

# Méthodes Empiriques et Langages de Script

## TP 1

Gabriele A. Musillo

[musillo4@etu.unige.ch](mailto:musillo4@etu.unige.ch)

October 24, 2006

### **Abstract**

GNU/Linux, shells, commandes, fichiers, wildcards, redirections, tubes et redirections, filtres, expressions régulières, gestion des processus et des jobs.

## Les systèmes GNU/LINUX

- freeware et opensource
- multitâche et multithread préemptifs
- multiutilisateur
- sécurisé
- GNU: réseaux (apache, . . . ), bases de données (mysql, . . . ), programmation (gcc, perl, . . . ), éditeurs (emacs, xemacs, vi, . . . ), etc  
...

## Shells: b(ourne)a(again)sh(ell)

- interaction au moyen d'un *shell*, i.e. un *interpréteur de commandes*
- les shells sont programmables:

```
$ for F in `ls`; do mv $F ${F}.old; done
```

- les shells sont éditables:

```
$ history
```

```
$ !!
```

```
$ !3141
```

```
$ !-3
```

```
$ !hist
```

```
$ !?ory
```

```
$ ^history^ls
```

## Commandes

- Une commande consiste en:
  - un identifieur de commande (une commande *built-in* du shell ou un programme)
  - quelques options
  - quelques arguments
- Exemples:

```
$ pwd
```

```
$ ls
```

```
$ ls -a1F
```

```
$ man man
```

```
$ man -wa perl
```

## Fichiers

- les fichiers et les répertoires sont les noeuds d'une structure hiérarchique arborescente dont la racine est /
- quelques commandes utiles:
  - `cd répertoire`  
change de répertoire courant
  - `mkdir [-m -p] répertoires`  
créé un ou plusieurs répertoires
  - `rmdir [-p] répertoires`  
détruit un ou plusieurs répertoires *vides*
  - `touch fichiers`  
créé des fichiers ou rafraîchit leur date d'accès ou de modification
  - `rm [-f -i -r] fichiers`  
détruit un ou plusieurs fichiers de la hiérarchie
  - `cp [-f -i -p -r] fichiers fichier ou répertoire`  
copie les fichiers
  - `mv [-f -i] source cible`  
déplace des fichiers ou des répertoires

## Les caractères *wildcard* du bash

- les *wildcard patterns* sont des séquences de caractères spéciales au moyen desquelles le shell bash génère des noms de fichiers
- E.g.:

```
$ ls file*
```

```
$ ls file[1-9]
```

```
$ ls file[!1-9]# équivalent à ls file[^1-9]
```

```
$ ls file[!123]# équivalent à ls file[^123]
```

```
$ ls file??
```

```
$ cd /; rm -rf *
```

```
$ ls -adlF ~/mels/dir[123]
```

```
$ ls -alF ~/mels/dir[345]/file?
```

## Redirections

- tout processus consomme des données provenant de son flot d'entrée, son *stdin*, et produit un flot de sortie, son *stdout* ou d'erreurs, son *stderr*
- par défaut, le clavier attaché au terminal alimente le flot d'entrée
- par défaut, le terminal affiche les flots de sortie et d'erreur
- les flots *stdin*, *stdout* et *stderr* peuvent être redirigés vers des fichiers
- le descripteur 0 désigne le *stdin*, le 1 désigne le *stdout* et le 2 désigne le *stderr*
- E.g:

```
$ ls file[1-9] > ls.out
```

```
$ ls file1 eifl > ls.out 2> ls.err
```

```
$ ls file1 eifl > ls.out_and_err 2>&1
```

```
$ ls file1 eifl >> ls.out 2>&1
```

```
$ ls file1 eifl 2>&1 > ls.out
```

## Redirections et Tubes

- le *stdout* ou le *stderr* d'un processus peuvent être redirigés sur le flot *stdin* d'un autre processus au moyen d'un tube qu'on note "|" et qu'on obtient en pressant Alt Gr-1 ou Ctrl-Alt-1
- les processus liés au moyen de tubes forment un *pipeline*
- logiquement, le pipeline

$$cmd_m \mid cmd_n$$

est équivalent à la séquence

$$cmd_m > out_m; cmd_n < out_m$$

- E.g.:

```
$ ls -i file* | cut -f 1 | sort -nu
```

```
$ ls -i eifl file1 2>&1 | tr '[a-z]' '[A-Z]'
```

```
$ head -n 7 txt | tail -n 1
```

```
$ head -n 9 | tail -n 7
```

## Filtres

- quelques filtres utiles:
  - `cat [-n] fichiers`  
affiche les fichiers concaténés
  - `cut [-d -f] fichiers`  
affiche quelques champs d'un ou plusieurs fichiers
  - `paste [-d 'n'] fichiers`  
colle les lignes de plusieurs fichiers
  - `head [-n n] fichier`  
affiche les  $n$  premières lignes d'un fichier
  - `tail [-n n] fichier`  
affiche les  $n$  dernières lignes d'un fichier
  - `tr [-d -s] stringsrc stringtrg`  
translittère
  - `wc [-c -l -w] fichiers`  
compte les caractères, les lignes et les mots d'un fichier
  - `join [-j n] fichier1 fichier2`  
joint 2 fichiers qui ont en commun leur  $n$ -ième champ

## Expressions régulières

- les expressions *régulières* sont des séquences de caractères spéciales au moyen desquelles des langages de programmation tels que Perl ou Python et des commandes telles que `egrep` ou `sed` filtrent les lignes d'un fichier régulier
- plus formellement, elles définissent un langage et acceptent les lignes qui appartiennent au langage défini
- E.g.:

```
$ echo 'ceci est une ligne.' | grep -E '.*est.*'
```

```
$ echo 'blablabla !' | grep -E 'bla(bla)*.*'
```

```
$ echo 'et' | grep -E '(et)|(ou)'
```

```
$ echo 'non ou si alors' | grep -E '(et)|(ou)'
```

```
$ echo 'si p alors q' | grep -E 'si p (alors )?q'
```

```
$ echo 'si p q' | grep -E 'si p (alors )?q'
```

```
$ echo 'ce langage' | grep -E '.*l.*g'
```

```
$ echo 'ce langage' | grep -Eo '.*l.*g'
```

```
$ echo 'ce langage' | grep -Eo '.*l[^g]*g'
```

## Wildcard patterns et expressions régulières

- les wildcard patterns et les expressions régulières ne sont pas équivalentes:

wcpat		regex
*	→	.*
?	→	.
[a-z]	→	[a-z]
[! a-z]	→	[^ a-z]
	↯	a*
	↯	(00) (11)

## Gestion des Processus

- un programme en exécution est un processus
- quelques attributs importants d'un processus:
  - **PID** : dénote un entier qui identifie le processus
  - **UID** : dénote l'utilisateur qui a démarré le processus
- quelques commandes indispensables à la gestion des processus:
  - `ps`: affiche sur la sortie standard les programmes en cours d'exécution
  - `pstree`: affiche la structure hiérarchique des processus
  - `top`: affiche en temps réel l'état des processus et les ressources qu'ils consomment
  - `kill`: envoie un signal à un processus identifié par son pid; ce signal peut être un signal de terminaison, un signal d'interruption, etc...
  - `killall`: envoie un signal à un processus identifié par la commande qui l'a lancé

## Gestion des Jobs

- un processus lancé en arrière-plan est un job:
  - le caractère `&` qui suit une commande lance celle-ci dans l'arrière-plan (e.g. `emacs &`, `xeyes &`)
  - la commande `jobs -l` liste les jobs ainsi que leurs PID respectifs
  - les commandes `bg` et `fg` permettent, respectivement, de placer un processus en arrière-plan et au premier plan