Formation continue

Unix avancé – exercices

2 Archivage et compression

Après tous ces exercices, il est temps de sauvegarder votre travail.

1. Placez vous dans votre répertoire personnel (*home directory*).
2. Créez une archive `unix.tar` avec la commande `tar`

   ```
   tar cvf unix.tar *
   ```
3. Constatez la création de l'archive.
4. Vérifiez le contenu de l'archive.
5. Quelle est la taille de l'archive (précisez l'unité) ?
6. Compressez l'archive avec la commande `gzip`. Quel est le nouveau nom de l'archive ?
7. Vérifiez à nouveau la taille de l'archive (précisez l'unité).

3 Flux, Filtres, *parsing* et expressions régulières

1. Python est un langage informatique interprété. On peut utiliser son interpréteur en mode interactif, c'est-à-dire qu'on peut entrer des commandes et tester leurs effets directement. Lancez Python (avec la commande `python`). Tapez les deux commandes interactivement `print 'Salut tout le monde'` puis `print 2 + 3`. Observez bien la sortie que Python génère.
2. Enregistrez les deux commandes (`print 'Salut tout le monde'` et `print 2 + 3`) chacune sur une ligne dans un fichier `toto`. Lancez Python en redirigeant l'entrée standard depuis le fichier `toto` avec la commande `python < toto`. Que se passe-t-il ?
3. Lancez Python avec cette série de commandes :

   ```
   python << INPUT
   print 'Salut les copains'
   print 2 + 3
   INPUT
   ```

   Que se passe-t-il ? À quoi sert `INPUT` ?
4. Créez un fichier avec *nedit* contenant une liste de courses (6 éléments, un élément par ligne). Affichez le fichier avec `cat`. Affichez le fichier trié par ordre alphabétique. Affichez les deux dernières lignes du fichier après l'avoir trié. Attention, pour cet exercice, il est interdit d'écrire dans un fichier.
5. Cherchez un fichier dont le nom est `nedit` à partir de la racine (`find / -name "nedit"`) en redirigeant la sortie dans le fichier `recherche.txt`. Que se passe-t-il ? Pourquoi ?
6. Refaites la même recherche en redirigeant les sorties standard et d'erreur dans le même fichier. Comment expliquez-vous la différence ? Regardez dans le fichier `recherche.txt`. Avez-vous finalement trouvé `nedit` ?

3.1 Aide pour jouer au pendu ou aux mots croisés avec Unix

Le fichier `/usr/share/dict/french` contient une liste de mots français.

1. Combien de mots contient le fichier `/usr/share/dict/french` ?
2. Quels mots commencent par la lettre z et se terminent par la lettre z ?
3. Quels mots contiennent 23 lettres ou plus ?
4. Quels mots commencent par `bio` et se terminent par `que` ?
5. Quel mot de 6 lettres commence par a, suivi d'une lettre, suivi de y, suivi de deux lettes et se termine par e ?

3.2 Extraction des carbones alpha d'un fichier PDB

1. Depuis la rubrique « Documents et liens » de la plateforme Didel, téléchargez le fichier `1MSE.pdb.gz`, copiez-le dans votre répertoire puis décompressez-le.
2. En une seule ligne de commande, affichez les lignes du fichier `1MSE.pdb` montrant les coordonnées tridimensionnelles des atomes. Filtrez la sortie de la commande précédente pour n'affichez que les carbones alpha. Toujours en une seule ligne de commande, combien y a-t-il de carbones alpha ?
3. À partir de la commande précédente et du programme `egrep`, comptez le nombre de résidus aromatiques (phénylalanine, tryptophane et tyrosine).
4. À partir du fichier `1MSE.pdb` et de la commande `awk`, affichez les colonnes contenant les coordonnées cartésiennes (x, y et z) des atomes.
5. Affichez en colonne la séquence de la protéine contenue dans `1MSE.pdb` avec les deux méthodes suivantes.

   Première méthode : utilisez deux `grep`, un `awk` et des *pipes* mais pas d'expressions régulières.

   Seconde méthode : utilisez un seul `awk` et une expression régulière.
6. **exercice +++** Avec la commande `awk`, calculez et affichez les coordonnées du centre géométrique de la protéine de `1MSE.pdb`. Pour cela, définissez des variables dans `awk` qui vont être incrémentées.

3.3 Extraction du nombre de gènes d'un fichier genbank

1. Depuis la rubrique « Documents et liens » de la plateforme Didel, téléchargez le fichier `NC_001133.gbk.gz`, copiez-le dans votre répertoire puis décompressez-le. Ouvrez ce fichier avec un éditeur de texte (*gedit* par exemple) et observez son format avec attention.
2. Avec `grep`, affichez le nom de l'organisme. Faites la même chose avec `awk`.
3. Exécutez la commande `grep gene NC_001133.gbk`. Pensez-vous que cette méthode soit satisfaisante pour extraire le nombre de gènes ?
4. Avec `egrep`, le métacaractère de début de ligne `^` et les métacaractères de comptage `{` et `}`, extrayez le nombre de gènes de ce fichier genbank. Pensez à bien vérifier la sortie de `egrep` avant de compter le nombre de lignes renvoyées.
5. En vous basant sur l'expression régulière précédente, extrayez le nombre de gènes sur le brin principal ainsi que sur le brin complémentaire.

3.4 Extraction de la séquence d'ADN d'un fichier genbank avec `grep` et `sed`

1. Pour démarrer avec le programme `sed`, reprenez l'exercice précédent qui extrait toutes les lignes contenant `gene` avec `egrep`. Avec un *pipe* et `sed`, transformez tous les mots `gene` de la sortie de `egrep` par `interesting gene`.
2. Avec `egrep`, extrayez toutes les lignes commençant par des d'espaces, suivis d'un nombre composé de 1 à 8 chiffres. En développant cette expression régulière, extrayez seulement les lignes contenant la séquence en bases. Vérifiez régulièrement vos résultats, par exemple en rédirigeant vos sorties vers `less`.
3. En utilisant la commande précédente et `sed`, extrayez la séquence nucléique complète du chromosome I de *S. Cerevisiae* sans aucun autre caractère que `a`, `t`, `c` et `g`.
4. Pour enlever également les retours à la ligne dans la séquence, vous pouvez ajouter ce filtre à votre précédente commande

   ```
   sed -r ':a;N;$!ba;s/\n//g'
   ```

   Cette utilisation de `sed` sort bien évidemment du cadre de ce cours.

3.5 Dé-htmliseur

Sauvez une page web de votre choix et observez le code html dans *gedit*. Créez avec `sed` un « dé-htmliseur », c'est-à-dire une commande qui retire toutes les balises HTML.

En HTML, les balises sont de la forme `<balise>` ou `</balise>`. Vous devez donc détruire tous les caractères compris entre < et >.

4 Commandes et processus

4.1 Filiation des processus

1. Ouvrez un terminal et imprimez à l'écran la filiation des processus. Repérez alors dans cet arbre le processus du terminal dans lequel vous vous trouvez (votre terminal s'appelle `gnome-terminal`). Où se trouve la commande que vous avez lancée pour afficher la filiation ?
2. Lancez un *xterm* à partir de votre terminal (avec la commande `xterm`). Dans ce nouvel *xterm*, imprimez de nouveau à l'écran l'arbre des processus et repérez où se trouve cet *xterm*. Tuez le terminal père (à l'aide de la petite croix). Que s'est-il passé ? Pourquoi ?

4.2 État des processus

1. Lancez l'éditeur de texte *nedit* à partir d'un terminal (juste en tapant `nedit`) et saisissez une phrase. Revenez dans le terminal et tentez de lancer la commande `ls`. Que se passe-t-il ? Pourquoi ?
2. Stoppez le processus *nedit* en tapant `Ctrl-z` dans le terminal. Retournez dans la fenêtre *nedit* et tentez de saisir du texte. Que se passe-t-il ? Pourquoi ?
3. Lancez un nouveau *nedit*. Stoppez-le à l'aide de `Ctrl-z`. Affichez dans votre terminal les processus en cours à l'aide de la commande `jobs`. Dans quel état sont ces jobs ?
4. Lancez un troisième *nedit*, mais cette fois-ci en arrière plan (en tapant `nedit &`). Imprimez les processus de votre terminal avec `jobs`. Quelle est la différence entre les deux premiers *nedit* et le troisième ? Comment l'expliquez vous ?
5. Tuez le premier *nedit* avec la commande `kill` et son numéro de job.
6. Passez en arrière-plan le deuxième *nedit* avec la commande `bg`. Contrôlez l'état de vos deux *nedit* avec la commande `jobs`.
7. Tuez les deux processus *nedit* restants avec la commande `kill`. Pensez à `jobs -l` pour obtenir leur PID.

4.3 État des processus (suite)

1. Créez un petit programme en Python que vous appellerez `infinite` et qui affiche des nombres de 1 à l'infini.
2. Lancez `infinite`. Stoppez-le à l'aide de `Ctrl-z`. Contrôlez son état avec `jobs`.
3. Repassez `infinite` en avant-plan avec la commande `fg`. Le programme affiche-t-il les nombres depuis 1 ? Pourquoi ? Tuez `infinite` avec `Ctrl-c`.

4.4 Priorités des processus

1. Lisez la page de manuel de la commande `yes`. Imprimez à l'écran la phrase de votre choix avec cette commande. Tuez `yes` avec `Ctrl-c`.
2. Lancez votre programme `infinite` en arrière plan et sans qu'aucune sortie n'apparaissent à l'écran (utilisez le trou noir de l'ordinateur `/dev/null`). Contrôlez que `infinite` tourne effectivement avec `top`. Quel est son *nice level* ?
3. Lancez un *yes* en silencieux (sans sortie à l'écran) et en arrière-plan avec un *nice level* de +10 (commande `nice`). Relancez `top`. Quel est le niveau de priorité du *yes* par rapport à `infinite` ?

4. Lancez un autre *yes* en silencieux et en arrière-plan avec un *nice level* de +19. Contrôlez le avec `top`. Quels sont les niveaux de priorité des trois processus ?
5. Modifiez le *nice level* de `infinite` à +10 (commande `renice`). Lancez `top`. Observez bien les deux processus de même *nice level* (à +10). L'un est-il plus prioritaire que l'autre ?
6. Tentez de modifier le *nice level* du second *yes* (celui que vous avez lancé à +19) à -15. Que se passe-t-il ? Pourquoi ?

Rappel : on peut mettre à jour l'affichage de la commande `top` avec la touche `espace` et quitter avec la touche `q`.

5 Variables et variables d'environnement

5.1 Affectation et portée des variables

1. En affichant le contenu de la variable d'environnement `SHELL`, vérifiez que votre *shell* actif est bien bash.
2. Affichez le contenu des variables d'environnement `PATH`, `USER`, `HOME`, `HOSTNAME`, `TERM`. Affichez toutes les variables d'environnement en une seule commande.
3. En bash, affectez la valeur `3` à la variable `u` et affichez le contenu de `u`.
4. Dans le même terminal, lancez un *xterm* et affichez le contenu de `u`. Que se passe-t-il ? Pourquoi ?
5. En bash, affectez la valeur `Salut tout le monde` à la variable d'environnement `MESSAGE`. Contrôlez la valeur de `MESSAGE`. Vérifiez que `MESSAGE` est bien une variable d'environnement avec `env` et `grep`. Lancez un *xterm* dans lequel vous afficherez à nouveau la valeur de `MESSAGE`. Y accédez-vous ? Pourquoi ?

5.2 Caractères spéciaux

1. Exécutez les deux commandes suivantes :

   ```
   echo 'Bonjour $HOSTNAME, je suis $USER'
   ```

   et

   ```
   echo "Bonjour $HOSTNAME, je suis $USER"
   ```

   Comment expliquez-vous la différence ?
2. Affichez le texte suivant avec des doubles guillemets :

   ```
   L'usage des ", \, #, $, etc... est interdit !
   ```

6 Programmation shell

6.1 Initiation à la programmation bash

1. Écrivez un script bash qui affiche les trois arguments qui lui sont donnés.
2. Écrivez un script bash avec une boucle `for` qui affiche tous les fichiers du répertoire courant sous la forme

   ```
   le nom du fichier 1 est ...
   le nom du fichier 2 est ...
   ...
   ```
3. Créez un script bash qui prend en argument le nom d'un fichier genbank. Le script doit afficher le numéro d'accès (champ `ACCESSION`) suivi de la définition (champ `DEFINITION`) dans le format

   ```
   numero_accession :: definition
   ```

Le script doit aussi afficher un message d'erreur si aucun argument n'est fourni.

4. Depuis la rubrique « Documents et liens » de la plateforme Didel, téléchargez le fichier `genomes.tgz`, copiez-le dans votre répertoire et décompressez-le. Le répertoire `genomes` contient des fichiers genbank tronqués de quelques organismes (les dix premières lignes seulement).
   Écrivez un script bash qui copie les fichiers genbank de staphylocoques dans un répertoire `staphylo` que vous créerez.

6.2 Bouquet final : création d'un script de conversion genbank vers fasta

Une séquence au format fasta présente la forme suivante :

```
>ligne de commentaire
atctggatgtatctggatgtatctggatgtatctggatgtatctggatgtatctggatgtatctggatgt
tgtatctggatgtatctggatgtatctggatgtatctggatgtatctggatgtatctggatgtatctgga
atctggatgtatctggatgt... (60 caractères de long)
```

Créez un script bash `gbk2fasta.sh` qui convertit un fichier genbank entré en argument en fichier fasta. Le script affichera un message d'erreur si aucun argument n'est fourni et vérifiera l'existence du fichier genbank (test `-e`). La ligne de commentaire contiendra le numéro d'accès (champ `ACCESSION`) suivi de la définition (champ `DEFINITION`) du génome.

N'hésitez pas à réutiliser vos filtres et vos scripts précédents.

Annexe : quelques ouvrages de référence

Unix in a Nutshell, A. Robbins, O'Reilly Media, 4e édition, 2005.

Mastering Regular Expressions, J. E. F. Fridel, O'Reilly Media, 2e édition, 2002.

sed & awk, D. Dougherty et A. Robbins, O'Reilly Media, 2e édition, 1997.

Learning the bash Shell, C. Newham et B. Rosenblatt, O'Reilly Media, 3e édition, 2005.

Le shell bash, C. Newham et B. Rosenblatt, O'Reilly Media, 3e édition, 2005.
(version française de *Learning the bash Shell*)

Advanced Bash-Scripting Guide
`http://www.tldp.org/LDP/abs/html/`