INTRODUCTION (TRÈS) RAPIDE À LA PROGRAMMATION CSH

IFT6285

Philippe Langlais

felipe@iro.umontreal.ca

RALI
Dept. Informatique et Recherche Opérationnelle
Université de Montréal

À PROPOS ...

- Ces transparents ne sont en aucun cas une apologie de la programmation shell :
  - Python (or any of the languages you like) is super, vive Python (or any of the languages you like) !
- Résoudre des problèmes complexes de TALN avec seulement de la programmation shell relève de la déviance.
- La syntaxe utilisée dans ces transparents est celle de **csh** qui est connue pour avoir des problèmes. Préférez lui **bash**.
- **Message :**
  - La connaissance de quelques commandes évite souvent de programmer (no need for an editor) et aide à mieux décomposer un logiciel en composants.

rali

SHELL SCRIPT

LIGNE DE COMMANDE SHELL

- dans un **terminal** :

`ls`	liste le contenu du répertoire courant
`cd rep`	se déplace dans le répertoire `rep`
`mkdir rep`	crée le répertoire `rep` dans le répertoire courant
`cd`	se déplace dans le répertoire **home**
`pwd`	indique le répertoire courant

- beaucoup de commandes, certaines complexes :

```
1  find . -name "*.cpp" -exec grep -i 'funcZ(' {} \;
```

rali

```
1 echo "bonjour" | rev
```

`echo` affiche une chaine

`rev` renverse le texte qu'on lui donne, ligne à ligne

`c1 | c2` l'entrée de `c2` est la sortie de `c1`

```
1 echo "bonjour" | rev >! a
```

`c1 >! file` la sortie de `c1` est **redirigée** dans le fichier `file` (qui est écrasé s'il existe)

`c1 >> file` la sortie de `c1` est **redirigée** dans le fichier `file` (après le contenu existant)

rali

% man tr

```
TR(1)                                                                User Commands

NAME
       tr - translate or delete characters

SYNOPSIS
       tr [OPTION]... SET1 [SET2]

DESCRIPTION
       Translate, squeeze, and/or delete characters from standard input, writing to standard output.

       -c, -C, --complement
              use the complement of SET1

       -d, --delete
              delete characters in SET1, do not translate

       -s, --squeeze-repeats
              replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character

       -t, --truncate-set1
              first truncate SET1 to length of SET2

       --help display this help and exit

       --version
              output version information and exit

       SETs are specified as strings of characters.  Most represent themselves.  Interpreted sequences are:

       \NNN   character with octal value NNN (1 to 3 octal digits)
```

rali

COMMANDE TR

```
% head -n 1 zola1.txt
```

Onze heures venaient de sonner à la Bourse, lorsque Saccard entra chez Champeaux, dans la salle blanc et or, dont les deux hautes fenêtres donnent sur la place.

```
% head -n 1 zola1.txt | tr ' ' '\012'
```

Onze
heures
venaient
de
sonner
à
la
Bourse,
lorsque ...

COMMANDE GREP

```
% head -n 60 zola1.txt | tr ' ' '\012' | grep -i ven
```

venaient
venu
venait
souvent
venu
vendre
venir

- `grep` recherche une chaine (ou expression régulière) dans l'**entrée standard**
- toutes les occurrences des chaines sont affichées sur la **sortie standard**

COMMANDE UNIQ

```
% head -n 200 zola1.txt | tr ' ' '\012' | grep -i
"^ven" | sort | uniq -c
```

```
1 venaient
5 venait
1 vend
1 vendez,
2 vendre
2 venir
1 vent,
1 vente
1 ventre
2 venu
1 venue
```

- `sort` tri un fichier (beaucoup d'options disponibles)
- `uniq` élimine les séquences de chaines identiques (l'option `-c` permet de compter les répétitions)

rali

COMMANDE SORT

```
% head -n 200 zola1.txt | tr ' ' '\012' | grep -i
"^ven" | sort | uniq -c | sort -k1,1nr
```

```
5 venait
2 vendre
2 venir
2 venu
1 venaient
1 vend
1 vendez,
1 vent,
1 vente
1 ventre
1 venue
```

- l'option `-k1,1n` donnée à `sort` demande le tri de la première colonne selon un critère numérique (`n`) et en ordre inverse (`r`) qui par défaut est croissant

rali

COMMANDE AWK

- permet d'introduire des programmes C en shell
- un tutoriel est nécessaire pour maîtriser cette commande
- on peut s'en servir pour afficher une colonne particulière (ici la seconde)

```
% head -n 200 zola1.txt | tr ' ' '\012' | grep -i
"^ven" | sort | uniq -c | sort -k1,1nr |
awk 'print $2'
```

```
venait
vendre
venir
venu
venaient
vend ...
```

rali

- afficher une ligne sous contrainte
  (ici la première colonne doit être supérieure à 2)

```
% head -n 1000 zola1.txt | tr ' ' '\012' | grep -
"^ven" | sort | uniq -c | sort -k1,1nr |
awk '$1 > 2 {print $0}'
```

```
25 venait
7 venir
6 venu
4 Vendôme,
3 venaient
3 venant
3 vendre
3 vente
3 ventre
```

TANNÉS DE TAPER DES COMMANDES À CHAQUE FOIS ?

- Créez un **script** !!!

```
% cat myscript
```

```
1  #!/bin/csh -f
2
3  set in = $1
4
5  cat $in | \
6        awk 'BEGIN {ok=0} \
7            /DEBUT DU FICHIER/ {ok = 1} \
8            /FIN DU FICHIER/ {ok=0} \
9            {if (ok>10) print $0; else if (ok) ok++}' \
10         | tr  ' ' '\012' | tr '[:upper:]' '[:lower:]'
```

- `chmod u+x myscript`
- `./myscript zola1.txt`

rali

USE CASE

BT.TAR.GZ ?

- **% tar -xzvf bt.tar.gz**

  crée une arborescence de racine `bt`

- **% ls bt/**

  ALIGNMENT-QUALITY/ MISALIGNED/ QUALITY/

- **% ls bt/ALIGNMENT-QUALITY/**

  000-SWINE FLU/ 001-AGRICULTURE AGRI-FOOD CAN/ ...

- **% ls bt/ALIGNMENT-QUALITY/000-SWINE FLU/**

  en-fr/

- **% ls bt/ALIGNMENT-QUALITY/000-SWINE FLU/en-fr**

  8926675/ 8326875/

rali

BT.TAR.GZ?

- `% ls bt/ALIGNMENT-QUALITY/000-SWINE FLU/en-fr/892`

```
8926675_code_FR.pptx
8926675_code_EN.docx
8926675-642140.tmx
flagging-details.txt
```

- `% cd bt/ALIGNMENT-QUALITY/000-SWINE FLU/en-fr/892`

- `% cat flagging-details.txt`

```
document name flag type flag date segment number
8926675_code_FR.pptx ALIGNMENT-QUALITY 2015-11-14 12 :13 :37
7
```

rali

BT.TAR.GZ ?

(on est remonté à la racine `bt`)

- Combien de fichiers `.txt` ?

```
% find . -name "*.txt" | wc -l
```

```
7695
```

- chaque fichier `.txt` contient une entête et une ligne par annotation. Combien d'annotations au total ?

```
% find . -name "*.txt" -exec wc -l {}   |
awk '{print i += $1}' | tail -n 1
```

```
18952
```

réponse :18952 - 7695

BT.TAR.GZ ?

- Chaque répertoire ”feuille” abrite un document source, un document cible (chacun dans des formats variés), ainsi qu'un (ou plusieurs) `.tmx` et un fichier `.txt`. Combien de formats différents pour les documents source et cible ?

```
% find . -type f | awk -F"." '{print $NF}' |
sort | uniq -c | sort -k1,1nr | head
```

```
8159 tmx
7695 txt
6675 doc
4955 docx
1847 ppt
718 pptx
258 xls
217 lwp
213 xlsx
210 wpd
```

donc majoritairement des documents Word et Excel

rali

SUR LA COMPOSITION DE PROGRAMMES

ENTRAÎNER/TESTER UN CLASSIFICATEUR (INPUT : TEXTE)

Quelques composantes de la solution :

- **préparation** du corpus : nettoyage, tokenization, lemmatisation/racinisation ou pas, découpage (ou pas) en sous mots, retrait (ou pas) des mots outils, cutoff, ...
- **apprenant :** choix d'un algorithme de classification, de ses métaparamètres (tuning)
- **évaluation** de performance

rali

ON PEUT BIEN SÛR TOUT EXPOSER À PLAT

```
% cat myscript

# repertoire ou se trouve mon corpus
rep_dir = dir_de_mon_gros_corpus

# gros code pour pretraiter le corpus
# le resultat de la preparation est ici:
prep_dir = dir_de_mon_gros_corpus_pret

# entrainement d'un modele
# le modele est ici
model_dir =  dir_de_mon_modele

# code pour appliquer le modele sur le test
# le resultat produit est ici:
mon_resultat = ici/mon_resultat

# mon code pour evaluer
```

rali

OU DÉCOMPOSER

% prepare.py

```
1  cat <corpus> | prepare.py --lemma --tokenize=space
```

% train.py

```
1  cat <corpus> | train.py --beta=0.1 \
2                 --lr=0.001 --modelname=./model/mon_modele
```

% apply.py

```
1  cat <test> | apply.py ./model/mon_modele >! result
```

% eval.py

```
1  cat <result> | eval.py --ref=<ref>
```

rali

OU DÉCOMPOSER

- plus facile d'étudier différentes variantes

ex.

```
foreach tok in ("space" "lm" "svm")
   foreach l in ("--lemma" "--racine" "")
       cat <corpus> | prepare.py $l --tokenize=$tok \
          >!  <rep>/corpus/corpus-${l}-$tok
   endfor
endfor
```

- chaque composant peut être écrit dans n'importe quel language (incluant un script)
- selon la nature des composants, on peut chaîner les commandes

ex.

```
cat <corpus> | prepare.py --lemma | train.py >! \
     <rep>/model/mon-model.lemma
```

rali