Formation ligne de commande

M. Gueguen

Pourquoi ?

- Pour soumettre des calculs sur les calculateurs, il faut utiliser la ligne de commande !
- connaissance des outils sur l'OS Linux
- Pour devenir un geek !

Pourquoi ?

- Pour soumettre des calculs sur les calculateurs, il faut utiliser la ligne de commande !
- connaissance des outils sur l'OS Linux
- Pour devenir un geek !

Au sommaire

- shell et commandes
- le systême de fichier
- gestion et manipulation des fichiers
- entrées/sorties des commandes
- gestion des processus

un compte sur Linux

Linux

- Systême d'exploitation (libre) multi taches, multi utilisateurs ;
- dans le cas des calculateurs, la distribution est une Red Hat (RHEL 6 `https://fr.wikipedia.org/wiki/Red_Hat_Enterprise_Linux`)

Informations d'un compte

identifiant	nom de login (ou user name) ou `uid`
groupe d'appartenance	group name ou `gid`
authentification	mot de passe (`password`)
shell de connexion	`login shell`
répertoire	`home directory`
autres	: nom, prénom, ...

se connecter

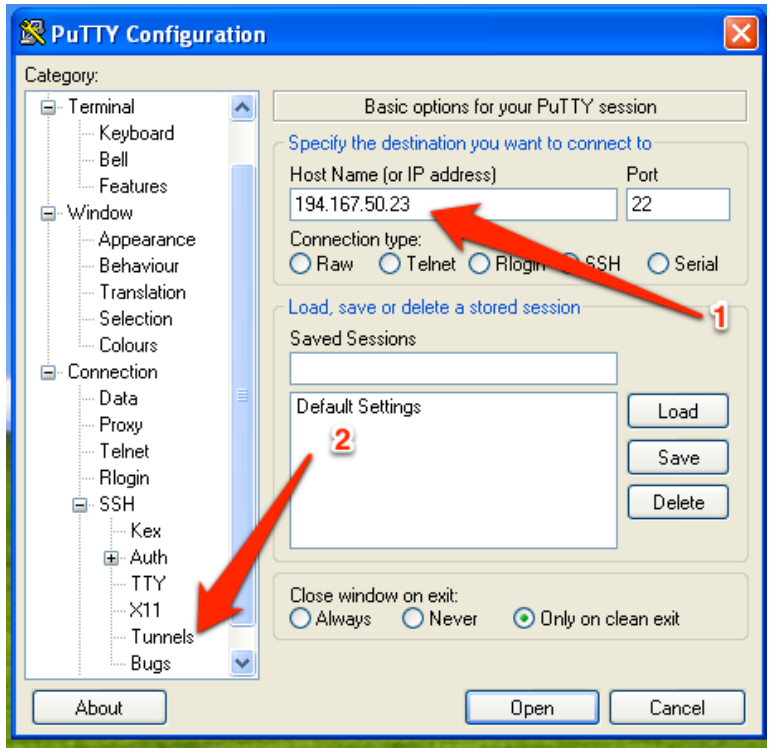
Connection à une machine distante

- Utilisation d'un protocole sécurisée pour la connection entre 2 machines : `ssh`
- `https://fr.wikipedia.org/wiki/Secure_Shell`

Comment ça marche ?

- Il faut renseigner le nom ou l'adresse IP de la machine distante ;
- utilise un port de communication (par défaut le 22 ; sur les calculateurs `THOR` ou `HULK` : 86)
- authentification par mot de passe (défaut) ou clé publique
- nombreux paramètres !
- disponible par défaut sous Unix/OSX/Linux ;

  (nombreux) utilitaires à installer sous windows
  - putty : `http://www.chiark.greenend.org.uk/~sgtatham/putty/` `http://marc.terrier.free.fr/docputty/`



se connecter

qq paramétrages ssh via Putty

Eviter les déconnections *connexion : Seconds between keepalives = 60*

en lignes de commandes (sous macos, linux)

```
cameleon:~ mik$ ssh -X gueguenm@thor.univ-poitiers.fr -p 86
Last login: Mon May 30 21:38:48 2016 from apoitiers-655-1-9-161.w90-50.abo.wanadoo.fr
```

transfert de fichier : utilisation de scp et le client winscp

- Il faut renseigner le nom ou l'adresse IP de la machine distante ;
- changer le port 22 en 86
- renseigner votre login ; mot de passe
- paramétrables : ajout de nouveaux éditeurs de texte comme *notepad++*, cacher les fichiers systèmes, éviter les déconnections

en lignes de commandes (sous macos, linux)

```
cameleon: mik$ scp -r -P 86 code_integ/ gueguenm@thor.univ-poitiers.fr:~/
cmake_install.cmake                   100% 1582     1.5KB/s   00:00
CMakeCache.txt                        100%   17KB  17.2KB/s   00:00
CMakeCCompiler.cmake                  100% 1619     1.6KB/s   00:00
```

un compte sur Linux

2 modes d'utilisation

- graphique (environnement KDE, GNOME, ... sous linux)
  - utilise un serveur X
  - un gestionnaire de connexion (display manager)
  - un gestionnaire de fenêtre
  - un environnement de bureau
- la ligne de commande via le `terminal` et l'interpréteur de commande (`shell`)

shell ? ? ? ?

shell [a] : Kesako ?

a. coquille en english

- programme qui interprète les commandes et les transmets au noyau (kernel)
- langage de programmation interprété
- interface entre la machine et l'utilisateur (au même titre que l'interface graphique)
- notion de variables, de structure de controle
- à l'ouverture du terminal (à la connexion), le shell s'initialise en exécutant des scripts et propose une invite de commande (`prompt`)

Plusieurs types de `shell`

famille Bourne shell

bourne shell	`sh` (l'original !)
korn shell	`ksh`
bourne again shell	`bash` (utilisé pour cette formation, par défaut sur de nombreux systèmes linux, mac os x)
zero shell	`zsh`

famille C-shell

C-shell	`csh`
TENEX C-shell	`tcsh`

caractéristiques

- chaque familles de `shell` a ses spécificités ; mot clés ; structures de contrôles
- pour une famille, il y a des évolutions suivant les `shell` (par ex `bash` est une "extension" de `sh`)
- les commandes de base sont généralement identiques pour tous les types de `shell`

l'utilisation du terminal

la connection via le terminal

- initialisation du `shell` ; définition de certaines variables ; des chemins vers les programmes ;
- lancement du `prompt` ; sa forme est modifiable :
  - "`%`" (famille c-shell) ou "`$`" (famille bourne) ou "`>`"
- 1ère commandes ...
- tous les `shell` sont sensibles à la casse (pour les commandes et les variables) ! : `date` $\neq$ `DATE` $\neq$ `Date`
- fermeture du terminal : `exit` ou `<CTRL-D>`

```
[gueguenm@thor ~]$ date
Tue Oct 13 15:03:03 CEST 2015
[gueguenm@thor ~]$ Date
-bash: Date: command not found
```

les 1ères commandes

syntaxe générale : `command_name options arguments`

- le caractère séparateur entre les différentes parties d'une commande est l'espace `<SPACE>`
- les options modifient le comportement de la commande ; précédées du signe moins "-"
- les arguments (parfois facultatifs) sont les données d'entrée du programme
- le programme s'exécute et effectue des opérations (affichage à l'écran ou autre)
- il retourne un code d'erreur si la commande s'est correctement exécuté ($= 0$) ou non ($\neq 0$)

```
[gueguenm@thor ~]$ date
Tue Oct 13 15:18:37 CEST 2015
[gueguenm@thor ~]$ date -u
Tue Oct 13 13:18:59 UTC 2015
[gueguenm@thor ~]$ date -R
Tue, 13 Oct 2015 15:19:27 +0200
```

```
[gueguenm@thor ~]$ man date
DATE(1)
User Commands
DATE(1)
NAME
       date - print or set the system date an
SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[
DESCRIPTION
```

les 1ères commandes

syntaxe générale : `command_name options arguments`

- chaque commande (système) doit fournir un support (`man`, `cmd -h`, `cmd --help`, `apropos`, `whatis`)
- normalement `man` doit donner des informations
  - `<SPACE>`, `<CTRL-F>`, `<CTRL-B>`, `<CTRL-↓>`, `<CTRL-↑>` pour se déplacer, touche "`q`" pour quitter
  - on peut effectuer des recherches avec le slash "`/`" et touche `<N>` pour continuer la recherche

```
[gueguenm@thor ~]$ date
Tue Oct 13 15:18:37 CEST 2015
[gueguenm@thor ~]$ date -u
Tue Oct 13 13:18:59 UTC 2015
[gueguenm@thor ~]$ date -R
Tue, 13 Oct 2015 15:19:27 +0200
```

```
[gueguenm@thor ~]$ man date
DATE(1)
User Commands
DATE(1)
NAME
       date - print or set the system date an
SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[
DESCRIPTION
       Display the current time in the given
       -d, --date=STRING
              display time described by STRIN
```

les informations de l'utilisateur

Afficher son login

```
[gueguenm@thor ~]$ whoami
gueguenm
```

Afficher le uid, gid

```
[gueguenm@thor ~]$ id
uid=1008(gueguenm) gid=1007(pprime) groups=10
  1009(endo)
```

Afficher plusieurs informations

```
[gueguenm@thor ~]$ finger gueguenm
Login: gueguenm          Name:
Directory: /home/gueguenm
Shell: /bin/bash
On since Tue Oct 13 13:20 (CEST)  \
  on pts/19 from 193.55.160.54
No mail.
No Plan.
```

connaitre le nom et le type de la machine

```
gueguenm@thor ~]$ hostname
thor
[gueguenm@thor ~]$ uname -a
Linux thor 2.6.32-431.17.1.el6.x86_64 #1 SMP
  x86_64 x86_64 x86_64 GNU/Linux
```

qq fonctionalités du terminal/shell

Complétions avec la touche <TAB>

```
[gueguenm@thor ~]$ da
dash   date
```

Historique history n avec n le nombre de dernières commandes (défaut : toutes)

```
[gueguenm@thor ~]$ history 2
20067  [2015-10-13 16:54:04] gload
20068  [2015-10-13 16:54:08] echo $TERM
```

- paramétrables avec les variables HISTSIZE, HISTCONTROL, HISTTIMEFORMAT
- les touches <CTRL-R> permet de faire des recherches dans l'historique

Déplacements sur la ligne de commande :

- <CTRL-A> pour aller au début de la ligne,
- <CTRL-E> pour aller à la fin de la ligne
- <CTRL-←> pour aller au début des mots
- <CTRL-→> pour aller à la fin des mots

les variables définies par le shell

les variables en langage shell

- à la connexion, définition de variables pour l'environnement utilisateur
- une variable est une case mémoire dans laquelle est enregistrée une valeur et identifiée par un nom
- pour bash, ces variables ne sont pas typées, (pas besoin de la déclarer au préalable)
- l'opérateur "=" permet d'affecter une valeur à un variable.
- le caractère "$" suivi du nom de la variable (éventuellement entre accolades) renvoi la valeur contenu dans la variable
- l'ensemble des variables définies dans le terminal peut être vue avec la commande env
- le contenu d'une variable peut être affiché avec la commande echo

```
[gueguenm@thor ~]$ echo $HOME
/home/gueguenm
[gueguenm@thor ~]$ echo $TERM
xterm
[gueguenm@thor ~]$ echo $PWD
/home/gueguenm
[gueguenm@thor ~]$ echo $pwd

[gueguenm@thor ~]$ echo pwd
pwd
```

paramétrage d'un compte pour un utilisateur

Les fichiers de configuration sont dans le home d'un utilisateur ($HOME, ie /home/homer pour l'utilisateur homer) :

A la connection

- .bash_profile va être lu et les commandes dans le fichier exécutées
- .bash_profile exécute le fichier .bashrc
- .bashrc va être lu et les commandes dans le fichier exécutées

```
cat .bash_profile

# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
```

toute commande faite dans le terminal peut être intégrée dans ce fichier.

File system

définition

Structure de données permettant de stocker et d'organiser des informations sur des mémoires secondaires (disques durs...)

unité d'information sur les FS linux : les fichiers

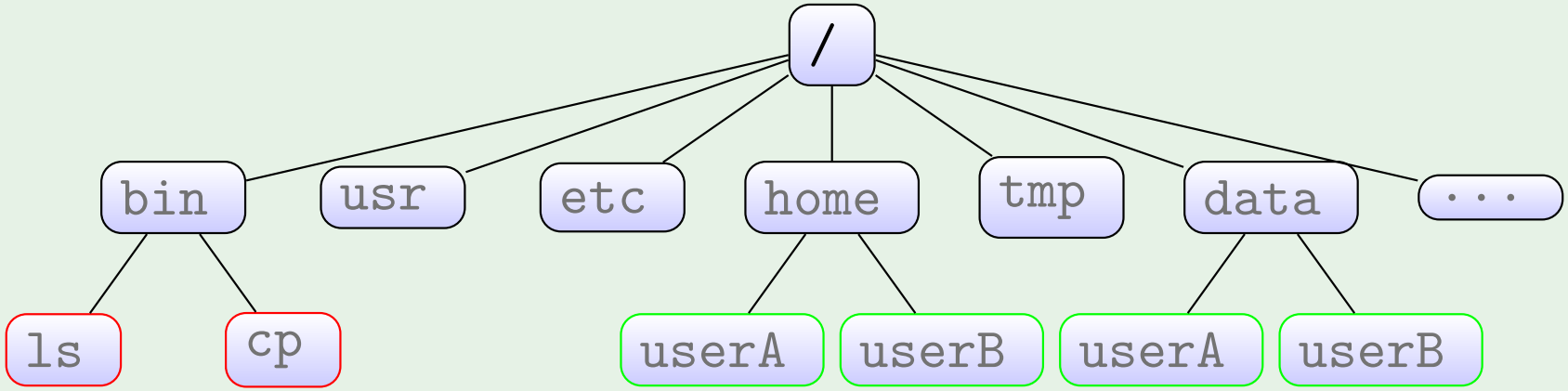
- fichiers normaux (`regular files`) : textes, exécutables
- répertoires (`directory`) : pouvant contenir d'autres fichiers
- fichiers spéciaux : périphériques, liens, ...

structure de l'espace de nommage

Structure arborescente dont la racine (`root directory`) est représentée par " /".

File system

exemple d'arborescence



- l'emplacement d'un fichier est donné par la suite des répertoires intermédiaires, séparés par " /" .
- l'emplacement peut être absolu à partir de la racine " /"
- l'emplacement peut être relatif à partir du répertoire courant

les fichiers

syntaxe générale

- 255 caractères maximum
- sensibilité à la casse

conseil

- ne pas utiliser d'espaces ou de caractères spéciaux (*, ?,&, !,/,$) dans les noms de fichiers/répertoires
- se limiter aux caractères alphanumériques `[a-z]`, `[A-Z]`, `[0-9]`, au point "." et au underscore "_".
- le suffixe est libre ; peut indiquer le type de fichier : texte, objet, archive, type de fichier texte (`.pdf`, `.f90`, `.c`, `.zip`, `.tex...`)
- pas de suffixe pour les répertoires ou exécutables en général

les fichiers

Caractéristiques d'un fichier

- type
- un nombre de liens physique
- un propriétaire et un groupe
- une taille et une adresse
- un horodatage (date de modif)
- droit d'accès (`lecture : r ; écriture : w ; exécution : x`) pour le propriétaire, le groupe du propriétaire et les autres

les fichiers

exemples

```
[gueguenm@thor ~]$ pwd # affiche le répertoire courant
/home/gueguenm
cameleon:cluster mik$ ls -l arbo.png # affiche les informations sur un fichier/répertoir
-rw-r--r-- 1 mik admin 258453 13 oct 21:54 arbo.png
```

`ls -l` : les champs

1. `-rw-r--r--` liste les droits, ainsi que le type du fichier (1er caractère)
2. nombre de liens physiques vers le fichier
3. propriétaire et groupe
4. taille du fichier
5. date de modification
6. nom du fichier

créer/déplacer les répertoires

`pwd` : afficher le répertoire courant

```
[gueguenm@thor ~]$ pwd
/home/gueguenm
```

`mkdir` : créer un répertoire

```
[gueguenm@thor ~]$ mkdir newrep
[gueguenm@thor ~]$ mkdir -p newrep1/newrep2
```

`cd` : se déplacer dans les répertoires

`cd sousrep`	se déplacer du répertoire courant vers un sous répertoire
`cd ..`	se déplacer vers (ou à partir) le répertoire parent
`cd .`	se déplacer vers (ou à partir) le répertoire courant (ne fait rien !)
`cd ~`	se déplacer vers (ou à partir) le répertoire `$HOME`
`cd`	se déplacer vers le répertoire `$HOME`
`cd /chemin/absolu`	se déplacer vers le répertoire absolu avec l'ensemble de l'arborescence
`cd ./sousrep/../../`	plusieurs combinaisons sont possibles ! !

manipuler les fichiers

lister les fichiers dans un répertoire : `ls -alrhtRd`

`ls`	liste standard (sans les infos)
`ls -l`	liste avec les info
`ls -lh`	liste les fichiers avec une taille "lisible"
`ls -lht`	liste les fichiers avec une taille "lisible" et trié par date de modification
`-r`	modification de l'ordre de tri
`-S`	trie par taille de fichiers
`-a`	liste les fichiers cachés

copie de fichier : `cp`

`cp fichier1 fichier2`	copie `fichier1` vers `fichier2`
`cp fichier1 rep`	copie `fichier1` vers le répertoire `rep` (existant)
`cp -r rep1 rep2`	copie `rep1` vers le répertoire `rep2`
`cp fichier1 fichier2 fichier3 rep`	copie `fichier1,fichier2,fichier3` vers le répertoire `rep`
`cp -i src dest`	copie interactive (demande confirmation avant l'écrasement éventuel d'un fichier)

déplacement de fichier / modification de nom : `mv`

`mv fichier1 fichier2`	changement de `fichier1` en `fichier2`
`mv fichier1 rep`	déplacement `fichier1` vers le répertoire `rep` (existant)
`mv fichier1 fichier2 fichier3 rep`	déplacement `fichier1,fichier2,fichier3` vers le répertoire `rep`
`mv rep1 rep2`	changement de `rep1` en répertoire `rep2`
`mv -i src dest`	changement interactif (demande confirmation avant l'écrasement éventuel d'un fichier)
`mv -f src dest`	force

destruction de fichier : `rm`

`rm fichier1 fichier2`	destruction de `fichier1` et `fichier2`
`rm -r rep`	destruction du répertoire `rep` (vide)
`rm -rf rep`	destruction en force du répertoire et tout ce qu'il contient
`rm -i fichier`	destruction interactive (confirmation)

pas de récupération possible après la destruction ! !

fonctionnalités pour la gestion des fichiers

Méta caractères ou wildcards

- certains caractères permettent de remplacer d'autres caractères

?	remplace 1 caractère
*	remplace tout un ensemble de caractère
[0-9]	remplace tous les caractères numériques

- permet de lister/manipuler un groupe de fichier suivant les occurrences dans les noms, les suffixes...

```
[gueguenm@thor fox_job]$ ls
cavity_diffusion  cub_12C3D4.inp    cub_8C3D8.inp              polycristal
ct_model          cub_8000C3D8.inp  domainDecompbyScotch.vtk  Traction_f1_R0_dif.inp
[gueguenm@thor fox_job]$ ls *.inp
cub_12C3D4.inp  cub_8000C3D8.inp  cub_8C3D8.inp  Traction_f1_R0_dif.inp
[gueguenm@thor fox_job]$ ls cub*C?D?.inp
cub_12C3D4.inp  cub_8000C3D8.inp  cub_8C3D8.inp
```

visualiser des fichiers

affichage de l'ensemble `cat fichier`

```
cameleon:ct_model mik$ cat ctplastique_iso_exp_980.pvtu
<?xml version="1.0"?>
<VTKFile type="PUnstructuredGrid" version="0.1" byte_order="LittleEndian" compressor="vt
(...)
```

`more fichier, less fichier` : **affiche le contenu page par page**

- <RETURN>, <SPACE> pour aller à la ligne/page suivante
- <q> pour quiter

`head [-n x] fichier , tail [-n x] fichier` : **affiche le début ou la fin d'un fichier**

```
 cameleon:ct_model mik$ head  ctplastique_iso_exp_980.pvtu
<?xml version="1.0"?>
<VTKFile type="PUnstructuredGrid" version="0.1" byte_order="LittleEndian" compressor="vt
  <PUnstructuredGrid GhostLevel="0">
    <PCellData>
      <PDataArray type="Float32" Name="strain" NumberOfComponents="6"/>
(...)
cameleon:ct_model mik$ head -n 1  ctplastique_iso_exp_980.pvtu
<?xml version="1.0"?>
cameleon:ct_model mik$ tail -n 2 ctplastique_iso_exp_980.pvtu
  </PUnstructuredGrid>
</VTKFile>
```

modifications les droits sur les fichiers

changement des droits : `chmod [ugo+-rwx] -R fichier`

Pour tous les fichiers et répertoires, il existe un mécanisme de droits permettant d'accéder à tels fichiers ou répertoires suivant que l'on est son propriétaire ou autre. A chaque fichier sont associés un propriétaire et un groupe d'appartenance. L'option `-R` permet d'appliquer les droits à un répertoire récursivement.

- droit d'accès (`lecture : r ; écriture : w ; exécution : x`) pour le propriétaire `u`, le groupe du propriétaire `g` et les autres `o`

- Pour modifier les droits de l'utilisateur sur le fichier :`u`
- Pour modifier les droits de groupe sur le fichier : `g`
- Pour modifier les droits des autres sur le fichier : `o`
- Pour ajouter des droits `+`
- Pour retirer des droits `-`
- Pour les droits en lecture `r`
- Pour les droits en écriture `w`
- Pour les droits en exécution `x`

```
[gueguenm@thor ~]$ ls -l calcul_cavite.rst
-rw-r--r-- 1 gueguenm pprime 4672 Oct 12 16:33 calcul_cavite.rst
[gueguenm@thor ~]$ chmod go-r calcul_cavite.rst
[gueguenm@thor ~]$ ls -l calcul_cavite.rst
-rw------- 1 gueguenm pprime 4672 Oct 12 16:33 calcul_cavite.rst
```

quelques commandes pratiques

compter le nombre de ligne : `wc [-lwc] fichier`

```
[gueguenm@thor ~]$ wc  calcul_cavite.rst
  83  657 4672 calcul_cavite.rst
[gueguenm@thor ~]$ wc -l calcul_cavite.rst
83 calcul_cavite.rst
[gueguenm@thor ~]$ wc -c calcul_cavite.rst
4672 calcul_cavite.rst
[gueguenm@thor ~]$ wc -w calcul_cavite.rst
657 calcul_cavite.rst
```

différence entre 2 fichiers : `diff fichier1 fichier2`

```
 [gueguenm@thor ~]$ diff calcul.rst calcul.r
3c3
<
---
> Ajout d'une difference
```

espace disque : `du -sh`

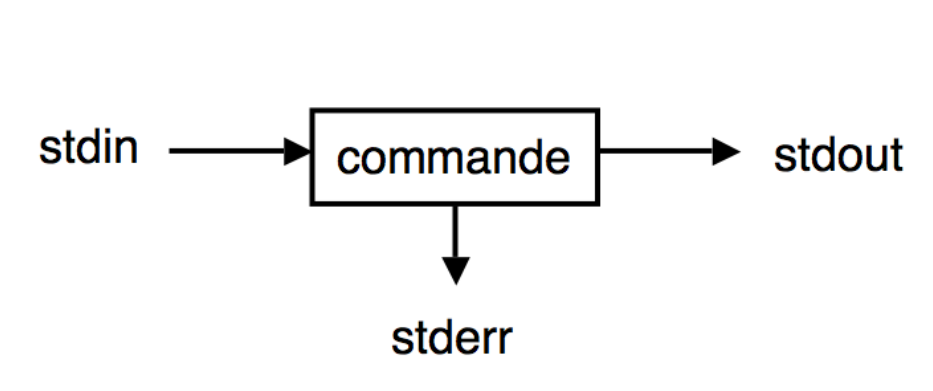
```
[gueguenm@thor ~]$ du -sh accounting/
3.0M accounting/
```

archivage : `tar`

- `tar -cvzf archive.tar.gz toto/` crée l'archive compressée du répertoire donné et de l'ensemble .
- `tar -tvzf archive.tar.gz` liste les composants de l'archive (sans les extraire).
- `tar -xvzf archive.tar.gz` extrait les fichiers de l'archive.

Les redirections

E/S

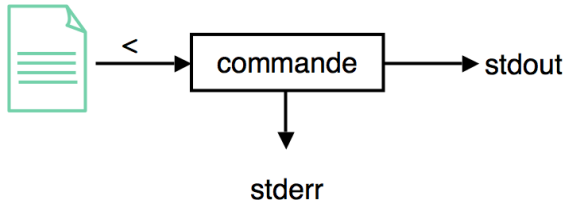


Flux de données du `shell`

- `stdin` : entrée standard ; descripteur de fichier 0, en général une saisie au clavier
- `stdout` : sortie standard ; descripteur 1, en général un affichage à l'écran
- `stderr` : erreur standard descripteur 2, en général un affichage à l'écran

On peut rediriger ces flux de données à partir ou vers un fichier ;

redirection de `stdin`

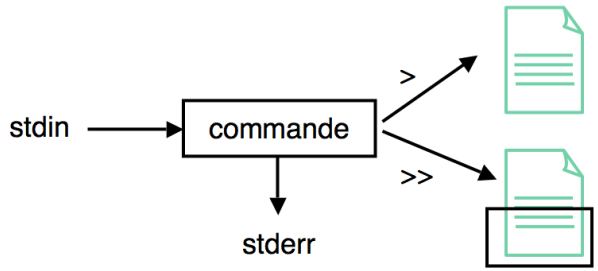


Famille C-shell (csh, tcsh)	Famille Bourne shell (sh, ksh, bash)
`command < file`	`command < file` équivalent : `command 0< file`

Lecture à partir d'un fichier

- ligne à ligne

redirection de stdout

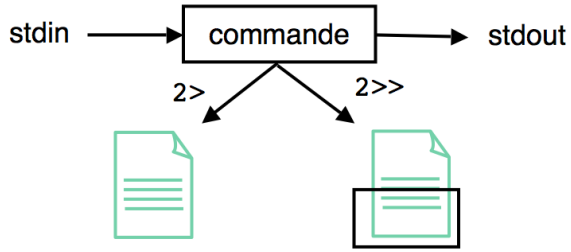


Famille C-shell (csh, tcsh)	Famille Bourne shell (sh, ksh, bash)
`command > file`	`command > file` équivalent : `command 1> file`
`command >> file`	`command >> file` équivalent : `command 1>> file`

vers un fichier

- ">" : si `file` existe déjà il est écrasé
- ">>" : si `file` existe déjà l'écriture se fait à la fin

redirection de stderr

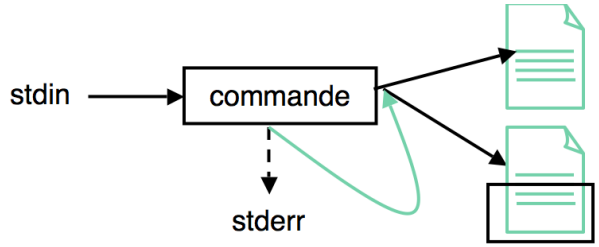


Famille C-shell (csh, tcsh)	Famille Bourne shell (sh, ksh, bash)
Pas de commandes directes	`command 2> file`
	`command 2>> file`

vers un fichier

- "2>" : si `file` existe déjà il est écrasé
- "2>>" : si `file` existe déjà l'écriture se fait à la fin

redirection de stderr et stdout



Famille C-shell (csh, tcsh)	Famille Bourne shell (sh, ksh, bash)
`command >& file`	`command > file 2>&1` équivalent : `command 1> file 2>&1`
`command >>& file`	`command >> file 2>&1` équivalent : `command 1>> file 2>&1`

gestion des flux

rediriger la sortie d'une commande vers l'entrée d'une autre : `pipes`

Ce mécanisme permet de chaîner des processus de sorte que la sortie d'un processus (`stdout`) alimente directement l'entrée (`stdin`) du suivant.

- syntaxe `cmd1 | cmd2 | cmd3`

Cette notion de tube permet de filtrer un fichier, une sortie d'une commande par l'intermédiaire de commandes "filtres" :

`wc`	comptage
`sort`	tri
`uniq`	élimination des doublons
`cut`	extraction de champ
`grep`	recherche de chaines de caractère

exemple : compter le nombre de fichier après la sortie de `ls`

```
[gueguenm@thor sim_abq]$ ls -l *.inp | wc -l
2
```

exemple : recherche du nombre de noeuds libre ou arrêté

```
[gueguenm@thor sim_abq]$ pbsnodes -a | grep "state =" | grep -v job | wc -l
5
 [gueguenm@hulk ~]$ pbsn|grep free | sort
hulk[28] free 32489472kb 16
```

l'utilisation de la commande `grep`

Recherche d'une chaine de caractère ou expression régulière dans un fichier :
syntaxe `grep [options] regex [fichier]`

Expressions régulières

+ chaines ou motif de caractères décrivant un ensemble de chaines de caractères possibles selon une syntaxe

+ les plus utiles
  - ⋆ ^ : début de ligne
  - ⋆ $: fin de ligne
  - ⋆ [x] : classe de caractère (ex : [a-z] : tous les caractères de a à z, [0-9] : tous les chiffres)
  - ⋆ . : n'importe quel caractère

+ options intéressantes
  - ⋆ `-v` : inversion (les lignes ne contenant pas le motif)
  - ⋆ `-i` : insensible à la casse
  - ⋆ `-R` : récursif pour la recherche dans un répertoire

l'utilisation de la commande grep

Recherche d'une chaine de caractère ou expression régulière dans un fichier :
syntaxe `grep [options] regex [fichier]`

exemples

```
 [gueguenm@thor ~]$ grep "^a" /tmp/user.txt
abeaudoi           2       769824         3170   242.847       271837   0.000
almorel            1    214132027       891768   240.121       107947   0.000
anassour          96     26425638      2306714   11.456         2331   0.000
apoux              9        13894        93977   0.148         4857   0.000
audiberc           8           72           63   1.143          645   0.000
[gueguenm@thor ~]$ grep "^ab" /tmp/user.txt
abeaudoi           2       769824         3170   242.847       271837   0.000
[gueguenm@thor ~]$ grep "^[ab]" /tmp/user.txt
abeaudoi           2       769824         3170   242.847       271837   0.000
almorel            1    214132027       891768   240.121       107947   0.000
anassour          96     26425638      2306714   11.456         2331   0.000
apoux              9        13894        93977   0.148         4857   0.000
audiberc           8           72           63   1.143          645   0.000
belhajss         105    512646404      2544518   201.471        15806   0.000
bhuang         33469     36261697      1952152   18.575            84   0.000
bilottag           1        43279        22986   1.883         7332   0.000
```

qq commandes supplémentaires

connaitre le type d'un fichier : `file filenames`

- type ASCII, exécutable, liens, répertoires

```
[gueguenm@thor ~]$ file /tmp/user.txt
/tmp/user.txt: ASCII text
[gueguenm@thor ~]$ file /bin/cp
/bin/cp: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses s
```

trouver un fichier exécutable : `which commands`

- recherche au sein des répertoires renseignés par la variable `$PATH`

```
[gueguenm@thor ~]$ which gload
/sw/tools/bin/gload
[gueguenm@thor ~]$ which abaqus
/sw/abaqus/Commands/abaqus
```

connaitre l'espace disque des systèmes de fichiers rattachés à la machine : `df`

- option `-h` pour faciliter la lecture

```
[gueguenm@thor ~]$ df -h
Filesystem                                  Size  Used Avail Use% Mounted on
/dev/sda31                                  457G  296G  139G  69% /
tmpfs                                        32G     0   32G   0% /dev/shm
/dev/sda11                                  291M   49M  227M  18% /boot
10.148.0.3@o2ib:10.148.0.4@o2ib:/home        22T   20T  1.2T  95% /home
```

Les processus

processus (ou tâche) ? ?

- correspond à toute exécution d'un programme à un instant donné, le programme constituant en lui-même un objet inerte, rangé sur disque sous la forme d'un fichier ordinaire exécutable.
- pour chaque processus, des ressources lui sont rattachées (mémoire, fichiers, librairies...)
- processus systême et utilisateurs
- certains processus sont actifs en permanence (les démons), d'autres non ;
- certains processus sont liés à un terminal, session (notion de hériarchie et de processus parent/fils)

chaque processus possède :

- **PID** (process identification) de type entier ; unique
- **PPID** (parent process identification) : PID de processus parent
- propriétaire et group identifiés par le UID et GID (également deux entiers)
- un terminal de controle TTY
- une priorité (NICE VALUE)
- d'autres arguments (répertoire de travail,...)

Etat d'un processus

différents états possibles entre la création et la fin d'un processus

- R en cours d'execution : le processus est actif
- Z processus terminé dont le parent n'a pas connaissance (plantage...)
- S endormi (attente d'une action)
- T stoppé

Les processus : connaitre les processus en cours

commande `top`

- permet de visualiser les processus du systême en temps réel ;
- visualise le `PID, USER, %CPU, COMMAND`
- trie par mémoire avec `M`, par cpu avec `P`
- liste par utilisateur avec en tapant `u` puis login
- aide avec `h`
- taper `q` pour quitter

```
top - 14:28:44 up 55 days,  2:40, 35 users,  load average: 3.10, 3.14, 3.39
Tasks: 1578 total,   4 running, 1573 sleeping,   1 stopped,   0 zombie
Cpu(s):  7.7%us,  3.2%sy,  0.0%ni, 88.8%id,  0.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  65738108k total, 38075592k used, 27662516k free,     2616k buffers
Swap:  2097144k total,  2095632k used,     1512k free,   768080k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
141358 rsebasti  20   0  172m 118m 2084 R 99.5  0.2 285:25.75 gnuplot
138868 rsebasti  20   0  151m  99m 2092 R 99.2  0.2 274:43.84 gnuplot
143480 rsebasti  20   0 64840  13m 2092 S 78.2  0.0 101:57.97 gnuplot
157153 lpizzaga  20   0 76036  30m  14m R 31.2  0.0   0:00.95 fortcom
...
```

Les processus : connaitre les processus en cours

commande `ps` (pour process status)

permet de lister l'ensemble des processus à un instant donné : `ps ux` ou `ps -ux` ; beaucoup d'options possibles (`man ps`

- par défaut : processus de l'utilisateur lancés depuis le terminale
- `-a` : en plus les processus des autres utilisateurs
- `-x` : processus non rattachés à un terminal
- `-u` : affiche les champs (`PID, %CPU, %MEM ; VSZ ; RSS` (mémoire), `TT` (terminal) , `STAT` (état), ... `COMMAND`)

```
[gueguenm@thor ~]$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
gueguenm    9081  0.0  0.0 109308  4748 pts/31   Ss   10:59   0:00 -bash
gueguenm  158408  0.0  0.0 109308  4712 pts/42   Ss+  14:30   0:00 -bash
gueguenm  159536  0.0  0.0 108340  1160 pts/31   R+   14:32   0:00 ps u
[gueguenm@thor ~]$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
gueguenm    9078  0.0  0.0  94800  1816 ?        S    10:59
0:00 sshd: gueguenm@pts/31
gueguenm    9081  0.0  0.0 109308  4748 pts/31   Ss   10:59   0:00 -bash
gueguenm  120503  0.0  0.0 742848     8 ?        Sl   Apr08
0:00 intelremotemond
gueguenm  158406  0.0  0.0  94800  1804 ?        S    14:30
```

Les processus : gestion des processus

Interruption de processus

- Un processus actif peut être interrompu avec <CTRL-C> (ex : taper top puis <CTRL-C>)

contrôle des processus par les signaux

- un signal peut être désigné par son nom (SIGKILL) ou par son numéro correspondant (9 dans ce cas)
- certains signaux peuvent être transparents pour (certains) processus (ex SIGTERM), d'autres non (SIGKILL)

signal	valeur	description
SIGINT	2	termine le processus (<CTRL-C>)
SIGKILL	9	termine le processus, Arret en force
SIGTERM	15	termine le processus, Arret propre
SIGSTOP	17/19	suspension du processus
SIGTSTP	18/20	suspension du processus par <CTRL-Z>
SIGSEGV	11	référence mémoire invalide

envoi d'un signal : commande kill -SIG PIDs

il faut être propriétaire de son processus !

```
[gueguenm@thor ~]$ kill -9 120503
[gueguenm@thor ~]$ kill -SIGKILL 120503
```

La commande killall fonctionne avec le nom du processus ; la commande pkill permet d'aller plus loin dans certains cas

Pour vos simulations

gestion des processus sur les calculateurs

- les processus liés aux simulations sont gérés directement via un gestionnaire de travaux qui administre les processus et les ressources attribuées
- le contrôle des processus est opéré par le gestionnaire

Nous utilisons sur `THOR` et `HULK` *PBSPro*

gestion des processus sur les calculateurs

- soumission des calculs : `qsub script`
- visualisation de l'état : `qstat`
- arrêt éventuel `qdel JID`

voir `https://forge.univ-poitiers.fr/projects/mesocentre-spin-git/wiki`

script bash

On a vu bash comme interpréteur pour la saisie des commandes dans le terminal. En tant que langage de programmation, on peut l'utiliser directement au sein d'un fichier (script) qui sera interprété par le shell. Pour lancer vos simulations, il faut écrire un script bash pour définir les directives et les ressources.

définition d'une variable

- initialiser une variable : `VAR="Hello World"` (**pas d'espace autour de =**)
- afficher le contenu d'une variable :

```
[gueguenm@thor bin]$ VAR="Hello World"
[gueguenm@thor bin]$ echo $VAR
Hello World
[gueguenm@thor bin]$ echo ${VAR}
Hello World
```

manipuler les variables du système (ex PATH, LD_LIBRARY_PATH)

attention à l'écrasement du contenu d'une variable (ne pas faire `PATH=$HOME/bin`)

- faire `PATH=$HOME/bin:$PATH` ou `PATH=$PATH:$HOME/bin`

script bash

variables et commandes (substitution de commandes)

- récupérer le retour d'une commande dans une variable 2 syntaxes : backquote (‘) ou parenthèses

```
[gueguenm@thor sim_abq]$ nbligne=‘wc -l < /tmp/user.txt‘
[gueguenm@thor sim_abq]$ echo $nbligne
40
[gueguenm@thor sim_abq]$ nbligne2=$(wc -l < /tmp/user.txt)
```

calcul arithmétique : limité (pas de nombres décimaux)

- syntaxe $((<expression>))

```
[gueguenm@thor sim_abq]$ echo $((1 + 1))
2
[gueguenm@thor sim_abq]$ echo $((9/2))
4
```

Ecriture des scripts pour lancer les calculs

Définition

- 1ère ligne : définition du `shell` utilisée
- définition des ressources (lues seulement par *PBSPro* et pas `bash` puisque le # correspond aux commentaires)
- défintion des lignes de commandes nécessaires au lancement du code
- *PBSPro* définit ses propres variables (`$PBS_O_WORKDIR, $PBS_NODEFILE, $PBS_JOBID ...`)

exemple

```
[gueguenm@thor sim_abq]$ cat run_abaqus.pbs
#!/bin/bash
#PBS -N ABQstd_60c
#PBS -l select=3:ncpus=20:mpiprocs=20
#PBS -l place=scatter:excl
#PBS -l walltime=02:00:00
#PBS -j oe
#PBS -l abq_lic=28
#PBS -m abe -M mikael.gueguen@ensma.fr

module load intel-tools-14/14.0.2.144
cd ${PBS_O_WORKDIR}
NCPU=$(wc -l < $PBS_NODEFILE)
abaqus  user=UMAT_singleslipfatigue.f interactive job=agg630oct_500k_hex.inp cpus=$NCPU
  > agg630oct_500k_hex.log.$PBS_JOBID 2>&1
```

conclusion

A retenir

- utilisation des principales commandes `rm, cp, ls, mv` et des "wildcards" ("*", " ?")
- certains utilitaires très pratiques `head, tail, cat`
- les notions de variables (`$HOME, $PATH, ...`, commande `env`)
- les redirections et flux
- les notions sur les processus