

Le Shell pour les poètes

Intro au TAL — cours n° 1

Guillaume Wisniewski
guillaume.wisniewski@limsi.fr
septembre 2019

Université de Paris & LLF



Cette œuvre est mise à disposition selon les termes de la licence Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 4.0 International.

<http://creativecommons.org/licenses/by-nc-sa/4.0/>

1

Inspiration



Une partie du matériel présenté ici se base sur les cours de SOFTWARE CARPENTRY
<https://software-carpentry.org/lessons/>

Le shell, c'est quoi ?

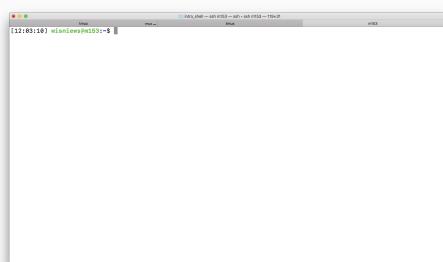


Une manière d'interagir avec un ordinateur

2

3

Le shell, c'est quoi ?



Une manière d'interagir avec un ordinateur

Comme interagir avec un ordinateur ?



Donner des instructions à un ordinateur

- lancer un programme
- supprimer un répertoire
- compter le nombre de mots dans un fichier

Différents types d'interactions

- interface tactile et commande vocale
- *Graphical User Interface (GUI)* commandes données à l'aide d'une souris
- shell

3

4

Pourquoi apprendre à utiliser un shell ?

- répéter des commandes (p. ex.

compter le nombre de mots dans tous les fichiers d'un répertoire)

- combiner des commandes (p.

ex. trouver les 5 fichiers avec le plus grand nombre de mots)

- interagir avec des machines à distance

(

- 1^{re} illustration des principes de la programmation (composer, répéter, ...)

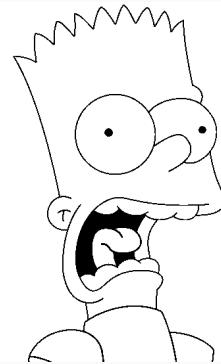
- le shell transparaît dans tous les langages de programmation et dans beaucoup d'usages

)



Mais...

```
1 > wc -l $(find . -name '*.*dat') | sort -n
```



- pas toujours intuitif euphémisme
- beaucoup de notions / commandes à apprendre
- intérêt pas toujours évident pour les opérations dont vous avez l'habitude
- interaction « fragile » : un caractère peut tout changer
⊕ risque d'erreurs irréversibles

6

Comment accéder au shell ?



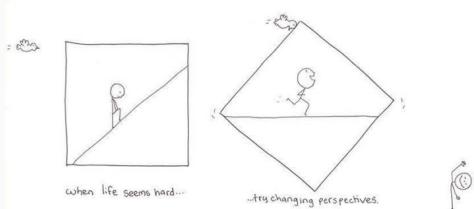
Pour les systèmes dérivés d'Unix

- Linux, MacOS, BSD, ...
- possibilité d'ouvrir un terminal qui contient un shell

Pour les utilisateurs de windows

- il n'est pas trop tard...
- git bash ou cygwin

Changement de perspectives



- shell / terminal = issus des débuts de l'informatique
-
- tous les environnements étaient multi-utilisateurs et partagés

8

Règle d'or n° 1



L' « ordinateur » qui exécute votre commande n'est peut-être pas celui devant vous

Règle d'or n° 2



Plusieurs utilisateurs peuvent « utiliser » le même ordinateur en même temps

9

10

Principe du shell

REPL

1. **R**ead (une commande)
 2. **E**xecute (cette commande)
 3. **P**rint (transmet le résultat à l'utilisateur)
 4. **L**oop (attend la commande suivante)

⇒ exécution interactive

En pratique (1)

Un shell :



- prompt = invite de commande (configurable)

11

12

En pratique (2)

Un shell avec des commandes & leur résultat :



Anatomie d'une commande (1)

1 > 1s

- Une fois la commande exécutée (touche `Enter`) ⇒ affiche le contenu du répertoire courant

13

14

Anatomie d'une commande (2)

1 > ls_-F

- Possibilité d'ajouter des options (**flag**) :
 - attention à l'espace entre le nom de la commande et les options
 - la casse est significative
 - option courte - / option longue --

Anatomie d'une commande (3)

1 > ls_u-F_u/

- possibilité de passer des arguments
 - arguments sont identifiés par leur position
 - ici : liste le contenu du répertoire /
 - commandes, arguments et options sont identifiés par les espaces (ne jamais mettre d'espace dans les noms de fichiers !)

15

16

Pour connaître tous les arguments d'une commande...

La commande magique :

```
1 > man ls
```

ou une version plus succincte :

```
1 > ls --help
```

Les erreurs



A screenshot of a terminal window on a Mac OS X system. The window title is 'Terminal'. The content shows several error messages from the zsh shell:

```
Last login: Sun Jul 28 08:29:02 on ttys002
zsh: command not found: z
zsh: command not found: cd /
zsh: command not found: ls-j
zsh: command not found: ls-j
zsh: illegal option --
usage: ls [-lRSt] [N] [Bcdfghilmnopqrstuwx] [file ...]
```

Les messages d'erreur peuvent varier suivant les configurations

17

18

À vous de jouer



- que fait la commande `ls -l -h` ?
- comment peut-on lister le contenu d'un répertoire et, récursivement, de tous ses sous-répertoires en affichant en premier les derniers fichiers modifiés ?

Résumé

Ce qu'il faut retenir

- objectif d'un shell : lire des commandes & exécuter des programmes
- avantage : automatisation, travail à distance, combinaison de programmes
- inconvénient : courbe d'apprentissage
- commande / arguments / options

19

20

Se déplacer dans un système de fichiers

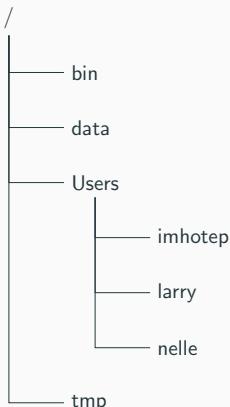
Système de fichiers



- système d'exploitation = programme(s) qui gère l'accès aux ressources
- ressources = stockage, mémoire, CPU, ...
- systèmes de fichiers =
 - fichiers = contient des données
 - répertoires = contient des fichiers ou d'autres répertoires

21

Exemple de système de fichiers



- arborescence / hiérarchie de fichiers
- / = racine répertoire dans lequel sont contenus tous les autres
- en général :
 - répertoires « système » : contient les programmes et données nécessaires au fonctionnement de l'ordinateur
 - répertoire « utilisateur » : contient les données des utilisateurs (plusieurs utilisateurs sur un ordinateur)
 - les répertoires / fichiers peuvent être situés sur une machine distante

22

Un détail d'importance



- Dès que l'on est dans un terminal / shell, on est exactement dans un répertoire du système de fichiers
- . = répertoire courant
- en général lors de l'ouverture → home directory / home / répertoire personnel

23

Chemin d'un fichier

- chemin = représentation d'un fichier / répertoire par une chaîne de caractères
- chemin le plus simple : mon_fichier.txt → désigne le fichier mon_fichier.txt dans le répertoire courant
- chemin « composé » : data/file1.txt ⇒ désigne le fichier file1.txt dans le répertoire data situé dans le répertoire courant
- / = séparateur (permet d'identifier les noms de répertoires / fichiers)
- 3 raccourcis :
 - . = désigne le répertoire courant
 - .. = désigne le répertoire parent du répertoire courant remonter d'un niveau dans la hiérarchie
 - ~ = désigne le répertoire de l'utilisateur (home directory)

24

Chemins relatifs versus chemins absolu



- tous les chemins sont exprimés par rapport au répertoire courant...
- ...sauf quand ils commencent par un / ⇒ exprimé par rapport à la racine du système de fichiers

25

Exercice (1)

Si l'on est dans le répertoire /Users/amanda/data, quels chemins, parmi ceux listés ci-dessous permettent de désigner son répertoire personnel (/Users/amanda/) :

1. .
2. /
3. /home/amanda
4. ../../
- 5.
6. home
7. /data/..
8. ..

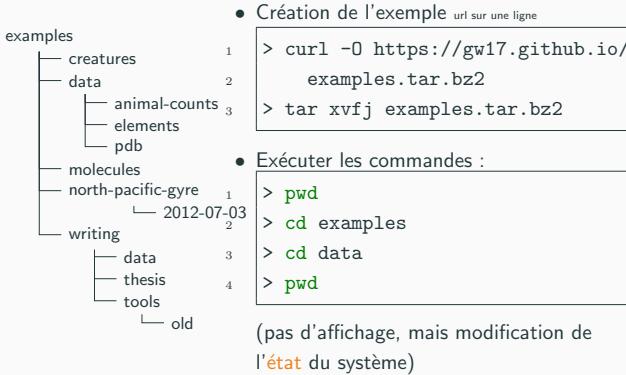
Les commandes pour interagir avec le système de fichiers

- pwd (*print working directory*) : affiche le nom du répertoire courant
- ls : liste le contenu d'un répertoire
 - sans argument : répertoire courant
 - avec argument : contenu du répertoire indiqué en argument
- cd (*change directory*) : permet de changer le répertoire courant
 - sans argument : va dans le *home directory*
 - avec argument : va dans le répertoire indiqué

26

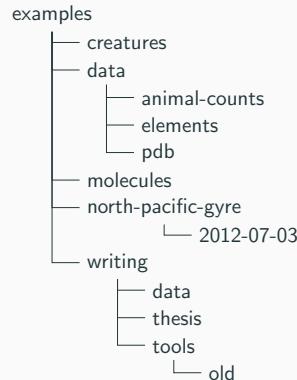
27

Exemple (1)



28

Exemple (1)



- comment revenir un niveau plus haut dans la hiérarchie ?
- comment aller dans le répertoire old ?

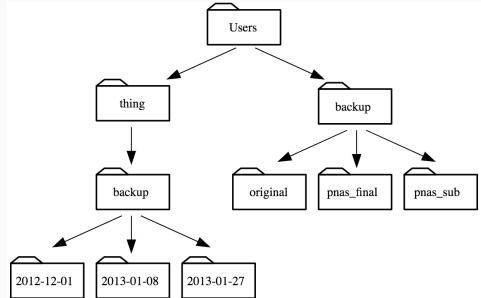
29

Complétion

- possibilité de compléter les commandes / arguments que l'on tape automatiquement
- complétion : à partir d'un préfixe, trouver le suffixe permettant de compléter l'instruction
- Par exemple : ls north-pacific + `[Tab]` → ls north-pacific-gyre
- complétion lorsqu'il n'y a qu'un suffixe possible, sinon affiche une liste des suggestions (possibilité de parcourir les suggestions avec `[Tab]`)

30

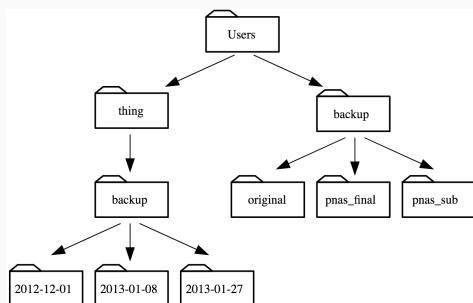
Exercice (2)



En considérant le système de fichiers ci-dessus, si la commande `pwd` renvoie `/Users/thing`, que renverra la commande `ls -F ./backup?`

31

Exercice (3)



En considérant le système de fichiers ci-dessus, si la commande `pwd` renvoie `/Users/backup` et `-r` permet d'inverser l'ordre dans lequel `ls` renvoie son résultat, quelle(s) commande(s) permettent d'afficher le résultat :

32

Résumé (1)

Système de fichiers

- le système de fichiers est chargé d'organiser les informations sur le disque
- les informations sont stockées dans des fichiers stockés dans des répertoires (*folder*)
- un répertoire peut stocker d'autres répertoires formant un arbre
- il est possible d'explorer / se déplacer dans une hiérarchie de fichiers à l'aide des commandes `cd`, `pwd` et `ls`

33

Résumé (2)

Chemins

- / désigne la racine du système de fichiers au début d'un chemin
- un chemin relatif spécifie la localisation d'un fichier à partir de la position courante
- un chemin absolu spécifie la localisation d'un fichier à partir de la racine
- / permet de séparer les répertoires/fichiers dans un chemin \ pour Windows
- . désigne le répertoire courant, .. le répertoire parent

Résumé (3)

Ce que vous devez savoir faire :

- similarités et différences entre fichiers et répertoires
- conversion chemin absolu - chemin relatif
- identifié un fichier donné par son chemin relatif/absolu
- utilisation de la complétion (`Tab`)

34

35

Fichiers & répertoires

Créer un répertoire

```
1 > cd /tmp  
2 > mkdir mon_nouveau_reperatoire  
3 > mkdir /tmp/mon_nouveau_reperatoire/test
```

- `mkdir make directory` pour créer un répertoire
- peut également être fait à partir de l'interface graphique
- arguments : chemin vers le répertoire à créer

36

En pratique



- dans l'exemple précédent comment créer le répertoire `/tmp/mon_nouveau_reperatoire` sans utiliser de chemin absolu ?
- quel est le résultat de la commande `mkdir /tmp/p1/p2/p3?` comment faire pour créer ce répertoire ?

Choisir les noms de fichiers / répertoires (1)

1. n'utilisez pas d'espaces
 - comment identifier les arguments/options lorsque le nom sera utilisé comme argument d'une commande ?
• à la place : utiliser _ ou -
2. ne commencez pas un nom par -
3. n'utilisez que des lettres (sans accent), des chiffres, et les caractères ., - et -
 - tout le reste peut poser des problèmes à l'interpréteur



37

38

Choisir les noms de fichiers / répertoires (2)



La règle d'or
utilisez des noms compréhensibles / décrivant le contenu
Lequel de ces chemins préférez-vous ?

- chat/chien/lamantin
- cours/2019/intro_tal

39

Déplacer des fichiers ou des répertoires

```
1 > cd ~/examples  
2 > mv thesis/draft.txt thesis/quotes.txt
```

- **mv moving** : création d'un nouveau fichier/répertoire ⊕ suppression de l'ancien
- 1^{er} argument : identifie un fichier / répertoire existant
- 2^e argument : destination = nouveau nom / emplacement
- que contiendra le répertoire **thesis** ?
- que fait la commande **mv thesis/quotes.txt** .

40

Particularité

```
1 > mv mon_fichier.txt destination
```



- deux possibilités pour la destination
 - répertoire → déplace la source dans le répertoire
 - fichier → nouveau nom ⊕ déplacement
- si la destination existe déjà, elle est écrasée
 - cf. option **-i/--interactive**
- toutes les opérations sont irréversibles notamment la suppression

41

Dans le même ordre d'idée

- **cp** pour copier un fichier (création à l'identique sans suppression)
- **rm** pour supprimer un fichier
⇒ avec l'option **-r** pour l'appliquer à des répertoires



42

Manipuler plusieurs fichiers / répertoires

Quel est le résultat des commandes suivantes ?

```
1 > cd examples/data  
2 > mkdir backup  
3 > cp amino-acids.txt animals.txt backup/
```

Sélectionner plusieurs fichiers à l'aide de jokers

joker/wildcards

- Dans un nom de fichier / répertoire
 - * → à n'importe quelle suite de caractères (entre 0 et *n*)
 - ? → à (exactement) un caractère quelconque
- le shell **remplacera** le nom avec un joker par l'ensemble des noms correspondant avant l'exécution de la commande

Exemples

- *.pdb → tous les fichiers qui se terminent par .pdb
- ???ane.pdb → tous les fichiers qui commencent par 3 caractères quelconques suivis de ane.pdb p.ex. cubane.pdb ethane.pdb octane.pdb

43

44

Les erreurs



```
/tmp ls
com.apple.launchd.1MK1SeS3XN i_an_a_file
ls: /tmp/i_an_a_file: No such file or directory
/tmp cd tnx-582b
cd: no such file or directory: tnx-582b
/tmp cd i_an_a_file
cd: not a directory: i_an_a_file
/tmp
```

Exercice

Quelle commande produira la sortie

```
ethane.pdb methane.pdb
```

1. ls *t*ane.pdb
2. ls *t?ne.*
3. ls *t??ne.pdb
4. ls ethane.*

45

46

Exercice

```
.   2015-10-23-calibration.txt
   2015-10-23-dataset1.txt
   2015-10-23-dataset2.txt
   2015-10-23-dataset_overview.txt
   2015-10-26-calibration.txt
   2015-10-26-dataset1.txt
   2015-10-26-dataset2.txt
   2015-10-26-dataset_overview.txt
   2015-11-23-calibration.txt
   2015-11-23-dataset1.txt
   2015-11-23-dataset2.txt
   2015-11-23-dataset_overview.txt
   backup
     calibration
       datasets
     send_to_bob
       all_datasets_created_on_a_23rd
         all_november_files
```

En considérant la hiérarchie ci-contre, donnez les commandes qui permettront de :

- copier tous les *datasets* dans le répertoire *backup/datasets*
- idem pour les *calibrations*
- copier tous les fichiers de novembre dans le répertoire *all_november_files*
- copier tous les fichiers créés le 23 novembre dans *all_datasets_created_on_a_23rd*

47

Pipes et filtres

Objectif

Ce que nous allons voir :

- comment combiner des commandes entre elles
 - deux notions essentielles du shell
 - redirection
 - création de pipelines
 - Philosophie Unix : *small pieces, loosely joined*
- ⇒ un des principaux intérêt du shell



Comptons...



```
1 > wc au_bohneur_des_dames.txt
2 19046 152638 974447 au_bo
```

- affiche le nombre de lignes, de mots et de caractères dans un fichier
- comment afficher uniquement le nombre de mots ?

48

49

Remarque

- Que fait la commande :



```
1   wc -l
```

- pas de paramètres ⇒ contenu donné interactivement
- en pratique : comme si le programme était bloqué
- pour sortir **Ctrl + C**
- sinon : pipe (voir la suite)

50

Exemple (1) : le contexte

- collections de livres écrit par Zola (projet Gutenberg)

```
zola_books ls
au_boheme_des_dames.txt    la_bete_jusquine.txt      le_ventre_de_paris.txt
au_boheme_des_dames.txt      la_cite_des_mille_villes.txt
doctor_beaucal.txt          la_faute_de_l_abbe_mouret.txt  mon.txt
doctor_beaucal.txt          la_terre.txt            pot_bouille.txt
l_argent.txt                la_soie.txt            thomas_rapin.txt
laveuses.txt                le_douceur_pascal.txt
l_oeuvre.txt                le_reve.txt            une_nuit_d_amour.txt
zola_books
```

51

Pour récupérer les données...

```
1 > curl -O gw17.github.io/zola_books.tar.bz2
2 > tar xvzf zola_books.tar.bz2
```

Exemple (2) : explorer le contenu des fichiers



Que permettent de faire les commandes suivantes ?

- head
- tail
- less
- cat

(avec un nom de fichier en argument)

53

Exemple (3) : livre le plus long



Comment :

- afficher le nombre mots de chaque fichier ?
- trouver le livre le plus long (avec le plus de mots) ?

Solution < naïve >

Une seule commande...

```
1 > wc -w *
```

et on regarde à la main !

Mais...

- que faire s'il y a 6 000 fichiers ?
- risque d'erreur élevé

54

55

Grâces soient rendues au shell (1)

Redirection

```
1 > wc -l * > lengths.csv
```

- résultat de la commande stocké dans le fichier lengths.csv

Avec la commande qui va bien

```
1 > sort -n lengths.csv
```

- que se passe-t-il si on ne passe pas l'option -n ?

Grâces soient rendues au shell (2)

```
1 > wc -l * > lengths.csv
2 > sort -n lengths.csv > sorted_lengths.csv
3 > head -1 sorted_lengths.csv
```

- solution complètement automatisée
- mais création de plusieurs fichiers intermédiaires !

56

57

Quelques remarques



- ne jamais rediriger un fichier vers lui-même
- ```
1 > sorted -n b.txt > b.txt
```
- ⇒ comportement indéterminé
- la redirection << écrase >> le fichier existant
  - >> permet d'ajouter la sortie d'une commande à la fin d'un fichier existant

58

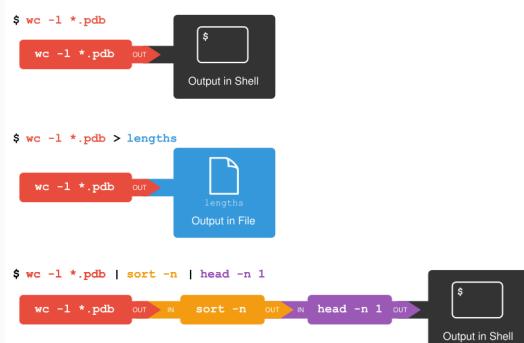
## Solution sans fichier intermédiaire

```
1 > wc -l *.txt|sort -n|head 1
```

- | = pipe
- A|B l'entrée de la commande B est la sortie de la commande A

59

## Résumé en images



60

## Exercice (1)

- quel est le fichier le plus petit (c.-à-d. qui contient le moins de mots) ?
- combien y a-t-il de fichiers dans le répertoire zola\_books ?

61

## Exercice (2)

Considérons le fichier animals.txt contenant :

```
2012-11-05,deer
2012-11-05,rabbit
2012-11-05,raccoon
2012-11-06,rabbit
2012-11-06,deer
2012-11-06,fox
2012-11-07,rabbit
2012-11-07,bear
```

Que contient le fichier animals-subset.txt créé par les commandes

```
1 > head -n 3 animals.txt > animals-subset.txt
2 > tail -n 2 animals.txt >> animals-subset.txt
```

## Exercice (3)

Parmi les commandes suivantes, lesquelles permettent de trouver les 3 fichiers du répertoire courant contenant le plus petits nombres de lignes ?

1. wc -l \* > sort -n > head -n 3
2. wc -l \* | sort -n | head -n 1-3
3. wc -l \* | head -n 3 | sort -n
4. wc -l \* | sort -n | head -n 3

## Exercice (4)

Considérons le fichier animals.txt dont le contenu est :

```
2012-11-05,deer
2012-11-05,rabbit
2012-11-05,raccoon
2012-11-06,rabbit
2012-11-06,deer
2012-11-06,fox
2012-11-07,rabbit
2012-11-07,bear
```

Quel est le résultat de la commande :

```
1 > cat animals.txt | head -n 5 | tail -n 3 | sort -r > fina
```

## Exercice (5)

On aimerait compter le nombres d'animaux différents apparaissant dans le fichier animals.txt. cf. transparent précédent

On pourra utiliser les commandes :

- uniq qui permet de supprimer les lignes identiques consécutives
- cut qui permet d'extraire des parties de lignes

## Boucle for

## Principe

- boucle for = structure de contrôle
- appliquer un traitement à chaque élément d'un ensemble

```
1 for thing in list_of_things
2 do
3 # Indentation within the loop is not required,
4 # but aids legibility
5 operation_using $thing
6 done
```

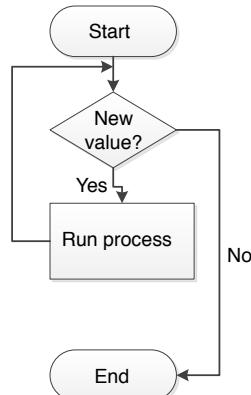
## Exemple

```
1 $ for filename in basilisk.dat minotaur.dat unicorn.dat
2 > do
3 > head -n 2 $filename | tail -n 1
4 > done
```

ou de manière plus compacte :

```
1 $ for filename in *.dat
2 > do
3 > head -n 2 $filename | tail -n 1
4 > done
```

## Ce qu'il se passe



La variable \$filename prendra successivement les valeurs :

1. basilisk.dat
2. minotaur.dat
3. unicorn.dat

et exécutera la commande successivement pour chaque valeur

## Même symboles, sens différents

- le signe > permet d'indiquer la **continuation** d'une même commande
- filename est une **variable**
- une fois la variable déclarée, on accède à sa valeur par \$nom ou \${nom}
- nom de variable arbitraire
- variable = **alias** pour une valeur

## Exercice

Quelle est la sortie des deux commandes suivantes :

```
1 $ for datafile in *.pdb
2 > do
3 > ls *.pdb
4 > done
```

et :

```
1 $ for datafile in *.pdb
2 > do
3 > ls $datafile
4 > done
```

## Utilisation de la variable

- pour créer un nouveau nom

```
1 $ for filename in *.dat
2 > do
3 > cp $filename original-$filename
4 > done
```

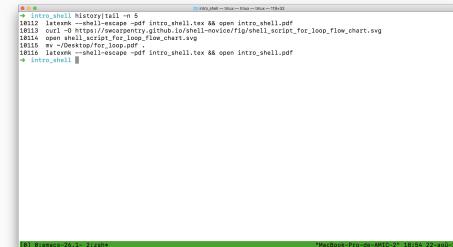
## Une bonne pratique



- les commandes shell sont irréversibles
- avant d'exécuter une boucle → *dry run*
- typiquement : echo cmd

## Ceux qui connaissent l'histoire peuvent la répéter

### La commande history



```
intro_shell$ history -5
10112 latex --shell-escape -o intro_shell.tex && open intro_shell.pdf
10113 curl -O https://raw.githubusercontent.com/igiselle/novice/1/g/shell_for_loop_flow_chart.svg
10114 open shell_for_loop_flow_chart.svg
10115 mv ~/Desktop/for_loop.pdf ~
10116 latex --shell-escape -o intro_shell.tex && open intro_shell.pdf
intro_shell$
```

- permet d'accéder à l'ensemble des commandes qui ont été exécutées
- possibilité de naviguer dans l'historique avec les flèches  et 

73

### Autres commandes

-  +  permet de rechercher une commande de l'historique (en complétant les lettres tapées)
- !! permet de ré-exécuter la dernière commande exécutée
- !\$ est remplacé par le dernier mot exécuté

74

### Chercher & Trouver

### Filtrer le contenu d'un fichier

- La commande grep permet de n'afficher que les lignes d'un fichier dans lesquelles apparaît un **motif** donné



```
examples$ cat haiku.txt
The tree is seen
Is not the tree too until
You bring fresh tones.

With searching comes loss
Is not the tree too until
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that
Is not the tree too until
"Thesis" not found.
Today It is not working
examples$
```

### Les options de grep

- Que se passe-t-il si on cherche le motif The dans le fichier précédent ?



75

76

## Les options de grep



- Que se passe-t-il si on cherche le motif The dans le fichier précédent ?
- Par défaut grep respecte la casse (-i chercher aussi bien the, ThE, ...)
- Par défaut grep cherche le motif n'importe où dans la ligne (y compris à l'intérieur d'un mot plus grand) → -w pour chercher les mots entiers

76

## Les options de grep



- Que se passe-t-il si on cherche le motif The dans le fichier précédent ?
- Par défaut grep respecte la casse (-i chercher aussi bien the, ThE, ...)
- Par défaut grep cherche le motif n'importe où dans la ligne (y compris à l'intérieur d'un mot plus grand) → -w pour chercher les mots entiers
- beaucoup d'autres options

76

## Exemple

On dispose d'un répertoire contenant des fichiers structurés de la manière suivante :

```
2013-11-05,deer,5
2013-11-05,rabbit,22
2013-11-05,raccoon,7
2013-11-06,rabbit,19
2013-11-06,deer,2
```

Écrire un script qui prend en argument le nom d'une espèce species et le nom d'un répertoire et qui produit un fichier species.txt qui contient une liste de dates et le nombre de spécimens de cette espèce vus à cette date. Appelé avec rabbit en argument, il produira :

```
2013-11-05,22
2013-11-06,19
```

## And the winner is...

Contenu du script :

```
1 grep -w $1 -r $2 | cut -d : -f 2 | cut -d , -f 1,3 > $1.txt
```

Exemple d'appel :

```
1 > bash count-species.sh bear .
```

## Exercice

1. Combien de lignes ne contiennent pas le mot the ?
2. Quelle est la sœur qui est mentionnée le plus souvent dans le livre *Little Women* de Louisa May Alcott ?
  - le livre raconte l'histoire de quatre sœurs : Jo, Meg, Beth et Amy
  - le livre est accessible  
<http://www.gutenberg.org/ebooks/514>
  - vous pouvez utiliser une boucle

## De l'intérêt du TAL (1)



Peut-on utiliser la méthode précédente pour compter les mentions des deux derniers PDG d'Apple ?

79

80

## De l'intérêt du TAL (2)



Tim Cook (PDG actuel)



Steve Job

- PDG actuel et précédent d'Apple ⇒ noms fortement polysémique
- ⊕ grande variabilité (Mr Cook, Tim Cook, The president Cook, ...)
- un grep même compliqué ne peut fournir qu'une estimation du nombre d'occurrences

## La commande find



- La commande find permet de chercher/trouver les fichiers
- utilisation la plus simple  

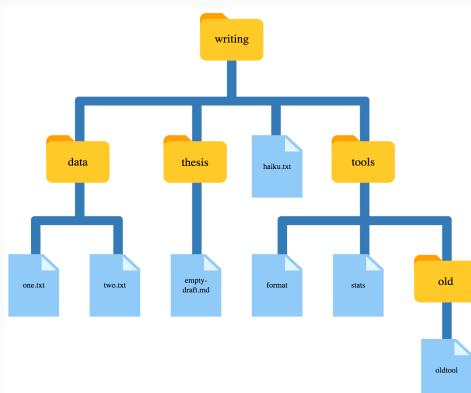
```
> find .
```

⇒ affiche tous les fichiers et répertoires contenus dans le répertoire courant et tous ses enfants
- nombreuses manières de filtrer

81

82

## Exemple (1)



## Exemple (2)

```
1 > find .
2 .
3 ./data
4 ./data/one.txt
5 ./data/littleWomen.txt
6 ./data/two.txt
7 ./tools
8 ./tools/format
9 ./tools/old
10 ./tools/old/oldtool
11 ./tools/stats
12 ./haiku.txt
13 ./thesis
14 ./thesis/empty-draft.md
```

83

84

## Exemple (3)

```
1 > find . -type d
2 .
3 ./
4 ./data
5 ./thesis
6 ./tools
7 ./tools/old
```

⇒ filtre selon le type (d pour les répertoires et f pour les fichiers)

## Exemple (4)

```
1 > find . -name *.txt
2 .
3 ./haiku.txt
```

⇒ filtre sur les noms de fichiers

85

86

## Combiner les commandes

- find permet de sélectionner un ensemble de fichiers
- comment appliquer un traitement à chacun de ces fichiers ?

```
1 > wc -l $(find . -name '*.txt')
```

- le shell exécute la commande entre parenthèses et remplace celles-ci par le résultat de la commande
- comme pour les expansions de fichiers (\*.txt)
- équivalent à

```
1 > wc -l ./data/one.txt ./data/LittleWomen.txt
2 ./data/two.txt ./haiku.txt
```