Outils informatiques

1. Commandes de manipulation de fichiers

DESS TEXTE

Introduction

Présentation des outils permettant :

- d'effectuer la recherche de fichiers contenant un motif particulier
- de réaliser des substitutions d'expressions dans un ensemble de fichiers

au moyen d'une commande DOS (ou d'un shell Unix)

Plan du chapitre 1

- Présentation de la commande *grep*
- Présentation de la commande *sed*
- Utilisation de l'éditeur *XEmacs*
- Outils de programmation en Perl

Présentation de la commande grep (1)

- *grep* permet de rechercher des lignes d'un (ou plusieurs) fichier(s) contenant un motif décrit par une expression régulière.

- **Attention :** le langage utilisé pour définir une expression régulière de grep est différent de celui de Perl.

- L'expression régulière est placée entre guillemets.
  L'appel à la commande *grep* se fait via la syntaxe suivante :
  - grep <option(s)> "<expression>" <Fichier(s)>

- Pour obtenir de l'aide : *grep –help*

Présentation de la commande grep (2)

- Parmi les options les plus importantes de *grep*, on peut noter :
  - **-i** : indique à *grep* de traiter indifféremment majuscules et minuscules.
  - **-w** : indique à *grep* que l'expression doit correspondre à des mots entiers.
  - **-x** : indique à *grep* que l'expression doit correspondre à des lignes entières.
  - **-v** : indique à *grep* d'inverser la sélection (lignes ne correspondant pas au motif).

Présentation de la commande grep (3)

- (suite)
  - **-f <fichier>** : indique à *grep* d'utiliser l'expression présente dans le fichier <fichier>.
  - **-n** : indique à *grep* de renvoyer le numéro de ligne avec la ligne sélectionnée.
  - **-m N** : indique à *grep* d'arrêter après N concordances.
  - **-H** : indique à *grep* d'afficher en préfixe le nom du fichier contenant la ligne.
  - **-r** : indique à *grep* d'effectuer une recherche récursive (sous-répertoires).

Présentation de la commande grep (4)

Remarques :

- Les fichiers à parcourir peuvent être défini par une expression régulière, exemple : *grep “toto” *.txt*.

- La sortie de *grep* peut servir d’entrée à une autre commande $\rightarrow$ utilisation des redirections.

- Les expressions utilisées dans *grep* pour définir un motif sont limitées :

  - absence de la disjonction de séquences, de l’optionnalité et de la présence d’au moins une occurence.
  - impossibilité d’exprimer le caractère de retour à la ligne.

Solution à ces problèmes : *egrep*

Présentation de la commande egrep

- *egrep* reprend la syntaxe de la commande *grep*
- *egrep* autorise l'emploi d'expressions régulières étendues, i.e. ?
  - **disjonction** : symbole **|**,
  - **optionnalité** : symbole **?**,
  - **itération** : symbole **+**.

(voir exercices)

Présentation de la commande sed (1)

- *sed* permet d'appliquer une commande portant sur une expression régulière à un fichier.

- Plus précisément, *sed* permet de :
  - de **substituer**,
  - de **détruire**,
  - d'**ajouter, insérer et modifier** un texte correspondant à une expression régulière donnée.

- L'appel à la commande *sed* se fait via la syntaxe suivante :
  - sed <commande_portant_sur_une_ER> <Fichier_à_traiter>

- Pour obtenir de l'aide : *sed –help*

Présentation de la commande sed (2)

Substitution

- La commande de substitution **s** permet de substituer un texte à une expression régulière.

- Sa syntaxe est :
  - sed [adresse]**s**/ER/texte/[options] <fichier>

- Exemple :
  - sed /rendez\-vous/**s**/soleil/lune/g fichier.in > fichier.out

- Remarque : g indique que sed doit subsituer toutes les occurences du motif, possibilité de les désigner explicitement 0,..,9.

Présentation de la commande sed (3)

Destruction

- La commande de destruction **d** permet de supprimer un texte correspondant à une expression régulière.

- Sa syntaxe est :
  - sed /ER/**d** <fichier>

- Exemple :
  - sed /^$/**d** fichier.in > fichier.out

- Remarque : suppression des lignes 1 à 10 : sed 1,10d fichier.in (et non sed /1,10/d fichier.in)

Présentation de la commande sed (4)

Ajout, insertion, modification

- Les commandes d'ajout, insertion et modification de *sed* se saisissent sur plusieurs lignes.

- Leur syntaxe est :
  - sed [adresse]commande\

- Avec :
  - *adresse* peut être une ER (/blabla/), un numéro de ligne ($), un intervalle (1,4),
  - *commande* peut être **a**(ajout), **i**(insertion) ou **c**(modification).

- Exemple :
  - sed 1i\
    Montitre

Utilisation de l'éditeur XEmacs (1)

- Plus qu'un éditeur, *XEmacs* est un environnement de développement pour les langages de programmation tels que C, Lisp ou Perl.

- Il intègre des facilités d'édition, de visualisation et parfois même de déboggage.

- A chaque fichier ouvert correspond un buffer, possibilité d'afficher plusieurs buffers dans la fenêtre de *XEmacs*.

- Dans *XEmacs*, les différentes options sont atteignables via des menus (et pour certaines des raccourcis clavier).

- **Remarque** : *XEmacs* dispose d'un mode graphique spécifique à Perl (coloration notamment), activable via **Esc - X** puis taper **perl-mode**.

Utilisation de l'éditeur XEmacs (2)

Parmi les raccourcis les plus utiles de *XEmacs*, on trouve :

- **Ctrl - X Ctrl- S** : « enregistrer »,
- **Ctrl - X Ctrl- W** : « enregistrer sous »,
- **Ctrl - X Ctrl- B** : visualiser la liste des buffers ouverts,
- **Ctrl - X 2** : découper la fenêtre dans le sens de la hauteur (2 buffers),
- **Ctrl - X b** : changer de buffer,
- **Ctrl - Espace** : poser une ancre de sélection,
- **Esc - W** : « copier »,
- **Ctrl - Y** : « coller »,
- **Ctrl - W** : « couper ».

Outils de programmation en Perl (1)

- directive « use strict ; » : vérification de la déclaration préalable des variables utilisées,
- directive « use diagnostics ; » : informations sur le résultat de l'interprétation,
- directive « use vars ; » : déclaration de variables globales,
- possibilité de passer des arguments au programme Perl via la ligne de commande, ceux-ci sont alors stockés dans le tableau @ARGV.
- Exemple : (ouverture en lecture du fichier passé en argument)
  - open(FICHIER,"<$ARGV[0]") || die "Erreur à l'ouverture de $ARGV[0], $! \n" ;

Outils de programmation en Perl (2)

Utilisation d'options à l'appel de programmes Perl

- intérêt : proposer à l'utilisateur de définir certains paramètres du programme (modifier la sortie, demander de l'aide, etc).

- fonctionnement : utilisation du module **Getopt** :
  - use Getopt : :Std ;
    getopts('oif :') ;
  - ici la fonction *getopts* déclare trois options au programme (o, i et f), la dernière étant suivie d'un argument,
  - pour chacune des options déclarées, *getopts* crée une variable $opt_X (X étant l'option) contenant sa valeur (booléenne ou non),
  - attention : en présence de la directive *use strict ;* il est nécessaire de déclarer ces variables d'options au moyen de
    use vars qw($opt_o $opt_i, $opt_f) ;

Outils de programmation en Perl (3)

Exemple d'utilisation de *Getopt : :Std* :

```
use Getopt::Std;
use strict;
our ($opt_h, $opt_o);
getopts('ho:');
my ($arg1_in,$arg2_out)=@ARGV;
if (!$arg1_in || ($opt_h == 1))
  { print "usage:...;" exit(1); }
if (defined($opt_o))
  {...}
```