

Marc Chantreux

- ▶ Responsable d'application à la direction informatique de l'université de Strasbourg
- ▶ unixien depuis 1996

aspects pratiques

- ▶ les horaires?
 - ▶ de 9h à midi
 - ▶ de 13h à 17h
- ▶ les comptes utilisateurs sont détruits après la formation

le rythme

- ▶ nous ne finirons pas . . . et vous serez fatigués
- ▶ je tente d'aller **vite** sans **perdre personne**
 - ▶ n'hésitez pas à me demander de ralentir
 - ▶ quand je demande si tout va bien, **ca n'est pas rhétorique**
 - ▶ n'hésitez jamais à m'interrompre
 - ▶ **ne vous laissez pas distancer**

les exos ne sont pas tous à réaliser, j'encourage

- ▶ les devoirs à la maison
- ▶ le travail en groupe (**binôme**) et le partage d'expérience
- ▶ l'**experimentation**
 - ▶ ne vous bornez pas aux exercices
 - ▶ testez des que vous doutez

évaluation finale

- ▶ prenez des notes
- ▶ pas de cadeaux

- ▶ la famille de systèmes d'exploitations (macOS, les BSD, linux, android, ...)
- ▶ culture (des symboles, des pratiques communes)
- ▶ des “standards” (POSIX, ...) et différents des degrés d'adoption

chaque diapo ...

- ▶ j'ai du faire des choix (pratiques courantes, intéressantes, ...)
- ▶ être exhaustif est inutile est chronophage (RTFM)

buts de cette formation

- ▶ comprendre et intégrer les bases culturelles par l'utilisation par la manipulation des outils standard.
- ▶ acquérir les bases pratiques permettant de réaliser et automatiser des tâches simples sous Unix grâce à des outils standard (si possible tels que définis par →**POSIX**).

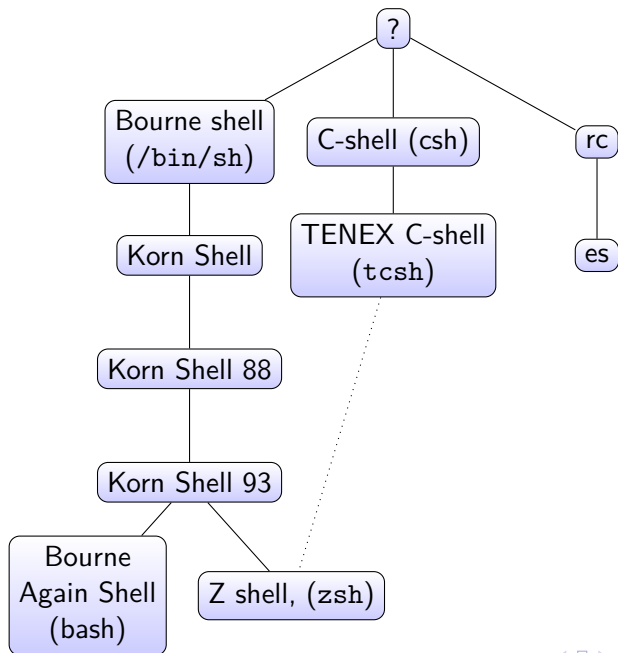
3 jours de modernité (distribution, parallélisme, ...) dans un monde sans souris: bienvenu dans “le terminal”

“le terminal” ?

outils	fonction
putty	émulateur de terminal
	client ssh
zsh	shell
tmux	un multiplexeur de terminaux

outils	fonction
editeur de texte	vi
fichiers	navigation, manipulation, edition
processus	les I/Os
filtres	cut, join, cmp, grep, sed, awk, m4
build automation	make

shells



exemple de différence syntaxique

sh, ksh, bash, zsh

```
for t in *txt; do
    for i in indexes*; do
        echo "$t $i"
    done
done
```

zsh

```
for t (*txt)
    for i (indexes*)
        echo "$t $i"
```

shells actuels

- ▶ minimalistes: strict respect du standard
- ▶ fonctionnels: ajout de fonctionnalités utiles (completion, coloration syntaxique, ...)

dialecte	implementation	orientation	par défaut sous
sh	dash	minimaliste	/bin/sh (certains linux)
ksh	pdksh	minimaliste	ksh (certains bsd)
ksh	mksh	fonctionnel	ksh (certains bsd)
sh,ksh	bash	fonctionnel	/bin/sh (certains linux)
sh,ksh,csk	zsh	fonctionnel	/bin/sh et /bin/ksh (certains linux)

connexion

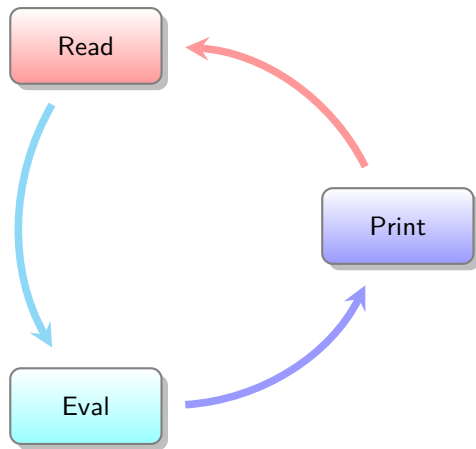
live demo avec putty ... mais depuis unix:

```
ssh user@machine
```

```
ssh user@machine echo hello world
```

```
seq 5 | ssh user@machine grep 2 | tr 2 X
```

Read, Eval, Print Loop (REPL)



en pratique

le prompt (\$PS1)

```
zsh>
```

```
zsh>
```



```
zsh> echo hello world
```



```
hello world
```

```
zsh> echo welcome to "the shell"
```



```
welcome to the shell
```

```
zsh> echo bye; echo world
```












```
bye
```

```
world
```

```
zsh>
```



les touches essentielles

	,		déplacer le curseur dans l'instruction
			supprimer un caractère
			valider les instructions
	,		naviguer dans l'historique
	+		quitter
			completion (contextuelle?)

- ▶ être exhaustif est chronophage et inutile
- ▶ dépend du shell
- ▶ configurable et extensible
- ▶ paramètrages par défaut inspirés de emacs et vi

conventions typographiques

vous tapez

```
zsh> echo bye; echo world
```



```
bye
```

```
world
```

sur la diapositives

```
z> echo bye; echo world
```

```
bye
```

```
word
```

et même

```
echo bye; echo world
```

pager, documentation

less, apropos, man

les systèmes de fichiers

- ▶ on appelle **filesystem (fs)** un système représentant une **collection hiérarchique d'objets** sous la forme de fichiers et de répertoires. (partitions de disque dur, clef usb, vues SQL, table de processus, ...).
- ▶ tout répertoire vide peut servir de **point de montage** pour un autre fs.
- ▶ le premier point de montage est appelé **racine** (noté /)

exemples de fs

- ▶ fs de partitions: ext4, zfs, fat32, ntfs, ufs
- ▶ fs spéciaux: sshfs, procfs, sysfs, ramfs, fuseiso, Fuse::DBI
- ▶ fuse et 9p permettent d'exposer tout type de données dans un fs en implémentant des **syscalls** (open, walk, read, write, ...)

les fichiers

- ▶ les **fichiers** sont des unités contenant un ensemble cohérent d'information.
 - ▶ sa structure interne est appelé **format**
 - ▶ l'organisation de données dans ce format est appelée **sérialisation**
 - ▶ l'opération inverse est appelée **désérialisation**
- ▶ des méta-informations lui sont associés
 - ▶ le owner, le groupe, les droits (ls -l)
 - ▶ les dates de dernier accès, modification et changement (stat)
- ▶ le tout est regroupé au sein d'une structure appelée **inode** associé par une valeur numérique unique dans son fs.

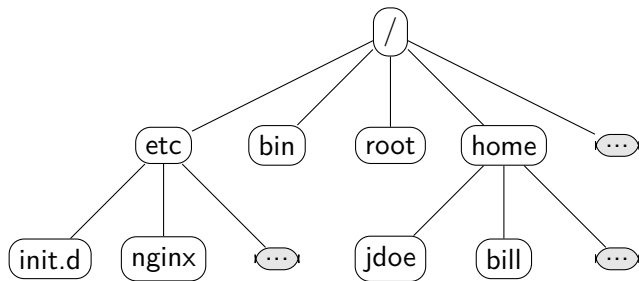
les répertoires

- ▶ les répertoires sont des collections pouvant contenir des fichiers et des répertoires (appelés **sous-répertoire**)
- ▶ le contenant est appelé **parent** et le contenu **enfant**.

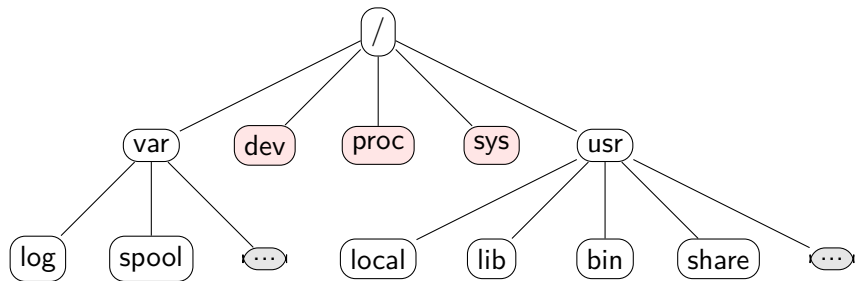
exemple d'arborescence

- ▶ les arborescences sont très variables d'un unix à l'autre
- ▶ sous linux, → **FHS (Filesystem Hierarchy Standard)** peut donner des indications

arborescence linux (1)



arborescence linux (2)



propriétés et droits

- ▶ un utilisateur est membre d'un groupe principal (facultativement de groupes secondaires)

navigation

- ▶ tout processus possède un **repertoire courant**
- ▶ / est (aussi) le séparateur dans la notation des chemins
- ▶ les fichiers dont le premier caractère du nom est . sont **cachés** par convention
- ▶ tout répertoire possède deux sous-répertoires spéciaux: . (lui-même) et .. (son parent)
- ▶ les chemins sont des notations des noms des ressources présentes dans l'arborescence
- ▶ tout utilisateur possède un répertoire de travail: son **home directory**

inspecter le répertoire courant

```
pwd
```

```
ls
```

```
ls -A # tous les fichiers
```

```
ls -d # le répertoire, pas son contenu
```

```
ls -l # informations supplémentaires
```

```
ls -sh # tailles
```

naviguer (chemins absolus)

navigation depuis la racine

```
cd /
```

```
cd /home/mc
```

```
cd /var/log
```

naviguer (les homes)

```
cd ~      # home  
cd        # home  
cd ~jdoe  # home de jdoe
```

naviguer (chemins relatifs)

```
# /var/log ou /usr/local/var/log ou ...  
cd ~jdoe/../../var/log  
cd ~/..    # répertoire parent du home
```


naviguer (chemins relatifs au répertoire courant)

```
cd .      # répertoire courant
cd ./foo  # sous-répertoire "foo" du répertoire courant
cd foo    # idem
cd foo/bar # (courant) → foo → bar
cd ..     # répertoire parent
cd ../..  # répertoire parent
```

navigation (chemins relatifs au répertoires courants précédents)

un historique des répertoires peut être maintenu avec pushd et popd

```
cd bar
pushd foo    # push bar dans l'historique
              # cd dans foo
pushd pan    # push bar dans l'historique
              # cd dans pan
popd         # retour dans foo
popd         # retour dans bar
```

setopt autopushd

```
cd -          # pénultième courant  
cd -1         # pénultième courant  
cd -2         # anté-anté-pénultième courant  
cd -...
```

cd est inutile

```
cd ~
```

```
ls
```

```
ls ~
```

- ▶ permettent de filtrer une collection sur la base d'un **motif** qui doit correspondre à **l'intégralité du nom**.
- ▶ en zsh, `setopt extendedglob` permet
- ▶ d'interroger l'age, la nature, ...
- ▶ de filtrer sur le nom
- ▶ de modifier la chaine résultante

introduction

se référer aux manuels

- ▶ `ksh` “File name patterns”
- ▶ `zshexpn` “FILENAME GENERATION”

les symboles

- ▶ les symboles se représentent eux-même (a, b, -, 0, 9, ...)
- ▶ * correspond à une suite de 0 à n symboles

les métacaractères

- représentent un symbole pouvant avoir différentes valeurs

?	un symbole (a, b, -, 0, 9, ...)
[abc]	a, b ou c
[a-f]	une lettre comprise entre a et f
[5-9]	un chiffre compris entre 5 et 9
[^a-f]	tout sauf [a-f]

les groupes

`(foo|bar)` 'foo' ou 'bar'

les quantifieurs

$A##$	au moins une répétition du A	1 à n
$A\#$	peut-être des répétitions de A	0 à n

les exclusions

$\neg A$	ne satisfait pas A
$A \sim B$	satisfait A et pas B
$A \sim B \sim C$	satisfait A et pas B et pas C

les modifieurs

#b active la capture des groupes (`$matches`)
#i insensible à la casse

parcours de répertoire

- ▶ le '/' permet de chercher dans des sous-répertoires, chaque motif est à lire séparément
- ▶ ** entraine un parcours récursif

examples

```
alias l='print -l'  
l */(bin|scripts)/col*[1-9]  
mv ^old old
```

stockage et interpolation de motif

```
images='*.*#i(jpe#g,png,svg,gif)'  
l $~images
```

Glob Qualifiers

- ▶ des microphrases entre parenthèses à la fin du motif
- ▶ remplace `find` dans de nombreux cas

Glob Qualifiers (exemple)

- ▶ les fichiers uniquement .
- ▶ dont l'extension est .data
- ▶ créés il y a
- ▶ plus de 10 heures ch+10
- ▶ mais moins de 5 jours cd-5
- ▶ appartenant à matts u:matts:
- ▶ dont la taille est supérieure à 5Go
- ▶ triés par taille oL
- ▶ n'afficher que le basename :r

```
*.data(.ch+10cd-5u:matts:L+5goL:r)
```

hello world

```
echo hello world  
print hello world
```

```
echo    l'original, passe partout  
print   seulement zsh, avec des ajouts intéressants
```

- ▶ les deux fonctionnent
- ▶ j'ai tendance à préférer print

les instructions

```
print      -1  -0  lorem  ipsum  dolor  
commande
```

les instructions

```
print  -1      -0  lorem  ipsum  dolor
            
      flags
```

les instructions

```
print  -1          -0  lorem  ipsum  dolor
      parameters
```

les instructions (en résumé)

une instruction permet d'évoquer une commande qui accepte d'éventuels arguments (parameters). ces composés sont séparés par un ou plusieurs espaces horizontaux (`space` et `tab`) (les espaces en début et fin de ligne sont insignifiants).

certain paramètre (les flags) paramètre le traitement des données

les instructions sont séparées par des ; ou des retours à la ligne

les flags

- ▶ paramètre préfixé d'un - qui apparaît avant --
- ▶ peuvent se regrouper

regrouper les flags

```
print -l -O hello world  
print -lO hello world
```

-- sépare les flags des autres paramètres

```
z> print -l -kilo-
```

```
zsh: bad option: -k
```

```
z> print -l -- -kilo-  
-kilo-
```

commentaires

introduit un commentaire

```
z> print hello world # commentaires (donc sans effet)  
hello world
```


expansions

avant execution de la commande, le shell cherche des motifs spéciaux dans les paramètres et les remplace par leur expansions.

expansions

globs	motifs de recherche de fichiers ou de comparaison de chaîne
~ (tilde)	~ repertoire de travail de ...
{,} {-}	distribution (produit cartésien)
=	equivalent de \$(which)

```
print jean-{paul,luc}
print {jean-,}{paul,luc}
print {5,9}
print {5-9}
print -l {5,6}x{1,3}
print -l {5,6}x{1-3} # setopt braceccl
print ~{nobody,root,www-data}
print ={sh,zsh} # setopt extendedglob
print ~
print ~mc
```

protection des caractères

- ▶ préfixer les caractères avec le symbole \ (backslash) empêche son expansion. (on parle de protection du symbole)
- ▶ un ensemble de symboles entourés par des ' ou des "
- ▶ contrairement à ', "
 - ▶ respecte l'interpolation
 - ▶ permet l'introduction de caractères d'échappement ("`\n`", "`\t`", ...)

exemples

```
print l\'apostrophe,\n\\{l\'accolade}, le \\~tilde et même\\  
le retour à la ligne
```

```
print "l'apostrophe,  
{l'accolade}, le ~tilde et même  
le retour à la ligne"
```

```
print \\  
print "c:\\> \\n\\n\\n"
```

variables

déclaration

```
greetings='hello world'  xp=15
```

interpolation

- ▶ \$ provoque l'interpolation de variable
- ▶ les variables peuvent être
- ▶ exportés lors de la création d'un processus (variables d'environnement)
- ▶ déclarées dans le shell courant
- ▶ spéciales, elles correspondent à l'état de l'interpreteur
- ▶ peut s'effectuer sur la commande

exemples

```
print $USER # variable d'environnement
n=5 b=6
c=\n d=\$n e=$n
print "n=$n ( $c $d $e )"
print 'n=$n ( $c $d $e )'
```

```
cmd=print
$cmd hello
```

```
to=play          # déclaration de variable
print $tomate
```

pourquoi?

setopt nounset

```
setopt nounset  
print $tomate
```

zsh: tomate: parameter not set

```
print ${to}mate
```

manipuler les variables

- ▶ être exhaustif est chronophage
- ▶ man ksh et zshall (chercher %%)

```
print ${email:=jdoe cs unistra fr}  
email=${email/ /@}  
email=${email// /.}  
print email      : $email  
print domain    : ${email#*@}  
print user      : ${email%@*}
```


zsh

nesting

```
email=${${${fake:=jdoe cs unistra fr}/ /@} // /.}  
print email $email from fake $fake
```

modifiers

```
print ${(U)email}
```

vim

- ▶ au commencement était ed
- ▶ puis sed, ex, vi, sam, les clones et améliorations de vi (nvi, elvis, yi, izis ...)
- ▶ →vim pour “vi iMproved”, remplaçant de facto
- ▶ →neovim possible remplaçant

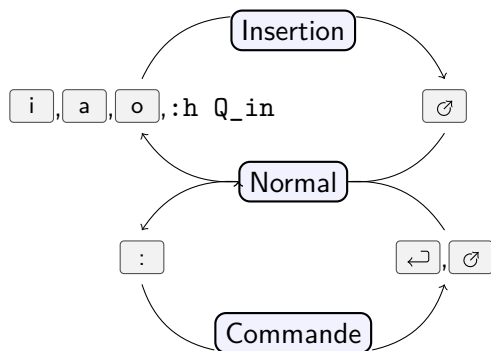
apprendre vim comme on apprend une langue

- ▶ une grammaire, un vocabulaire
- ▶ voir l'aide comme un dictionnaire
 - ▶ ne pas vouloir tout lire
 - ▶ apprendre à trouver l'info
- ▶ tout le monde commence en parlant le “petit-vi”, ne pas se décourager

définitions







les buffers	des zones éditables (traditionnellement un fichier)
les fenêtres	des zones de l'écran affichant un buffer
la barre de status	permet d'afficher des informations
la ligne de commande	permet de taper des commandes
les modes	conditionnent le fonctionnement de vi

les principaux modes



le mode commande

comme dans le shell

- ▶  (on note <tab>) pour compléter
- ▶  (on note <cr>) pour valider
- ▶  et  (on note <up> et <down>) pour naviguer dans l'historique
- ▶  +  pour éditer l'historique

le mode commande (les incontournables)

w	enregistrer
q	quitter
h	aide en ligne
qa	tout quitter
!	comme suffixe pour forcer

- ▶ :wq! le contenu du buffer courant et quitte
- ▶ :wq! le contenu du buffer courant et quitte
- ▶ :w! /tmp/foo le contenu du buffer courant dans /tmp/foo et quitte
- ▶ **pas d'espace avant le !**, la signification serait tout autre

:h >>> stack overflow

- ▶ la commande `help` (ou `hel` ou `he` ou `h`) ouvre une fenêtre avec la documentation.
- ▶ si `wildmode=longest,list`, `tab` déclenche la completion.
- ▶ les bonnes lectures
 - ▶ `:h`
 - ▶ `:h quickref`
 - ▶ `vimtutor` (depuis le shell)

vim --help, le best of

```
vimdiff new old      # diff entre deux fichiers
vim scp://r/.zshrc    # machine distante (ou sshfs)
# les deux ensemble
vimdiff {scp://r,~}/.zshrc
vim                  # votre GUI a tout faire
vim fichier.md +5    # fichier.md ouvert,
vim *.md
ls | vim -           # utiliser vim comme filtre
vim - < /var/log/mail.log
```

la grammaire du mode normal

- ▶ des phrases impératives
- ▶ des compléments (mouvements ou blocks)
- ▶ des adjectifs numéraux
- ▶ des verbes
 - ▶ le verbe par défaut est “déplacer le curseur”
 - ▶ lorsque le verbe est doublé, le complément est “toute la ligne”

les mouvements classiques

aide: Q_lr, Q_ud

h,l	caractère a gauche,droite
j,k	ligne supérieure,inférieure
^,0,\$	début, premier caractère significatif, fin de la ligne
{,}	début, fin de paragraphe
Fx,fx	prochain caractère x à gauche,droite
Tx,tx	juste avant prochain caractère x à gauche,droite

les verbes classiques

	aller à
y	copier jusqu'à
c	changer jusqu'à
d	supprimer jusqu'à
gu	lowercase jusqu'à
gU	uppercase jusqu'à
p	coller (puts)

exemples de phrases

yy 5p c\$ d3w **dd** dj y2j p ct,

le mode commande

yy 5p c\$ d3w **dd** dj y2j p ct,

les registres

les marques

exercice, editez votre fichier ~/.vimrc

```
" interactivité  
set hlsearch incsearch
```

recherche et motifs