TP 01 : La ligne de commande sous Linux

Introduction: Les deux premiers TP du cours Architecture et Système font à la fois office d'introduction, et de révision des connaissances de base du système d'exploitation GNU/Linux que vous utiliserez tout au long de votre cursus. Toutes les notions abordées ici seront considérées comme acquises et maitrisées pour tous les cours à venir quelque soit le module.
Ce premier TP vous invite à (re)découvir la ligne de commande (shell).
Les postes informatiques de la salle 411 fonctionnent sous Xubuntu Xenial 16.04. Il s'agit d'un système d'exploitation GNU/Linux (noyau Linux et outils GNU) basé sur la distribution Ubuntu (une «Debian like») utilisant l'environnement de bureau (Desktop Environment) Xfce.
La connexion à votre session vous donne accès à votre environnement de travail. Vous choisirez un interpréteur de commandes (Terminal) pour saisir les instructions de ce TP.

1 Ligne de commande : mode interactif

Dans cette première partie, nous utiliserons la ligne de commande en mode interactif, à chaque commande correspond un retour plus ou moins immédiat.

1.1 Terminologie

Afin de se comprendre lorsque l'on parle du shell, définissons un certain nombre d'éléments :
— interpréteur de commande :

— terminal :

— commande :

— shell :

— prompt :

— invite de commande :

— home / homedirectory :

— root :

— utilisateur :

— login :

— racine :

— complétion :

— wildcard :

De manière générale, et en très simplifié, les commandes sont saisies par l'utilisateur dans le terminal où elles sont «buffurisées». L'utilisateur «valide» sa commande, ou suite d'instructions, par un retour chariot (touche «Enter» ou «Entrée»). Le contenu du buffer est alors transmis à l'interpréteur de commande pour exécution.

1.2 Trouver de l'aide

Lorsque l'on recherche de l'aide sur l'utilisation d'un programme, le réflexe est d'ouvrir son navigateur internet et de lancer une recherche sur son moteur de recherche favori. Cependant, il est parfois plus rapide de consulter la documentation locale. Sous Linux, elle est accessible depuis le shell via la commande **man** (pour manuel).

1.2.1 Man pages

Lancez la commande : **man man** et lisez le document. De quelle commande parlent ces pages ? A quoi sert cette commande ?

1.2.2 Utilisation des pages de manuel

Toutes les pages de **man** sont organisées de la même manière.

1. NAME : nom de la commande ;
2. SYNOPSIS : résumé court de la syntaxe de la commande ;
3. DESCRIPTION : description longue ou courte de la commande ;
4. OPTIONS (facultative) : description des options supportées ;
5. Il existe bien sûr la possibilité de mettre autant de sous-sections que l'on désire ;
6. AUTHOR : les personnes qui ont développé la commande ;
7. SEE ALSO : des références croisées.

1.2.3 Manuel de la commande ls

Vous aller maintenant lire la page d'aide et tenter d'y trouver des informations répondant aux questions suivantes :

1. En 10 mots maximum, à quoi sert cette commande ?
2. Quelle option permet d'afficher tous les fichiers (même ceux cachés : ceux dont le nom commence par un point) ?
3. Quelle option permet d'afficher les fichiers avec un format long en liste ?
4. Quelle option permet d'afficher la taille des fichiers sous une forme facilement lisible par l'homme ?
5. Quelle option permet d'afficher récursivement le contenu d'un répertoire ?

Utilisez le «joker» «*» pour lister tous les fichiers commençant par «u» à la racine de votre système.

1.2.4 La commande cd

— A quoi sert cette commande ?
— rendez vous dans le répertoire **tmp** à la racine du système et lister son contenu
— retournez dans votre home directory
— Que fait **cd -** ?

1.2.5 Manuel de la commande mkdir

Considérons que nous voulons créer à la racine de votre compte le chemin suivant :

```
ArchiSys/tp/01
```

Lancez la commande suivante et observez ce qui se passe :

```
cd ; mkdir ArchiSys/tp/01
```

Cherchez dans la page de man de la commande «mkdir», l'option qui vous permet de créer une hiérarchie de répertoire en une seule commande.

1.3 La commande echo

À quoi sert la commande «echo» ?
Que fait la commande suivante ? Pourquoi ?

```
echo -ne "\n\n *\tHello World $LOGNAME\t\t*\n\n"
```

Essayez la commandes suivantes, quel est la différence ? Pourquoi ?

```
echo -e '\n\n *\tHello World $LOGNAME\t\t*\n\n'
```

1.4 Autres commandes

Dans la suite, nous utiliserons la commande «ps» qui liste les programmes en cours d'execution sur la machine et «cat» qui lit une entrée et la restitue sur la sortie standard. Consultez les pages de manuel de ces commandes si elles ne vous sont pas familières.

2 Les redirections

Comme vous l'avez tous remarqué sans y prêter attention, l'exécution d'une commande en ligne de commande peut vous afficher des informations à l'écran. Par exemple **«ls»** vous liste le contenu du répertoire courant.

Par défaut, le résultat de l'exécution d'une commande est affiché sur la **«sortie standard»** (aussi appelée **«stdout»**). Il est possible de changer ce comportement par défaut pour «afficher» dans un fichier plutôt que sur le terminal le résultat d'une commande : cette pratique s'appelle la redirection.

Notez également que tout programme possède sa sortie standard, mais ne l'utilise pas forcement.

De manière générale, la plupart des programmes fonctionnent comme suit :

— si un ou plusieurs fichiers sont spécifiés, le programme travaille sur ces fichiers ;
— si aucun fichier n'est spécifié, l'entrée standard est utilisée.

2.1 Premières redirections

Exécutez chacune des commandes suivantes (dans l'ordre) dans votre terminal, examinez les résultats et déduisez ce qui ce passe (il faudra bien évidemment regarder le contenu des fichiers générés) :

```
ls
ls > file1
pwd > file1
ps aux > file2.txt
cd > file3
cat file1 file2.txt > file4
cat file1 >> file1
pwd >> file1
cat file1 > /dev/null
```

À quoi servent les mots clefs «>» et «>>» dans les exemples précédents ?

2.2 Le pipe

Le pipe est aussi une forme de redirection, mais cette fois au lieu de rediriger la sortie standard sur un fichier comme précédemment, nous allons la rediriger vers une commande (pour simplifier, un programme). Notez que, comme pour la sortie standard, tout programme possède une **«entrée standard»**, **«stdin»**, qui peut ou non être utilisée. Dans le terminal, l'entrée standard est souvent le clavier.

2.2.1 pipe et cat

Lancez la commande cat dans un terminal. Écriver quelques mots à l'écran puis appuyez sur «Enter» (ou «Entrée» en fonction de votre clavier). La ligne est dupliquée. Pourquoi ?

2.2.2 Exemples de pipe

Exécutez chacune des commandes suivantes (dans l'ordre) dans votre terminal, examinez le contenu des fichiers manipulés et en déduire ce qui se passe :

```
ls -R > file1
less file1
cat file1
cat file1 | less
ps aux
ps aux | grep -v root
ps aux | grep root
ps aux > file2
cat file2 | grep -v root
```

Questions complémentaires : à quoi sert grep ? et son option '-v' ?

Lister les processus n'appartenant pas à «root» mais contenant root dans leurs nom et les consigner dans le fichier 'processwithroot'.

2.3 Redirection d'entrées

Tout comme il est possible de rediriger la sortie d'une commande, il est parfois intéressant de rediriger son entrée. Expliquez le comportement de la commande suivante :

```
cat <<eob
a
b
```

```
c
eob
```

Quel est la différence entre **<**, **<<** et **<<<** ?

3 Gestion de processus

Nous désignons par «gestion de processus» la manipulation des programmes en cours d'exécution à partir d'un terminal. Il existe bien sur des clients graphiques, comme sous Windows ou Mac OS, qui permettent de faire ce que nous allons faire dans cette section en ligne de commande.

Un programme en cours d'exécution s'appelle un **processus**. Les processus sont identifiés sur le système par un numéro appelé «PID» (process identifier).

Sous Unix en ligne de commande, un processus peut être exécuté de deux manières différentes :

— En tâche de fond (background) : lorsque que le processus est lancé en tâche de fond, le terminal vous rend la main immédiatement, alors que que le processus continue à s'exécuter : une commande qui s'exécute rapidement vous rendra la main immédiatement alors qu'elle n'est pas lancée en tâche de fond ;

— En avant plan (foreground) : le terminal vous rend la main seulement quand le processus s'est terminé.

Nous allons voir qu'il est possible de basculer un processus d'un état à un autre dans le terminal, et de tuer des processus. Nous verrons aussi qu'il est possible de lister les processus en cours d'exécution dans un terminal grâce à la commande **«jobs»**.

3.1 Processus en tâche de fond

Le «et commercial», aussi appelé «esperluette» en fin de commande permet de lancer un processus en tâche de fond :

```
xterm &
emacs monfichier &
```

Dans ce cas, le Shell exécute la commande xterm et rends la main après avoir affiché le PID numéroté par le shell depuis 1, puis il lance de même emacs.

Exemple de retour après xterm :

```
[1] 1360
```

puis après le emacs :

```
[2] 1366
```

Dans ce cas, le PID d'xterm est «1360» et ce processus est numéroté «1» dans le shell tandis que emacs (PID 1366) est numéroté «2» dans ce shell.

3.2 Processus en avant plan

Lancer les mêmes commandes que précédemment mais sans l'esperluette.

3.3 Basculement d'un état à un autre

Lorsqu'un processus est lancé en «foreground», vous ne pouvez plus interagir avec le Shell qui l'a lancé. Pour «reprendre la main» il faut basculer le processus en «background». La séquence de touche «CTRL + Z» met tout d'abord en sommeil le processus en cours d'exécution en avant plan, ensuite la commande «bg» (pour «background» bien sûr) réveille le processus et l'exécute en tâche de fond.

NB : La touche «Contrôle» peut être notée «CTRL» ou «^». Ces notations sont équivalentes.

À l'inverse, le dernier processus lancé en background (avec le «&»), peut être basculé au premier plan avec la commande «fg» suivie de «Enter» (ou «Entrée» en fonction des claviers).

— Dans un terminal lancer un nouveau terminal «xterm» ;
— lancer la commande ls dans le nouveau terminal ;
— mettre le nouveau terminal en sommeil ;
— lancer la commande ls dans le nouveau terminal, que se passe-t-il ? Pourquoi ?
— basculer le terminal en arrière plan
— relancer «ls» que se passe-t-il ? Pourquoi ?

3.3.1 Application sur les états de processus

Saisir le code suivant dans le fichier "script1" (utiliser cat et une redirection) :

```
while true ; do
 for i in {1..5} ; do
  echo bla
  sleep 1
 done
 echo -n ">> "
 read a
 echo $a
done
```

Lancez la commande suivante «bash script1», lorsque la ligne le terminal affiche «>> » saisir un caractère et valider par «Enter» ou «Entrée».

Après plusieurs executions, mettre le script en sommeil. Que ce passe-t-il ?

Réveiller le processus en avant plan, que ce passe-t-il ?

Basculer le processus en arrière plan. Que ce passe-t-il si vous saisissez «ls» lors de l'affichage de «>>» ? Proposez une explication.

Re prenez la main sur le processus le script et arrêtez le.

3.4 Tuer un processus en foreground

Pour tuer un processus exécuté en premier plan, saisir la séquence «CTRL + C» dans le terminal exécutant le processus.

3.5 jobs

En vous aidant de la page d'aide du bash à la section «JOB CONTROL», exécutez les commandes ou séquences claviers suivantes et analysez le comportement des processus (vous aurez bien entendu besoin de deux terminaux, un pour la manpage, l'autre pour l'exercice).

NB : Pensez à vous repositionner dans le terminal après le lancement des applications.

```
emacs &
xterm &
firefox &
```

```
emacs
Ctrl + C
jobs
fg %3
Ctrl + Z
bg
fg %2
Ctrl + C
kill %1
jobs
```

Quel processus reste-t-il après l'exécution de ces commandes ?

3.6 Chaîner des processus

Lorsqu'un programme termine son exécution, il retourne une valeur liée à la manière dont le processus s'est arrêté. Cette valeur peut être utilisée afin de chaîner des commandes. Pour cela, nous utiliserons les opérateurs «&&» pour «ET» et «||» pour «OU».

3.6.1 Première chaîne de processus

Avec la commande «grep», testez l'existance du compte «root» sur votre système et afficher le message : «root existe» si le compte est trouvé.

NB : le fichier «/etc/passwd» contient la liste des comptes existant sur le système.

3.6.2 Chaîne plus évoluée

En utilisant ces opérateurs et ce qui a été vu précedemment, écrire une commande affichant **UNIQUEMENT** le texte «Firefox est en cours d'exécution» sur la sortie standard si le programme firefox est exécuté sur votre machine et «Pas de firefox actuellement» sinon.

4 Le système d'exploitation GNU/Linux

4.1 Généralités

- La racine du système d'exploitation Linux est notée "/".
- Sous Linux tout est fichier.
- Un utilisateur est une personne ou un programme enregistré sur le système (disposant d'un compte).
- Des groupes permettent de classer les utilisateurs en fonction de critères définis par l'administrateur du système.
- Les utilisateurs peuvent appartenir à un ou plusieurs groupes.
- Les utilisateurs disposent de privilèges en fonction de leurs groupes d'appartenance.
- Pour le système, un utilisateur est un numéro appelé «uid» associé à un «login».
- Pour le système, un groupe est également un numéro.
- L'utilisateur appartient par défaut à un groupe dont le numéro est le «gid».
- Les droits, ou permissions, pouvant être affectés à un fichier sont : aucun droit noté «-», «lecture» noté «r», «écriture» noté «w» et exécution ou droit de traverser noté «x».
- Les droits peuvent être octroyés aux utilisateurs, aux groupes ou aux autres (personnes n'appartenant pas aux deux ensembles précédents).

— Les droits sont affichés sous la forme de triplets "rwx" pour chaque ensemble auquel ils peuvent être octroyés (au final nous obtenons des permissions allant de «− − − − − − − − −», aucun droit, à «rwxrwxrwx» tous le monde peut accéder en lecture écriture au fichier).
— L'option «-l» de «ls» permet de consulter les droits appliqués aux fichiers.
— La commande «chmod» (change mode) permet de modifier les droits sur les fichiers.
— La commande «chown» (change owner) permet de modifier le propriétaire et/ou le groupe propriétaire d'un fichier.
— Sur un système d'entreprise, un utilisateur simple ne peut élever ses privilèges sans autorisation.
— Le répertoire personnel d'un utilisateur est noté «~login». Si «login» est omis, l'utilisateur courant est sous-entendu.
— «groups» Permet de lister les groupes d'appartenance d'un utilisateur et «id» précise leur numéro.
— «pwd» Vous localise sur le système (affiche le répertoire courant dans lequel vous vous trouvez).

4.2 Questions

— Quels sont les droits sur votre répertoire personnel ?
— A quel groupe appartient votre «homedir» ?
— A quels groupe appartenez-vous ? et votre voisin ?
— Rendez-vous dans le «homedir» de «fhh». Que se passe-t-il ? Pourquoi ?
— Quel est votre gid ?
— «chmod» Dispose d'une notation octale des droits. Donner deux commandes, une avec notation octale, l'autre avec notation alphabétique, permettant de définir les droits suivants sur le répertoire «mydir» (droits de départ «rwxr-xr-x») :
  — rwx− − − − − −
  — rwxr-x− − −
— Changez le propriétaire du répertoire tmp. Que se passe-t-il ? Expliquez.

5 Les variables

Sous Unix, le système est capable d'échanger ou de récupérer des informations par l'utilisation de variables. Ces variables sont rendues accessibles par le Shell. Certaines sont des variables du système d'exploitation (variables d'environnement), d'autres sont propres au Shell. Nous allons en étudier quelques unes, comment les modifier, les créer ou les afficher.

5.1 Déclaration

La déclaration d'une variable est du type :

```
1 nomvar=valeur
```

Où «valeur» est une chaîne de caractères, et «nomvar» est une chaîne composée des caractères ASCII de base et ne commençant pas par des chiffres. Par exemple :

```
1 name="Jean"
```

5.2 Affichage

Le bash (qui est un type de Shell) vous fournit la commande «echo». Cette commande vous permet d'afficher la valeur associée au nom d'une variable. Les variables doivent être préfixées par le caractère «$». La commande suivante affiche la valeur associée à la variable denommée «name» :

```
echo $name
```

Bien que cette notation fonctionne, la notation recomandée par bash est :

```
echo ${name}
```

5.3 Modification

Modifier une variable consiste à écraser sa précédente déclaration.

5.4 Exercices

1. Affichez le contenu de la variable HOME. A votre avis que représente cette variable ?
2. Affichez le contenu de la variable PWD. A votre avis que représente cette variable ?
3. Affichez le contenu de la variable PATH. A votre avis que représente cette variable ?
4. Affichez le contenu de la variable PS1. A votre avis que représente cette variable ? Cherchez dans la page d'aide du bash.
5. Modifiez le contenu de la variable PATH, puis essayer de lancer une commande. Que se passe t'il ?
6. Modifiez la variable PS1, lancez un nouveau terminal. Que constatez-vous ?
7. Modifiez le fichier de configuration du bash pour y inclure votre définition de la variable PS1. Quelles sont les différences de comportement ?

5.5 Chaîne de caractères

Avec le shell vous pourrez trouver des chaînes de caractères entre double quotes (“foo”), simple quote (’foo’) ou anti-quote (‘foo‘). Testez les commandes suivantes :

```
echo $HOME
echo "$HOME"
echo '$HOME'
echo `$HOME`
TEST=ls ; echo "`$TEST`"
test=$(ls) ; echo $test
```

6 Installer un programme

Vous n'êtes pas autorisé à installer des paquets (fichiers permettant de déployer les programmes sous Linux) sur les systèmes de la salle 411, cependant, vous pouvez compiler et installer un programme ou une bibliothèque que vous souhaiteriez utiliser dans votre répertoire personnel. Dans cet exercice, nous allons compiler une version récente d'OpenSSH dans notre «home» afin de l'utiliser dans le prochain TP.

— Télécharger les sources d'OpenSSH dans un répertoire src à la racine de votre compte via la commande «wget» ;
— Extraire les sources à l'aide de la commande «tar» ;
— Configurer les sources pour installer le programme dans ~/Applications
— Compilez les sources en utilisant la commande «make»

— Installez le programme, que se passe-t-il ? Corrigez si besoin.
— Vérifiez la version de SSH et commentez le résultat.
— Ajouter le répertoire ~/Applications/bin au PATH
— Vérifiez la version de SSH et commentez le résultat.
— Rendrez la modification pérenne

7 Les scripts

Un script est un ensemble d'instructions consignées dans un fichier qui seront exécutées séquentiellement.

Dans la suite de ce TP nous utiliserons l'interpréteur de commande bash.

Tous les blocs d'instruction d'un script peuvent être testés directement dans le shell.

```
cat > myscript
cd /tmp
ls
cd
^d
```

Le script est exécuté depuis la ligne de commande :

```
bash myscript
```

Dans cet exemple nous précisons le shell à utiliser («bash») et lui passons en argument le fichier contenant les instructions.

Nous pouvons également préciser, directement dans le script, l'interpréteur de commande à utiliser en le spécifiant dans l'entête du fichier

```
cat > myscript
#!/usr/bin/bash

cd /tmp
ls
cd
^d
```

Cette première ligne s'appelle le **«shebang»**.

La commande «whereis» permet de localiser le shell que vous souhaitez utiliser :

```
whereis bash
```

le script peut ensuite être invoqué comme n'importe quel programme.

```
./myscript
```

Là, ça ne fonctionne pas, pourquoi ? Corrigez.

7.1 Structure de contrôle

«man test» donne la liste des tests disponible en bash.

7.1.1 if .. then ... else

Syntaxe :

```
if commande(s) ; then
 commande(s)
[ elif commande(s) ; then
 commande(s) ] ...
[ else commande(s) ]
fi
```

En utilisant un «if», testez l'existence du répertoire «/var/logs», puis du répertoire «/var/lOg» et afficher un commentaire en cas de répertoire trouvé ou non.

NB : Dans le cas de ligne de commande longue, vous pouvez utiliser «\» pour passer à la ligne et continuer votre commande.

Afficher «Alerte» si vous avez le droit d'écrire dans «/etc».

7.1.2 for ... do ... done

Syntaxe :

```
for var in list ; do
 commande(s)
done
```

Ecrire un «for» qui affiche les lettres «a b c d» a raison d'un par ligne.

7.1.3 while ... do ... done

Syntaxe :

```
while commande(s) ; do
 commande(s)
done
```

Que fait la séquence suivante ?

```
1 ls | while read a ; do echo $a ; done
```

7.2 Scanner réseau

Ecrire une boucle testant votre environnement réseau. Vous utiliserez «ping» pour tenter d'appeler les 254 machines de votre plage IP et n'afficherez que les adresses IP disponibles (ayant répondu au ping).

7.3 Script poubelle

Écrire un petit script qui prend en paramètre un ensemble de fichiers et les déplace dans la poubelle (un répertoire «.trash» à la racine de votre répertoire utilisateur). Afficher un message d'erreur si un fichier n'existe pas. Dans un premier temps trouvez la variable spéciale qui représente tous les paramètres de la ligne de commande (voir man bash). Réalisez le petit script (10 à 15 loc).

8 Quelques raccourcis bien pratiques

Si vous utiliser l'éditeur de texte **emacs** (Ô joie !), vous aurez la chance de pouvoir réutiliser certains raccourcis clavier lors de la saisie d'une commande [1].

— Que fais **Ctrl+R** ?
— Que fais **Ctrl+A** ?
— Que fais **Ctrl+E** ?
— Comment copier/coller du texte vers ou depuis votre terminal ?

Si vous utilisez la commande **man** cependant, les raccourcis changent et utilisent plutôt ceux de **vi** : Pour rechercher du texte, il faut utiliser la commande / puis les touches **n** et **p** pour naviguer entre les différentes occurences par exemple.

9 Tester vos connaissances

Pour aller plus loin avec la ligne de commande, je vous recommande d'aller voir l'URL suivante : `http://overthewire.org/wargames/bandit/`. Votre but est d'aller jusqu'au niveau 34 ! Il faut réussir le niveau n pour débloquer le niveau $n + 1$. Bonne chance !

10 Questions bonus

— Que fait la commande **which** ?
— Quelle est la différence entre les redirections **>** et **2>** ?
— Comment mélanger ces deux redirections ?
— Quel résultat produit la commande **[** ? la commande **]** ? Comment l'expliquez-vous ?
— Que fait la commande **pkill** ?
— Que fait la commande **less** ? la commande **more** ? Et oui, les informaticiens avaient le sens de l'humour dans les années 90...
— Que fait le commande **yes** ? À votre avis à quoi peut-elle servir ?
— Chercher sur internet une réponse qui explique pourquoi les noms des fichiers cachés commencent par un point.
— Faites des recherches sur **zsh**.

11 Gallerie photo

L'objectif de cette partie est de créer un script générant une gallerie photo html.

Dans un répertoire se trouve des photos aux formats PNG et JPG. Ce répertoire peut contenir des sous répertoires, correspondant aux collections de photos.

Nous souhaitons un script bash prenant en argument le répertoire contenant les photos et générant une page html contenant la liste des collections et une icône associée, puis une icône de chaque image classée par date de prise de la photo. Losque l'on clique sur une icone de photo, la photo apparaît en 1024x768 (en conservant le ratio longeur largeur). Les images sont pivotées en fonction de la prise de vue. Dans le répertoire contenant les photos, ou les collections, un fichier "comment.txt" permet de définir des commentaires au format "nom_photo : Commentaire sur la photo" qui sera affiché sous la photo lors de l'affichage plein écran.

Toutes les évolutions imaginées sont les bienvenues.

Pour extraire la position des photos et les dates de prises de vues, vous utilisez le programme «exiv2».

Les icônes des images appelées «thumbnail» seront générées via la commande «convert». Les icônes déjà générées ne seront pas regénérées (utilisez le md5sum des fichiers comme nom d'icône par exemple).

1. Si vous êtes utilisateur de **vi**, il faut changer le mode d'édition : `https://sanctum.geek.nz/arabesque/vi-mode-in-bash/`