Programmation C et Système
Commandes usuelles en pipeline

Régis Barbanchon

L2 Informatique

Commandes classiques utilisées souvent en pipeline

- `echo/printf` : affiche ses arguments.
- `cat` : affiche des fichiers concaténés.
- `nl` : numérote les lignes
- `wc` : compte les caractères, les mots, les lignes.
- `head/tail` : affiche les premières/dernières lignes.
- `grep` : filtre les lignes contenant un motif régulier.
- `cut` : affiche des colonnes.
- `sort` : trie les lignes.
- `uniq` : filtres les lignes consécutives identiques.
- `paste` : fusionne les lignes de plusieurs fichiers.
- `join` : effectue la jointure de deux fichiers.
- `tr` : substitue des caractères par d'autres.
- `sed` : substitue un motif régulier par du texte.

Les commandes d'affichage `echo` et `printf`

`echo` affiche ses arguments séparés par une espace sur une ligne.

```
$ echo "bonjour"      "tout le monde"
bonjour tout le monde
```

```
$ echo "bonjour         tout le monde"
bonjour         tout le monde
```

L'option `-n` supprime le retour à la ligne final implicite.
L'option `-e` interprète les séquences d'échappement avec backslash.

```
$ echo "bonjour\ntout le monde\n"
bonjour\ntout le monde\n
```

```
$ echo -ne "bonjour\ntout le monde\n"
bonjour
tout le monde
```

`printf` est la commande analogue à la fonction éponyme en C.

```
$ name="Bob"
$ age="15"
$ printf "%s is %d old.\n"   "${name}"   "${age}"
Bob is 15 year old.
```

La commande `cat` (*catenate*) et ses variantes `tac` et `rev`

`cat` concatène sur `stdout` le contenu de fichiers.
On s'en sert aussi pour afficher le contenu d'un seul fichier.

```
$ cat ./fruits
ananas
banana
```

```
$ cat ./animals
dog
cat
```

```
$ cat ./colors
red
blue
```

```
$ cat ./fruits ./animals ./colors
ananas
banana
dog
cat
red
blue
```

Pour chaque fichier, `tac` affiche ses lignes dans l'ordre inverse.
Pour chaque fichier, `rev` affiche ses lignes renversées.

```
$ tac  ./fruits ./animals ./colors
banana
ananas
cat
dog
blue
red
```

```
$ rev  ./fruits ./animals ./colors
sanana
ananab
god
tac
der
eulb
```

La commande `nl` (*number lines*)

`nl` affiche sur `stdout` des lignes numérotées de fichiers.
Par défaut, les lignes vides ne sont pas numérotées.
(Une ligne vide ne contient que le caractère `\n`.)

```
$ cat ./alcohols
beer

wine
vodka
```

```
$ cat ./hot-drinks
tea
coffee

chocolate
```

```
$ nl ./alcohols ./hot-drinks
     1  beer

     2  wine
     3  vodka
     4  tea
     5  coffee

     6  chocolate
```

```
$ nl -ba -v 100 -i 10   ./alcohols ./hot-drinks
   100  beer
   110
   120  wine
   130  vodka
   140  tea
   150  coffee
   160
   170  chocolate
```

Le numérotation est paramétrable. Par exemple :

- `nl -ba` numérote aussi les lignes vides (*Body style : All lines*).
- `nl -v start` numérote à partir de `start`.
- `nl -i incr` numérote par incréments de `incr`.

La commande `wc` (*word count*)

`wc` affiche le compte des bytes, mots, et lignes des fichiers.

- `wc -c` compte les bytes.
- `wc -w` compte les mots.
- `wc -l` compte les retours à la ligne.

```
$ cat ./sodas
coca cola
pepsi cola
```

```
$ cat ./colors
red
blue
```

```
$ wc -c ./sodas ./colors
21 sodas
 9 colors
30 total
```

```
$ wc -w ./sodas ./colors
 4 sodas
 2 colors
 6 total
```

```
$ wc -l ./sodas ./colors
 2 sodas
 2 colors
 4 total
```

S'il n'y a qu'un fichier en entrée, le total n'est pas affiché :

```
$ wc -c ./sodas
21 sodas
```

```
$ wc -w ./sodas
 4 sodas
```

```
$ wc -l ./sodas
 2 sodas
```

Si l'entrée est une redirection de `stdin`, seul le compte est affiché :

```
$ cat ./sodas   | wc -w
4
```

```
$ wc -w    < ./sodas
4
```

Les commandes `head` et `tail` (*print head and tail lines*)

`head` affiche sur `stdout` les 10 premières lignes seulement.

- `head -n num` affiche les `num` premières lignes.
- `head -n -num` affiche toutes les lignes sauf les `num` dernières.

```
$ cat ./games
chess
go
shogi
checkers
hold'em
omaha
belote
```

```
$ cat ./games | head -n 2
chess
go
```

```
$ cat ./games | head -n -5
chess
go
```

`tail` affiche sur `stdout` les 10 dernières lignes seulement.

- `tail -n num` affiche les `num` dernières lignes.
- `tail -n +num` affiche toutes les lignes à partir de la `num`$^{\text{ème}}$.

```
$ cat ./games
chess
go
shogi
checkers
hold'em
omaha
belote
```

```
$ cat ./games | tail -n 2
omaha
belote
```

```
$ cat ./games | tail -n +6
omaha
belote
```

La commande `sort` (*sort lines*)

`sort` affiche les lignes triées (lexicographiquement par défaut).

Les principales options sont :

- `sort -r` : renverse l'ordre des lignes.
- `sort -t tab` : utilise le car. `tab` pour séparer les colonnes.
- `sort -k num1,num2` : trie sur les colonnes `num1` à `num2`.
- `sort -n` : utilise un tri numérique plutôt que lexicographique.

```
$ cat ./people
charlie#8#male
alice#23#female
bob#42#male
daniella#10#female
```

```
$ cat ./people | sort
alice#23#female
bob#42#male
charlie#8#male
daniella#10#female
```

```
$ cat ./people | sort -r
daniella#10#female
charlie#8#male
bob#42#male
alice#23#female
```

```
$ cat ./people | sort -t '#' -k 2,2
daniella#10#female
alice#23#female
bob#42#male
charlie#8#male
```

```
$ cat ./people | sort -n -t '#' -k 2,2
charlie#8#male
daniella#10#female
alice#23#female
bob#42#male
```

La commande `uniq` (*print unique consecutive lines*)

`uniq` détecte les groupes de lignes consécutives identiques, et n'affiche qu'une seule ligne par groupe. Options :

- `uniq -c` : affiche devant chaque groupe sa taille.
- `uniq -u` : n'affiche que les groupes de taille $= 1$.
- `uniq -d` : n'affiche que les groupes de taille > 1.

```
$ cat ./numbers
one
two
two
three
three
three
two
one
```

```
$ uniq ./numbers
one
two
three
two
one
```

```
$ uniq -c ./numbers
      1 one
      2 two
      3 three
      1 two
      1 one
```

```
$ uniq -u numbers
one
two
one
```

```
$ uniq -c -u ./numbers
      1 one
      1 two
      1 one
```

```
$ uniq -d ./numbers
two
three
```

```
$ uniq -c -d ./numbers
      2 two
      3 three
```

`uniq` est souvent utilisé en pipeline avec `sort` :

```
$ cat ./numbers | sort | uniq -c
      2 one
      3 three
      3 two
```

La commande `fgrep` (ou `grep -F`)

`fgrep string` affiche les lignes contenant une chaîne `string`.

```
$ cat ./schedule
monday#english#math
tuesday#math#prog
thursday#math#sport
wednesday#prog#english
friday#prog#sport
```

```
$ cat ./schedule | fgrep math
monday#english#math
tuesday#math#prog
thursday#math#sport
```

Les options usuelles sont :

- `-v` : matche l'inverse, c-à-d, affiche les lignes sans `string`.
- `-c` : affiche le nombre de matchs, au lieu d'afficher ceux-ci.
- `-i` : matche en étant insensible à la casse.

```
$ cat ./schedule | fgrep -v math
wednesday#prog#english
friday#prog#sport
```

```
$ cat ./schedule | fgrep -c math
3
$ cat ./schedule | fgrep -v -c math
2
```

On peut rechercher plusieurs chaînes, en les séparant par des `\n`.

```
$ LIST=$(echo sport ; echo math )
$ echo "$LIST"
sport
math
```

```
$ cat ./schedule | fgrep "$LIST"
monday#english#math
tuesday#math#prog
thursday#math#sport
friday#prog#sport
```

Une ligne qui matche plusieurs fois n'est affichée qu'une fois.

La commande `grep` (ou `grep -G`)

`grep regex` affiche les lignes matchant le motif `regex`,
où `regex` est une expression régulière **basique**.
L'expression est construite en concaténant les syntaxes suivantes :

- `r?` : `r`, éventuellement absent.
- `r*` : une suite de `r`, éventuellement vide.
- `^r` : `r` en début de ligne.
- `r$` : `r` en fin de ligne.
- `[A-Za-z]` : un caractère dans la classe (ici, des lettres).
- `[^A-Za-z]` : un caractère hors de la classe (ici, des lettres).
- `.` : joker, un caractère quelconque (différent de `\n`).

```
$ cat ./mots
1#chaise#nom#feminin#singulier
2#elles#pronom#feminin#pluriel
3#feminin#adjectif#masculin#singulier
4#pluriel#adjectif#masculin#singulier
```

```
$ cat ./mots | grep '^[^#]*#[^#]*#[^#]*#feminin#'
1#chaise#nom#feminin#singulier
2#elles#pronom#feminin#pluriel
```

```
$ cat ./mots | grep 'pluriel$'
2#elles#pronom#feminin#pluriel
```

La commande egrep (ou grep -E)

`egrep regex` affiche les lignes matchant le motif `regex`,
où `regex` est une expression régulière **étendue**.
Les syntaxes basiques sont acceptées ainsi que les suivantes :

- `(r)` : le motif `r` lui-même.
- `r|s` : soit `r`, soit `s`.
- `r+` : une suite de `r`, non vide.
- `r{n}` : une suite de `r`, de longueur exactement `n`.
- `r{n,m}` : une suite de `r`, de longueur comprise entre `n` et `m`.
- `r{n,}` : `r` une suite de `r`, de longueur au moins `n`.
- `r{,m}` : une suite de `r`, de longueur au plus `m`.

```
$ cat ./mots
1#chaise#nom#feminin#singulier
2#elles#pronom#feminin#pluriel
3#feminin#adjectif#masculin#singulier
4#pluriel#adjectif#masculin#singulier
```

```
$ cat ./mots | egrep '^([^#]*#){2}(nom|pronom)#'
1#chaise#nom#feminin#singulier
2#elles#pronom#feminin#pluriel
```

La commande `cut` (*cut line fields*)

`cut` découpe les lignes en champs délimités par un caractère.

Les principales options sont :

`cut -d delim` spécifie le caractère délimiteur de champ.

`cut -f num,...` sélectionne un numéro de champ.

`cut -f num1-num2,...` selectionne une plage de champs.

```
$ cat ./people
charlie#8#male
alice#23#female
bob#42#male
daniella#10#female
```

```
$ cat ./people | cut -d '#' -f 1,3
charlie#male
alice#female
bob#male
daniella#female
```

```
$ cat ./people | cut -d '#' -f 2
8
23
42
10
```

On ne peut pas permuter l'ordre des champs avec `cut` :

```
$ cat ./people | cut -d '#' -f 3,1   # yields same result as -f 1,3
charlie#male
alice#female
bob#male
daniella#female
```

La commande `paste` (*paste lines together*)

`paste` colle les lignes correspondantes de plusieurs fichiers.
On spécifie le caractère délimiteur de ligne avec l'option `-d delim`

```
$ cat ./english
cat
dog
bird
```

```
$ cat ./french
chat
chien
oiseau
```

```
$ paste -d '#' ./english ./french
cat#chat
dog#chien
bird#oiseau
```

On peut voir `paste` comme la réciproque de `cut` :

```
$ cat ./people
charlie#8#male
alice#23#female
bob#42#male
daniella#10#female
```

```
$ cut -d# -f1 ./people | tee ./names
charlie
alice
bob
daniella
```

```
$ cut -d# -f3 ./people | tee ./genders
male
female
male
female
```

```
$ paste -d '#' ./genders ./names
male#charlie
female#alice
male#bob
female#daniella
```

On obtient ainsi la permutation tentée au slide précédent.

La commande `join` (*join lines of 2 files on a common field*)

`join -t tab -j num file1 file2`

joint les lignes de `file1` et `file2` qui ont le même champ n° `num`.
Les champs doivent être séparés par le caractère `tab`.
Les fichiers doivent être triés sur le champ de jointure.

```
$ cat boys
bob#35#tennis
fred#43#tennis
hank#20#rugby
joe#60#golf
```

```
$ cat girls
anna#70#golf
cathy#19#tennis
emma#25#golf
julia#38#tennis
```

```
$ sort -t# -k3,3 boys
joe#60#golf
hank#20#rugby
bob#35#tennis
fred#43#tennis
```

```
$ sort -t# -k3,3 girls
anna#70#golf
emma#25#golf
cathy#19#tennis
julia#38#tennis
```

On joint les garçons et les filles qui ont un sport commun, via :

```
$ sort -t# -k3,3 boys  > boys.sorted-by-sport
$ sort -t# -k3,3 girls > girls.sorted-by-sport
$ join -t# -j3 boys.sorted-by-sport girls.sorted-by-sport
golf#joe#60#anna#70
golf#joe#60#emma#25
tennis#bob#35#cathy#19
tennis#bob#35#julia#38
tennis#fred#43#cathy#19
tennis#fred#43#julia#38
```

Ou encore, en utilisant la substitution de processus :

```
$ join -t# -j3    <( sort -t# -k 3,3 boys)   <( sort -t# -k 3,3 girls )
```

La commande `sed s/regex/string/flag`

Dans toute les les lignes, `sed s/regex/string/flag` remplace la chaîne matchant `regex` par la chaîne `string`, où `regex` est une expression régulière **basique**.
Le drapeau `flag` peut être :

- `g` : tous les matchs de la ligne sont remplacés,
- un nombre `num` : le `num`$^{\text{ème}}$ match de la ligne est remplacé,
- *(vide)* : le premier match de la ligne est remplacé.

```
$ cat ./people
charlie#8#male
alice#23#female
bob#42#male
daniella#10#female
```

```
$ cat ./people | sed 's/#[^#]*male$/#human/'
charlie#8#human
alice#23#human
bob#42#human
daniella#10#human
```

```
$ cat ./people | sed 's/#/:/g'
charlie:8:male
alice:23:female
bob:42:male
daniella:10:female
```

```
$ cat ./people | sed 's/#/:/1'
charlie:8#male
alice:23#female
bob:42#male
daniella:10#female
```

```
$ cat ./people | sed 's/#/:/2'
charlie#8:male
alice#23:female
bob#42:male
daniella#10:female
```

La commande `tr` *(translate or delete characters)*

`tr chars1 chars2` remplace les cars `chars1` par les cars `chars2`.

`tr -d chars1` supprime les caractères `chars1`.

```
$ cat ./mots
1#chaise#nom#feminin#singulier
2#elles#pronom#feminin#pluriel
3#feminin#adjectif#masculin#singulier
4#pluriel#adjectif#masculin#singulier
```

```
$ cat ./mots | tr '#' ':'
1:chaise:nom:feminin:singulier
2:elles:pronom:feminin:pluriel
3:feminin:adjectif:masculin:singulier
4:pluriel:adjectif:masculin:singulier
```

```
cat ./mots | tr [:lower:] [:upper:]
1#CHAISE#NOM#FEMININ#SINGULIER
2#ELLES#PRONOM#FEMININ#PLURIEL
3#FEMININ#ADJECTIF#MASCULIN#SINGULIER
4#PLURIEL#ADJECTIF#MASCULIN#SINGULIER
```

```
$ cat ./mots | tr '#' '\n' | egrep '^(feminin|masculin)$'
feminin
feminin
feminin
masculin
masculin
```

```
$ cat ./mots | tr '#' '\n' | egrep '^(feminin|masculin)$' | wc -l
5
```