86:48:ef:ac:34:26:72:69:87:65:85.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'euler.local,192.168.1.43' (RSA) to the list
    of known hosts.
laure@euler.local's password:
laure@euler:~$ exit
logout
Connection to euler.local closed.
laure@meert:~$ ssh euler.local
laure@euler.local's password:
Welcome to Ubuntu 12.04.2 LTS
[...]
Last login: Tue Aug 20 09:42:26 2013 from meert.local
laure@euler:~$
```

Plan

1. L'interpréteur de commandes
   - Commandes pour l'arborescence de fichier
   - Notions sur les I/O
   - Vers les scripts

2. Quelques commandes utiles
   - Le retour des fichiers
   - Accès distant
   - Les filtres

Principe

Le principe :

- lecture d'une entrée (texte) ;
- filtrage (supprimer des lignes, des colonnes, remplacer des mots, etc.) ;
- production d'une sortie (texte) ;
- souvent utilisés en série à l'aide du "|".

Très utile pour automatiser des traitements textuels répétitifs.

tail

`tail [-number | -n number] [filename...]`

↪ donne les `number` dernières lignes d'un fichier

```
laure@meert:~/tail -n 5 /var/log/cups/access_log
192.168.1.43 - - [20/Aug/2013:12:48:51 +0200] "POST /printers/
    SamsungNoireLyon HTTP/1.1" 200 278 Validate-Job successful-ok
192.168.1.43 - - [20/Aug/2013:12:48:51 +0200] "POST /printers/
    SamsungNoireLyon HTTP/1.1" 200 278 Create-Job successful-ok
192.168.1.43 - - [20/Aug/2013:12:48:51 +0200] "POST /printers/
    SamsungNoireLyon HTTP/1.1" 200 593519 Send-Document successful-ok
localhost - - [20/Aug/2013:14:08:35 +0200] "POST /printers/
    SamsungNoireLyon HTTP/1.1" 200 320 Create-Job successful-ok
localhost - - [20/Aug/2013:14:08:35 +0200] "POST /printers/
    SamsungNoireLyon HTTP/1.1" 200 296 Send-Document successful-ok
```

head

`head [-number | -n number] [filename...]`

↪ donne les `number` premières lignes du fichier. Par défaut, `n` = 10

```
laure@meert:~$ head -n 3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

wc

`wc [options] [<fichier> ...]`

↪ Compte le nombre de lignes, de mots et de caractères des fichiers.

- option `-w` : nombre de mots
- option `-l` : nombre de lignes
- option `-c` : nombre de caractères

```
laure@meert:~$ wc /etc/passwd
  36   51 1675 /etc/passwd
laure@meert:~$ wc -l /etc/pass*
  36 /etc/passwd
  37 /etc/passwd-
  73 total
```

cut

`cut -f<liste> [-d<caractère>]`

↪ Permet de découper les lignes d'un fichier en champs et d'extraire des champs particuliers.

- option `-f` : numéro des champs à reporter
- option `-d` : caractère délimitant les champs

```
laure@meert:~$ cat colonnes.txt
Du:texte:en:colonnes
Et:sur:deux:lignes
laure@meert:~$ cut -f2,3 -d: colonnes.txt
texte:en
sur:deux
```

grep

`grep [options] <expression> [<fichier>]`

- Permet de sélectionner des lignes contenant un texte ou un motif (cf. expressions régulières) ;
- Quelques options (voir `man` pour d'autres options) :
  - -i pas de différence entre majuscule et minuscule
  - -c compte le nombre de lignes
  - -v inverse le résultat

Attention, les expressions utilisées comme argument de grep/egrep ont une syntaxe différente des regexp du shell (voir plus tard).

(`egrep` ou `egrep -E` permettent des expressions régulières plus détaillées)

Des exemples

```
laure@meert:~$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
laure:x:1000:1000:laure,,,:/home/laure:/bin/bash
stephane:x:1001:1001:Stephane Gonnord,,,:/home/stephane:/bin/bash
laure@meert:~$ grep -c bash /etc/passwd
3
laure@meert:~$ grep -c BASH /etc/passwd
0
laure@meert:~$ grep -ci BASH /etc/passwd
3
laure@meert:~$ grep -civ BASH /etc/passwd
33
laure@meert:~$ wc -l /etc/passwd
36 /etc/passwd
```

```
laure@meert:~$ grep "\(ba\|sa\)" /etc/passwd
root:x:0:0:root:/root:/bin/bash
backup:x:34:34:backup:/var/backups:/bin/sh
messagebus:x:102:105::/var/run/dbus:/bin/false
laure:x:1000:1000:laure,,,:/home/laure:/bin/bash
saned:x:113:122::/home/saned:/bin/false
stephane:x:1001:1001:Stephane Gonnord,,,:/home/stephane:/bin/bash
debian-spamd:x:115:124::/var/lib/spamassassin:/bin/sh
```

tr

`tr [option] set1 [set2]`

↪ transpose ou supprime des caractères

```
laure@meert:~$ a="essai"
laure@meert:~$ echo "$a : tagada"
essai : tagada
laure@meert:~$ echo "$a : tagada" | tr a-z A-Z
ESSAI : TAGADA
laure@meert:~$ echo "$a : tagada" | tr [:lower:] [:upper:]
ESSAI : TAGADA
laure@meert:~$ echo "$a :                blu" | tr -s " "
essai : blu
```

sort

`sort [options] [fichier]`

- `sort` trie les lignes d'un fichier texte
- principales options :
  - -n numérique
  - -k champ
  - -t séparateur
  - -r reverse

```
laure@meert:~$ cat /etc/passwd | sort -t: -k3nr | head -n 5
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
stephane:x:1001:1001:Stephane Gonnord,,,:/home/stephane:/bin/bash
laure:x:1000:1000:laure,,,:/home/laure:/bin/bash
spampd:x:116:125::/nonexistent:/bin/false
debian-spamd:x:115:124::/var/lib/spamassassin:/bin/sh
```

join

join [options] fichier1 fichier2

↪ affiche sur la sortie standard une ligne pour chaque paire de lignes, l'une provenant de fichier1 et l'autre de fichier2, qui disposent de champs de jointure.

```
laure@meert:~$
sudo cat /etc/shadow | grep sy
sys:*:15818:0:99999:7:::
sync:*:15818:0:99999:7:::
syslog:*:15818:0:99999:7:::
```

```
laure@meert:~$ cat /etc/passwd | grep
      sy
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
syslog:x:101:103::/home/syslog:/bin/
     false
```

```
laure@meert:~$ join -j 1 -t: /etc/passwd /etc/shadow | grep sy
sys:x:3:3:sys:/dev:/bin/sh:*:15818:0:99999:7:::
sync:x:4:65534:sync:/bin:/bin/sync:*:15818:0:99999:7:::
syslog:x:101:103::/home/syslog:/bin/false:*:15818:0:99999:7:::
```

Expressions régulières (regexp)

- Permet de désigner des ensembles de chaînes de caractères sous la forme de "motifs généraux"
- Utilisées par de nombreuses commandes UNIX, en particulier les filtres : `egrep`, `sed`, ...
- Les jokers sont des regexp traitées directement par le Shell. D'autres commandes traitent directement leurs regexp : les syntaxes varient :-(

Expressions régulières

Exemples :

- . caractérise n'importe quel caractère
- [ ] un des caractères entre crochets ou si le premier caractère est ^, alors caractérise ceux qui ne correspondent pas avec ceux entre crochets
  - [abc] a, b ou c
  - [a-z] une lettre minuscule
  - [a-d5-8w-z] a, b, c, d, 5, 6, 7, 8, w, x, y, z
  - [^0-9] pas un chiffre
  - [^a-zA-Z] pas une lettre
- ^ début de ligne
- $ fin de ligne ; par exemple ab$ est une ligne finissant par ab

Expressions régulières

Exemples :

- * signifie de 0 à n fois le caractère qui précède
- a* 0 à n fois a
- aa* au moins un a
- .* n'importe quelle chaîne de caractères (même vide)
- ^[0-9][0-9]*$ ligne qui ne contient que des chiffres
- \(\) isoler des sous-chaînes. On peut les réutiliser grâce à \1 \2

Filtre-éditeur sed

- Commande *très* puissante, à la fois un filtre et un éditeur.

- Utilisation courante :
  - recevoir, en entrée chaque ligne d'un fichier ;
  - lui faire subir des modifications ;
  - l'envoyer sur la sortie standard.

- Forme la plus utilisée : `sed -e 's/regexp/replacement/' file`

⇒ remplace `regexp` par `replacement` pour chaque ligne de `file`

```
laure@meert:~/tmp$ cat colonnes.txt
Du:texte:en:colonnes
Et:sur:deux:lignes
laure@meert:~/tmp$ sed 's/nes/nneess/' colonnes.txt
Du:texte:en:colonnneess
Et:sur:deux:lignneess
```

Filtre-éditeur sed

Quelques remarques :

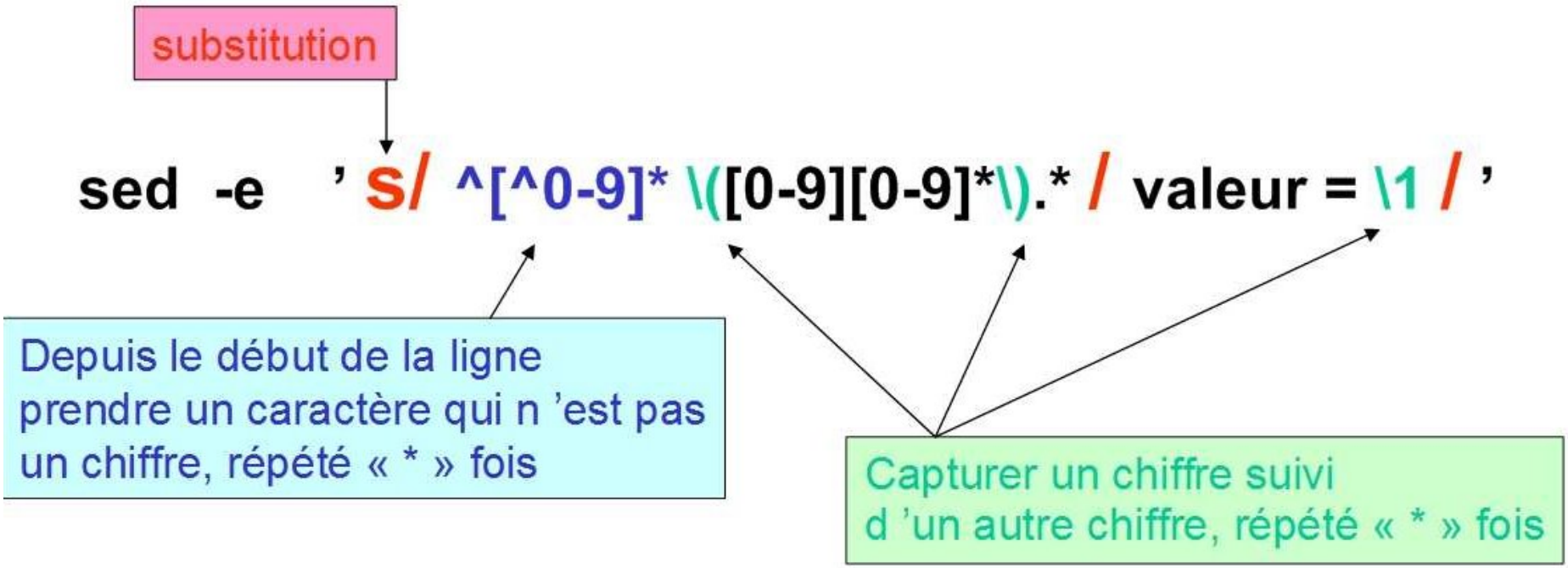
- sed cherche toujours le plus grand motif correspondant à une regexp ;
- Pour faire plusieurs remplacements sur une même ligne, ajouter l'option g

```
laure@meert:~/tmp$ cat colonnes.txt
Du:texte:en:colonnes
Et:sur:deux:lignes
laure@meert:~/tmp$ sed 's/n.*s//' colonnes.txt
Du:texte:e
Et:sur:deux:lig
laure@meert:~/tmp$ sed 's/te/tt/' colonnes.txt
Du:ttxte:en:colonnes
Et:sur:deux:lignes
laure@meert:~/tmp$ sed 's/te/tt/g' colonnes.txt
Du:ttxtt:en:colonnes
Et:sur:deux:lignes
```

Filtre-éditeur sed

Capture de groupe dans regexp :

```
laure@meert:~$ echo "ceci est un poids : 676Kg" | sed -e 's
    /^[^0-9]*\([0-9][0-9]*\).*/Valeur=\1/'
Valeur=676
```



Filtre-éditeur sed et autres commandes

Combinaison de filtres :

```
laure@meert:~$ cat /etc/passwd | grep Gonnord
laure:x:1000:1000:Laure Gonnord,,,:/home/laure:/bin/bash
stephane:x:1001:1001:Stephane Gonnord,,,:/home/stephane:/bin/bash
laure@meert:~$ cat /etc/passwd | grep Gonnord | cut -f7 -d:
/bin/bash
/bin/bash

laure@meert:~$ cat /etc/passwd | grep Gonnord | cut -f7 -d: | sed -e 's
    /\//*/'
*bin/bash
*bin/bash
laure@meert:~$ cat /etc/passwd | grep Gonnord | cut -f7 -d: | sed -e 's
    /\//*/g' *bin*bash
*bin*bash
```