Chapitre 2 : Introduction au shell

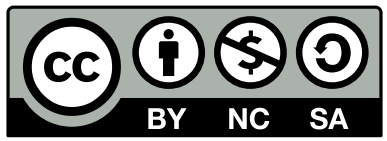
INF1070

Utilisation et administration des systèmes informatiques

Jean Privat & Alexandre Blondin Massé

Université du Québec à Montréal

v243



Plan

1. Shell
2. Manuel en ligne
3. Éditeur de texte
4. Commandes et arguments
5. Redirection et tube
6. Développement et caractères spéciaux du shell

Shell

Shell vs. terminal

Shell: le programme qui **interprète les commandes**

Le shell lit des commandes et les exécute

- `bash` (*Bourne-Again shell*) de GNU (1989) — le plus commun
- `ash` (*Almquist shell*) (1989) — minimaliste (embarqué, scripts)
- `dash` (*Debian Almquist shell*) (2002) — conforme à POSIX et dérivé de ash
- `PowerShell` de Microsoft (2006)

Terminal: l'**interface** physique (ou virtuelle)

Un terminal c'est un clavier et un écran texte

- Fenêtre de terminal (émulateur de terminal)
  Exemples: `xterm`, `gnome-terminal`, `konsole`, `iTerm2`, `cmder`
- Autre pseudo-terminal. Exemples: `ssh`, `screen`, `tmux`
- Terminal physique. Exemple: VT100 (de plus en plus rare)

Par défaut, l'émulateur de terminal exécute un shell

Pourquoi le shell ?

- Toujours présent
- Relativement portable (norme POSIX et standard de fait `bash`)
- Versatile
- Économe et fonctionne en réseau
- Automatisable (scriptable)
- Parfois le seul moyen pour des usages avancés

Un des objectif du cours: savoir utiliser **efficacement** le shell

Fonctionnement du shell

Le shell est un programme comme les autres

1. Affiche l'invite de commande
2. Lit la commande de l'utilisateur
3. Analyse la ligne de la commande et ses caractères spéciaux
4. Exécute la commande
5. Recommence au point 1

Quoi faire avec le shell?

- Naviguer dans et interagir avec le système de fichiers
- Exécuter et contrôler des commandes et des utilitaires
- Développer des petits programmes (scripts shell)

Invite de commande shell

Invite (*prompt*): indique que l'utilisateur peut entrer des commandes

```
privat@lama:~/ens/INF1070$
```

- Nom de l'utilisateur: `privat`
- Nom de la machine: `lama`
- Répertoire courant: `~/ens/INF1070`
- Indicateur d'utilisateur (à la fin)
  - « `$` » (utilisateur normal)
  - « `#` » (super-utilisateur)
- Invite secondaire: « > » quand le shell a besoin de plus de saisies

L'invite est configurable

Saisie de commandes shell

Chaque commande saisie est autonome

- ⏎ exécute la commande
  Le shell affiche un message d'erreur si la commande est invalide
- ← et → naviguent dans la ligne
- ↑ et ↓ naviguent dans l'historique
- Plein de petites fonctionnalités pratiques (ça dépend du shell)

```
$ echo Bonjour le monde
Bonjour le monde
$ uptime
 14:07:42 up 1 day, 21:34, 1 user
 load average: 1,65, 1,81, 1,76
$ cmatrix
$ xeyes
```

Quelques commandes de base

- `echo` — afficher un message
- `ls` — lister le contenu du répertoire
- `cat` — afficher un fichier
- `cd` — changer de répertoire
- `cd ..` — revenir au répertoire parent
- `cd -` — retourner au dernier répertoire actif
- `pwd` — afficher le nom du répertoire courant
- exit — fermer le shell

```
$ ls
hello.txt lisez-moi.txt  répertoire
$ cat lisez-moi.txt
Bash est un interpréteur [...]
$ cd répertoire
$ ls
document.txt
$ cat document.txt
INF1070 - Utilisation et [...]
$ exit
```

Afficher un fichier texte

Commande `cat` (*concatenate*)

- `cat toto.txt` — Affiche le contenu de `toto.txt`
- `cat toto.txt tata.txt` — Affiche le contenu de `toto.txt` suivi de celui de `tata.txt`

```
$ cat fichier1
$ cat fichier1 fichier2
$ cat fichier2 fichier1
$ cat --number fichier1
$ cat -n fichier1
```

Autres commandes d'affichage (+ ou - utiles)

- `head` — afficher les première lignes
- `tail` — afficher les dernières lignes
- `less` (et `more`) — afficher un fichier page par page
- `nl` — numéroter les lignes
- `tac` — afficher un fichier en commençant par la dernière ligne
- `rev` — inverser chacune des lignes affichées
- `wc` — compter le nombre de lignes, mots et octets

```
$ wc fichier1 fichier2
$ wc --lines fichier1 fichier2
$ wc -l fichier1 fichier2
$ wc --words fichier1 fichier2
$ wc -w fichier1 fichier2

$ head --lines 2 fichier1 fichier2
$ head --lines=2 fichier1 fichier2
$ head -n 2 fichier1 fichier2
$ head -n2 fichier1 fichier2
```

Manuel en ligne

Comment trouver de l'information?

Approche typique des utilisateurs

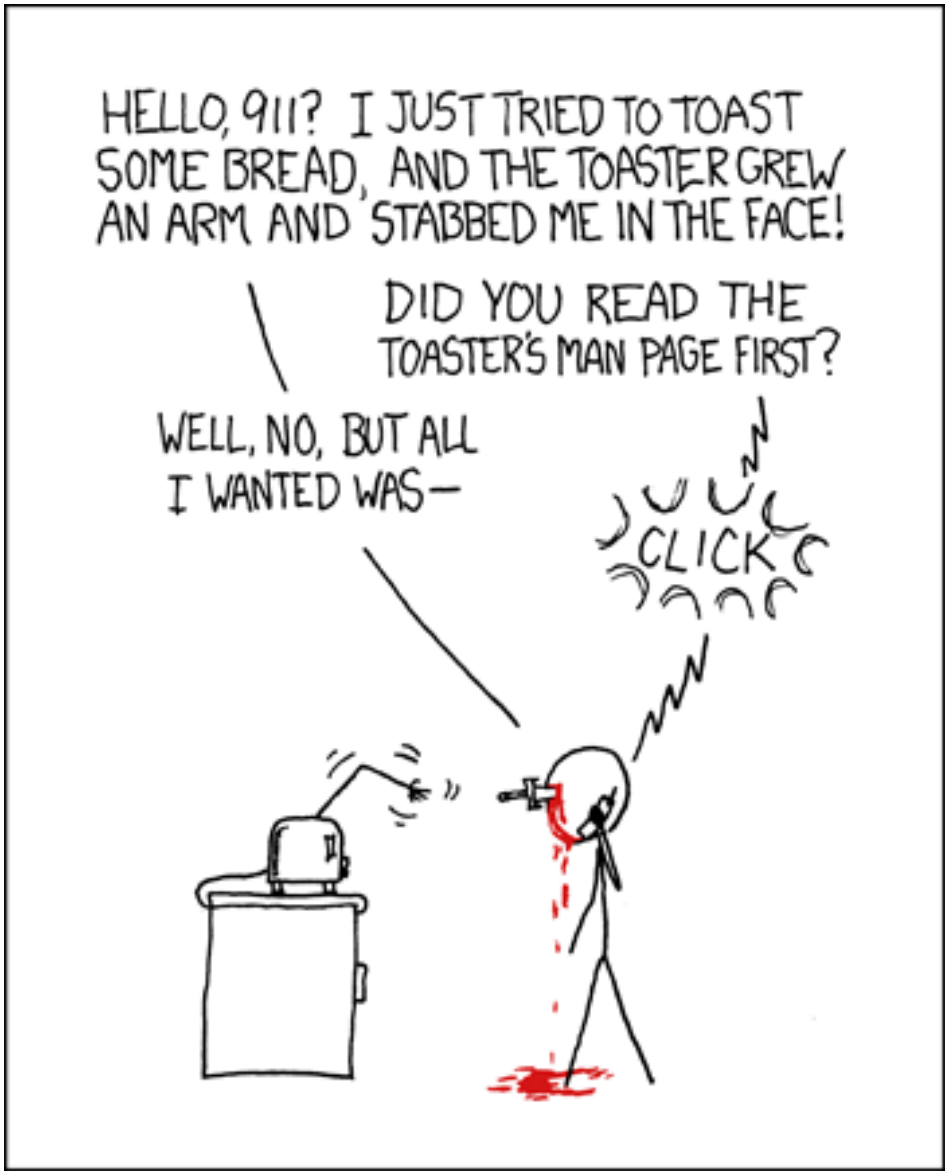
- Google
- Wikipedia
- StackOverflow
- Enseignant
- Démonstrateur
- L'association étudiante (AGEEI)

Approche unixienne efficace

En INF1070 (et cours suivants): le `man`

```
$ man ls
$ man date
$ man intro
$ man man
```

RTFM



Source: https://xkcd.com/293/

Naviguer dans le man

`man` utilise le paginateur (*pager*) `less` pour naviguer dans le document

- q quitte
- Espace ou Page↓ défile d'une page
- ⏎ ou ↓ défile d'une ligne
- g ou Home va au début
- G ou End va à la fin
- un nombre et ⏎ avance du nombre de lignes indiquées
- `/motif` cherche l'occurrence suivante de `motif`
- n cherche l'occurrence suivante
- N cherche l'occurrence précédente
- h pour un résumé des commandes et des raccourcis clavier

`less` s'utilise aussi de façon autonome

```
$ less /usr/share/common-licenses/GPL-3
```

Sections de manuel

1. Programmes de l'utilisateur
2. Appels système
3. Fonction de bibliothèque
4. Fichiers spéciaux
5. Formats de fichiers et conventions
6. Jeux
7. Divers
8. Programmes de l'administrateur

Pour spécifier la section (en cas d'ambiguïté)

- `man 2 mkdir`
- `man -s 2 mkdir`
- `man mkdir.2`

Contenu des pages de man

Les pages de man suivent souvent le même schéma

- **Titre** avec le numéro de la section
- **Nom** nom de la commande
- **Synopsis** syntaxe générale
  - « `[]` » désigne une information optionnelle
  - « `|` » désigne une alternative (ou)
  - « `...` » désigne une information répétable
- **Description** explications détaillées
- **Options** liste et description des options
- **Voir aussi** manuels connexes

Autre sources d'information

- `info` pour la doc complète des outils GNU (par exemple, `info info`)
- `yelp` pour l'aide de GNOME
- `apropos` pour chercher dans le `man` (`apropos uptime`)
- `help` pour les primitives du shell (`help cd`)
- option `--help` des commandes (`ls --help`)

Sur internet

- https://explainshell.com
  explications interactives des lignes de commandes
- https://tldr.inbrowser.app/pages/linux
  manuels courts basés sur des exemples
- https://manpages.debian.org/
  les manpages de Debian
- https://pubs.opengroup.org/onlinepubs/9799919799/
  La spécification POSIX (IEEE-1003.1-2024)

Éditeur de texte

Éditeur de texte

- Logiciel facilitant la manipulation de fichiers textes
- Texte brut (sans mise en forme) $\neq$ traitement de texte
- Police à chasse fixe pour l'alignement vertical (indentation)
- Utilisé pour la programmation (code source)
- Utilisé pour l'administration système (fichiers de configuration)

L'offre d'éditeurs de texte est très variée

- Notepad/Notepad++ (Windows)
- TextEdit (MacOS)
- gedit (multiplateforme)
- SublimeText (multiplateforme)
- Visual Studio Code (multiplateforme)
- Emacs et ses dérivés (multiplateforme)
- Vi/Vim et ses dérivés (multiplateforme)
- nano (Unix)
- ed le premier éditeur texte Unix ($\approx$ 1969), toujours disponible

Vi/Vim

Éditeur utilisé en classe: `vim`

- Vous pouvez utiliser `vim` ou `nano`

Caractéristiques de vi/vim

- Un des plus anciens éditeurs de texte
- Un des éditeurs de texte les plus utilisés dans le monde
- Ancêtre: Vi, créé par Bill Joy en 1976
- Vim = Vi iMproved
- Multiplateforme (Linux, MacOS, Windows)
- Standard sous UNIX ($\approx$ déjà installé)

Avantages/inconvénients de vim

Avantages

- Très mature
- Interaction directe avec le terminal
- Rapide et léger, en particulier pour le travail à distance
- Hautement configurable

Inconvénients

- Orienté seulement clavier (certains dérivés, comme GVim, permettent une utilisation limitée de la souris)
- Courbe d'apprentissage difficile quand on débute

Commandes et arguments

Principe des commandes UNIX (*Unix philosophy*)

Quelques règles de conception

- Chaque programme fait une chose et le fait bien
- Afficher seulement ce qui est demandé (Ne pas polluer les résultats)
- Lire et afficher du texte (car c'est universel)
- Éviter les commandes interactives
  Préférer les options, les arguments et l'entrée standard

Objectifs

- Les commandes sont claires et simples
- Les commandes sont destinées à l'humain
- Les commandes sont scriptables et combinables

Principe des commandes UNIX (*Unix philosophy*)

> *« Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools. »*
> *— Brian Kernighan et Rob Pike, The UNIX Programming Environment (1984)*

On y reviendra...

Standards des utilitaires

D'un **système** à l'autre et d'une **version** à l'autre,

Les commandes de base et leurs options **varient**

C'est pourquoi des **standards** existent

- POSIX (IEEE-1003.1): outils et comportements standard Unix
  Maintenu par l'IEEE (et l'Open Group)
- LSB (*Linux standard base*): suppléments pour distributions
  Maintenu par la fondation Linux
- GNU - extensions des utilitaires GNU
  Inclut les options longues des commandes POSIX
- Extra: le reste
  Habituels, souvent installés par défaut

Sauf mention contraire (ou oublis) les commandes et options vues en cours sont conformes POSIX.1-2017

Type des commandes

```
$ bonjour
bash: bonjour : commande introuvable
```

Une commande simple peut être, entre autres,

- Un fichier **exécutable**
  Un programme autonome, souvent dans `/bin` ou `/usr/bin`
- Une **commande interne** du shell (primitive, *built-in*)
  Habituellement un programme lié à l'état interne du shell, par exemple, `cd` ou `pwd`

On y reviendra au chapitre 7.

Manuel des commandes internes

- `man bash`
- `man bash-builtins`
- commande `help` (une commande interne de bash)

Arguments des commandes

Chaque commande traite ses arguments

- De sa façon **spécifique**
- Lisez le manuel

Il y a quand même des conventions

- Des options: qui commencent par « - »
- Le reste des arguments: qui ne commencent pas par « - »

Attention

Chaque commande peut avoir une gestion spécifique des arguments

Options des commandes

- Activent certains comportements spécifiques
- Configurent certains paramètres
- Sont combinables

Options courtes

- Commencent par « `-` » (tiret)
- Une lettre (ex. « `cat -n` »)
- S'agglutinent (ex. « `cat -nE` » vaut « `cat -n -E` »)
- Note: `-n` et `-E` sont des extensions GNU

Options longues (style GNU)

- Commencent par « `--` » (deux tirets)
- Ont parfois un synonyme court
- Exemple: « `cat --number --show-ends` »

Conventions

Certaines options sont comprises par de nombreuses commandes

- `--help` — affiche l'aide
- `--verbose` — mode verbeux
- `--version` — affiche la version de la commande

Rappel: chaque commande a ses propres règles

Conventions

Certaines options sont comprises par de nombreuses commandes

- `--help` — affiche l'aide
- `--verbose` — mode verbeux
- `--version` — affiche la version de la commande

Rappel: chaque commande a ses propres règles

Questions

- Que fait l'option « `-h` » de la commande `ls` (GNU) ?
- Comment afficher la version de la commande `java` ?
- Qu'affiche « `echo --help` » ?

Options avec valeur

Plusieurs syntaxes. Ça dépend de l'outil.

- `head -n 5 hello.txt`
- `head -n5 hello.txt`
- `head --lines 5 hello.txt`
- `head --lines=5 hello.txt`

Mais tout ne fonctionne pas

- `head -n=5 hello.txt`
- `head --lines5 hello.txt`
- `head --lines =5 hello.txt`
- `head --lines= 5 hello.txt`

Options avec valeur

Plusieurs syntaxes. Ça dépend de l'outil.

- `head -n 5 hello.txt`
- `head -n5 hello.txt`
- `head --lines 5 hello.txt`
- `head --lines=5 hello.txt`

Mais tout ne fonctionne pas

- `head -n=5 hello.txt`
- `head --lines5 hello.txt`
- `head --lines =5 hello.txt`
- `head --lines= 5 hello.txt`

Question

« `git commit --amend` » vs. « `git commit -amend` » ?

Options avec valeur

Plusieurs syntaxes. Ça dépend de l'outil.

- `head -n 5 hello.txt`
- `head -n5 hello.txt`
- `head --lines 5 hello.txt`
- `head --lines=5 hello.txt`

Mais tout ne fonctionne pas

- `head -n=5 hello.txt`
- `head --lines5 hello.txt`
- `head --lines =5 hello.txt`
- `head --lines= 5 hello.txt`

Question

« `git commit --amend` » vs. « `git commit -amend` » ?

- « `--amend` » est une option longue
- « `-amend` » = « `-a -mend` » = « `--all --message=end` »

Complètement (*completion*)

Le *shell* `bash` peut aider à écrire les commandes

Tabulation simple ⟶ (ou tab)

- complète l'argument ou l'option (si possible)
- `cat /etc/pas` ⟶
→ complète: `cat /etc/passwd`

Tabulation double ⟶ ⟶ (ou tab-tab)

- affiche une liste d'options ou d'arguments possibles
- `ls --re` ⟶ ⟶
→ propose: `--recursive --reverse`

Bash adapte le complètement aux commandes qu'il connaît

Certains shells sont plus avancés (`zsh`, `powershell`)

Redirection et tube

Redirection en sortie

Le résultat des commandes va sur la **sortie standard**

- Par défaut, la sortie standard est l'écran terminal
- Mais le **shell** peut la rediriger vers un fichier

```
$ ls -l hello.txt > ls.out
$ cat ls.out
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
```

Redirection en sortie

Opérateurs shell de redirections: « > » et « >> »

- « `> f` » écrit dans le fichier `f` depuis le début (écrase, *overwrite*)
- « `>> f` » écrit à la suite du fichier `f` (ajoute, *append*)
- Dans les deux cas, si `f` n'existe pas, il est créé

```
$ echo uno > tata
$ echo dos >> tata
$ cat tata
uno
dos
```

Sortie d'erreur standard

Les commandes distinguent deux sorties

- la **sortie standard** pour les résultats normaux
- la **sortie d'erreur standard** pour les messages d'erreurs et de diagnostics

```
$ ls -l hello.txt epic.fail > ls.out
ls: impossible d'accéder à 'epic.fail':
Aucun fichier ou dossier de ce type
$ cat ls.out
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
```

Redirection d'erreur standard

« `2>` » (et « `2>>` ») redirigent la **sortie d'erreur standard** vers un fichier

```
$ ls -l hello.txt epic.fail > ls.out 2> ls.err
$ cat ls.out
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
$ cat ls.err
ls: impossible d'accéder à 'epic.fail':
Aucun fichier ou dossier de ce type
```

/dev/null

- Un fichier spécial qui accepte (et ignore) des données
- On l'utilise pour ignorer des sorties de commande
- Souvent utilisé pour *taire* la sortie d'erreur standard

```
$ ls -l hello.txt epic.fail 2> /dev/null
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
```

Tubes

Le « | » (tube, *pipe*) connecte des commandes

- La **sortie standard** de l'une est connectée à
- l'**entrée standard** de la suivante

```
$ echo bonjour le monde | wc
1 3 17
$ echo bonjour le monde | rev | lolcat
ednom el ruojnob
```

Une **conduite** (*pipeline*) est une suite de commandes connectées par des tubes

Tubes et filtres

De nombreuses commandes traitent, filtrent ou transforment l'entrée standard vers la sortie standard

- `head`, `tail`, `less`, `tac`, `rev`, `wc`
- `sort` – trier les lignes
- `uniq` – éliminer les lignes répétées
- `tr` – convertir ou éliminer des caractères
- `grep` – afficher les lignes correspondant à un motif
- `cowsay` – vache qui parle (extra)
- `lolcat` – colorer en arc-en-ciel (extra)

Fichiers en arguments et entrée standard

La plupart des commandes traitent les fichiers et l'entrée standard

Convention habituelle des commandes

- Si plusieurs fichiers
  → Ils sont traités dans l'ordre
- Si un fichier n'existe pas
  → La commande affiche un message d'erreur
- Si pas de fichier
  → La commande lit l'*entrée standard*
- Si un argument est « `-` » (tiret seul)
  → Ça désigne aussi l'entrée standard

Convention $\neq$ standard

- Chaque commande a son propre comportement
- Lisez le manuel (`cat` par exemple)

Questions de tube

Construire une conduite qui

- Affiche le fichier `hello.txt`
- Trie les lignes
- Sélectionne les lignes qui contiennent « `il` »
- Transforme a, e, i et o en 4, 3, 1 et 0
- Fait parler la vache

Que font les commandes suivantes ?

```
$ rev hello.txt | tac
$ rev hello.txt | tac -
$ rev hello.txt | tac lisez-moi.txt -
$ rev hello.txt | tac lisez-moi.txt
```

L'entrée standard au clavier

L'**entrée standard** est l'entrée naturelle des commandes

- Par défaut, l'entrée standard est le clavier du terminal
- Les touches ctrl + D terminent l'entrée clavier
  Prononcer « contrôle-dé », écrire « `^D` »

```
$ tac
bonjour
le monde
^D
le monde
bonjour
```

L'entrée standard au clavier

L'**entrée standard** est l'entrée naturelle des commandes

- Par défaut, l'entrée standard est le clavier du terminal
- Les touches ctrl + D terminent l'entrée clavier
  Prononcer « contrôle-dé », écrire « `^D` »

```
$ tac
bonjour
le monde
^D
le monde
bonjour
```

Questions

- Quelle est la différence entre `^D` et la touche *enter* (↵) ?
- Quelle est la différence entre `^D` et `^C` ?
- Quelle est la différence si on remplace `tac` par `rev` ?

Commandes interactives

- Les commandes **interactives** dialoguent avec l'utilisateur
- Il saisit des instructions ou répond aux questions via le terminal

`sh` — démarrer un interpréteur de commande (*shell*)

```
$ sh
$ echo hello
hello
$ exit
```

`python` — lancer l'interpréteur du langage de programmation Python

```
$ python3
>>> print(1+1)
2
>>> quit()
```

Redirection en entrée

« < » redirige l'**entrée standard** depuis un fichier

```
$ cat < hello.txt
Bonjour [...]
```

Nécessaire pour lire un fichier avec certaines commandes

Redirection en entrée

« < » redirige l'**entrée standard** depuis un fichier

```
$ cat < hello.txt
Bonjour [...]
```

Nécessaire pour lire un fichier avec certaines commandes

Questions

Quelle est la différence entre ?

- `tr B b hello.txt`
- `tr B b < hello.txt`

Même question avec :

- `head -v hello.txt`
- `head -v < hello.txt`

Même question en supposant qu'`epic.fail` n'existe pas :

- `head -v epic.fail`
- `head -v < epic.fail`

Questions de redirection et tubes

Quelle est la différence entre les commandes suivantes ?

```
$ ls | wc
$ ls > wc
$ ls < wc
```

Questions de redirection et tubes

Quelle est la différence entre les commandes suivantes ?

```
$ ls | wc
$ ls > wc
$ ls < wc

$ ls | wc -l
$ ls > wc -l
$ ls < wc -l
```

Questions de redirection et tubes

Quelle est la différence entre les commandes suivantes ?

```
$ ls | wc
$ ls > wc
$ ls < wc

$ ls | wc -l
$ ls > wc -l
$ ls < wc -l
```

Quelle est la différence entre

```
$ ls > fichier
$ wc -l < fichier
```

et

```
$ ls | wc -l
```

Entrée standard = concept complexe

Plusieurs entités entrent en jeu

- L'**entrée standard** est un concept abstrait du **système d'exploitation**
- Le **terminal** s'occupe du clavier
  Lit du texte et gère `^D`, `^C`, *enter*...
- La **commande** se contente de lire l'*entrée standard*
  Sans se soucier de son origine

Les entrées-sorties et les terminaux sont plus complexes qu'il n'y parait

- On entrera plus dans les détails une autre fois
- Et surtout en INF3135 et INF3173

Conclusion des redirections et des tubes

Principe des commandes UNIX (*Unix philosophy*)

« Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools. »
— Brian Kernighan et Rob Pike, *The UNIX Programming Environment* (1984)

Conclusion des redirections et des tubes

Principe des commandes UNIX (*Unix philosophy*)

« Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools. »
— Brian Kernighan et Rob Pike, *The UNIX Programming Environment* (1984)

{~}

On ira plus loin sur les redirections et tubes une autre fois

Développement et caractères spéciaux du shell

Développement des noms de fichiers (glob)

Désigner simplement un ensemble de fichiers selon un motif

- Point d'interrogation `?` — un caractère quelconque
- Étoile `*` — zéro, un ou plusieurs caractères
- Crochets `[]` — un seul des caractères de la liste

Exemples

- « `cat *.txt` » se terminent par `.txt`
- « `cat ?[oa]*` » la deuxième lettre est `a` ou `o`

Note

- Le glob est géré par le shell (pas par la commande)
- La commande *ne voit* que les arguments une fois développés

Les détails: le manuel de `glob(7)`

Commande *echo*

`echo` — Affiche chacun des arguments séparés par un espace

```
$ echo a b                cd
a b cd
```

Pratique pour comprendre ce qui se passe

Commande *echo*

`echo` — Affiche chacun des arguments séparés par un espace

```
$ echo a b                cd
a b cd
```

Pratique pour comprendre ce qui se passe

Questions

Qu'affichent les commandes suivantes ?

- `echo --ya-quelqu-un`
- `echo *`
- `echo [*]`
- `echo ? ?`

Caractères spéciaux

Un **caractère spécial** est un caractère qui n'a pas un sens littéral

Exemples

- « `^D` » et « `^C` » sont spéciaux pour le terminal
- « `*` » et l'espace sont spéciaux pour le shell
- « `-` » en début d'argument est spécial pour la plupart des commandes

Difficultés

- Le sens d'un caractère dépend du contexte
- Chaque commande a ses règles
- Les règles sont + ou - compliquées
- Forcer l'interprétation littérale est + ou - complexe

Échapper avec la contre-oblique

- Échapper avec « \ » (contre-oblique, *backslash*)

→ Annule le caractère spécial qui suit

```
$ echo \*
*
```

Le « \ » s'utilise pour échapper des caractères dans d'autres contextes

Échapper avec la contre-oblique

- Échapper avec « \ » (contre-oblique, *backslash*)

→ Annule le caractère spécial qui suit

```
$ echo \*
*
```

Le « \ » s'utilise pour échapper des caractères dans d'autres contextes

Questions

Qu'affichent les commandes suivantes ?

```
$ echo a     \      b
$ echo a     \\     b
$ echo a     \\\    b
$ echo a \   \  \   b
```

Encadrer avec les guillemets simples

- « ' » (guillemet simple, *simple quote*)

→ Force l'interprétation littérale jusqu'au prochain « ' »

```
$ echo '*   \'
*   \
```

On peut utiliser « \ » sur le premier « ' » pour l'ignorer

```
$ echo \'
'
```

Encadrer avec les guillemets simples – Questions

Qu'affichent les commandes suivantes ?

```
$ echo 'abc\\def'
$ echo abc'\\'def
$ echo abc''def
$ echo abc '' def
$ echo 'abc'\\'def'
```

Encadrer avec les guillemets doubles

- « " » (guillemets doubles, *double quote*)

→ Une version amoindrie de « ' »

→ Permet 3 caractères spéciaux internes: « \ », « $ » et « ` » (on les verra plus tard)

```
$ echo "*    \"    \\"
*    "    \
```

Attention! Ne pas confondre avec les chaînes de caractères des langages de programmation

Encadrer avec les guillemets doubles – Questions

Qu'affichent les commandes suivantes ?

```
$ echo hello
$ echo "hello"
$ "echo" hello
$ "echo hello"
```

Commande *printf*

`printf` — Affiche chacun des arguments selon un format

```
$ printf "*«%s»\n" a   cd
*«a»
*«cd»
```

Dans le format, `%` et `\` indiquent des séquences spéciales

- `%s` Un argument sous forme de chaine (*string*)
- `\n` Un saut de ligne (*newline*)
- Consulter le man pour le reste.

Commande *printf*

`printf` — Affiche chacun des arguments selon un format

```
$ printf "*«%s»\n" a   cd
*«a»
*«cd»
```

Dans le format, `%` et `\` indiquent des séquences spéciales

- `%s` Un argument sous forme de chaine (*string*)
- `\n` Un saut de ligne (*newline*)
- Consulter le man pour le reste.

Questions

Qu'affichent

- `printf %s *`
- `printf %s\\n *`
- `printf * %s\n a b cd`

Caractères spéciaux du shell

Questions

- Les caractères « `\` », « `"` » ou « `'` » sont-ils **spéciaux** ?
- Qu'affiche « `echo "'\""\\'"'\'` » ?
- Comment afficher le contenu du fichier « `l'espace infini` » ?

Autres caractères spéciaux du shell

- Il y en a plein, on en verra quelques-uns au fur et à mesure
- Tout est documenté dans `bash`

Caractères spéciaux des commandes

- Chaque commande a ses propres règles
- Et ses propres options qui changent les règles

```
$ echo 'Une ligne\n\tbrisée'
Une ligne\n\tbrisée
$ echo -e 'Une ligne\n\tbrisée'
Une ligne
        brisée
```

Note: `echo -e` n'est pas POSIX, `printf` l'est.

Caractères spéciaux des commandes

- Chaque commande a ses propres règles
- Et ses propres options qui changent les règles

```
$ echo 'Une ligne\n\tbrisée'
Une ligne\n\tbrisée
$ echo -e 'Une ligne\n\tbrisée'
Une ligne
        brisée
```

Note: `echo -e` n'est pas POSIX, `printf` l'est.

Question

- Qu'affiche « `echo -e Une ligne\\n\\tbrisée` » ?

Caractères spéciaux des commandes

Par convention « -- » désigne la fin des options

```
$ cat -n world.txt
     1  Bonjour
     2  Monde
$ cat -- -n
Un bon numéro
$ cat -- -n world.txt
Un bon numéro
Bonjour
Monde
```

Caractères spéciaux des commandes

Par convention « -- » désigne la fin des options

```
$ cat -n world.txt
     1  Bonjour
     2  Monde
$ cat -- -n
Un bon numéro
$ cat -- -n world.txt
Un bon numéro
Bonjour
Monde
```

Questions

- Comment numéroter les lignes du fichier « -n » ?
- Comment afficher le contenu du fichier « -- » ?
- Qu'affiche « `echo -- -e "a\nb"` » ?