

# TD 3 d'Environnement Logiciel

Mardi 21 septembre 2004

## Entrée sortie, encore, ou le retour du TD 2.

- Écrivez une version en C de `cat` en utilisant uniquement les fonctions d'entrée sortie de la bibliothèque standard `fread` et `fwrite`, et en lisant et écrivant des blocs d'un caractère.
- Écrivez une seconde version de `cat` en utilisant uniquement les appels systèmes `read` et `write`.
- En utilisant la commande `time` comparez les performances des deux versions pour une lecture/écriture d'un fichier raisonnablement volumineux (par exemple `/lib/libc.so.6` en n'oubliant pas de rediriger la sortie standard vers `/dev/null`).
- Interprétez les résultats obtenus. Quels sont les commandes de plus bas niveau: `read/write` ou `fread/fwrite`?
- Essayez le programme suivant:

```
#include <stdio.h>
#include <unistd.h>

int main ()
{
    fwrite ("Hello ", sizeof(char), 6, stdout);
    write (STDOUT_FILENO, "World !", 7);

    return 0;
}
```

Le résultat est-il surprenant?

## Scripts shell

- Dans un script `bash`, écrivez une fonction qui affiche le nombre de lignes du fichier donné en premier argument. Vous avez le droit d'utiliser `wc` (en nettoyant la sortie, `wc` affiche généralement le nom du fichier passé en argument, ce que nous ne voulons pas).
- Modifiez votre fonction pour pouvoir afficher au choix le nombre de lignes, de mots ou de caractères. On indiquera ce choix par `L`, `M` et `C`, respectivement. Les courageux ont le droit d'essayer un `case` pour laisser plus de liberté à l'utilisateur (par exemple permettre "un mot qui commence par L" plutôt que d'imposer simplement L).
- Utilisez cette fonction dans le script pour qu'il affiche le nombre total de lignes dans les fichiers sources C (y compris les headers) du répertoire courant.

- Modifiez votre script pour inclure les fichiers C des sous-répertoires, de façon récursive.
- Modifiez votre script pour n'inclure que les fichiers contenant le mot 'copyright' sans tenir compte de la casse.
- Modifiez votre script pour remplacer dans la foulée dans tous les fichiers visités les lignes contenant 'copyright' par une ligne de copyright de votre choix. (*exercice à tester sur une copie de travail des sources de GMP par exemple.*)

### Un peu d'awk

- Écrivez un script awk qui simule `wc`. Il devra lire l'entrée standard et n'afficher qu'une ligne de sortie contenant respectivement le nombre de lignes, de mots, et de caractères lus.
- Écrivez un script awk qui renverse l'entrée standard: les champs de chaque ligne doivent être affichés sur la sortie standard dans l'ordre inverse dans lequel ils sont lus. On ne fait pas d'hypothèse *a priori* sur le nombre de champs sur une ligne, qui peut varier d'une ligne à l'autre.

### Un peu de sed

- Rotation 13. La rotation 13 est un codage par substitution qui n'offre aucune sécurité mais permet de masquer temporairement du texte dans un message, par exemple la solution d'une énigme envoyée par mail. On fait correspondre à chaque lettre la lettre décalée de 13 positions dans l'alphabet (a devient n). À l'aide de `sed` écrivez une fonction shell `rot13` que vous chargerez automatiquement dans votre `.bashrc` (les fonctions s'exportent avec `export -f`). Votre fonction devra traduire correctement les majuscules. Votre fonction devra se comporter comme un filtre entre l'entrée standard et la sortie standard (on ne demande pas d'accepter des fichiers en argument sur la ligne de commande, mais c'est une amélioration possible si vous avez le temps).
- Et la fonction inverse? Vérifiez que `rot13 | rot13` produit bien le résultat attendu.
- Petit raffinement: ajoutez une rotation 5 sur les chiffres.
- Les fichiers sources de GMP commencent tous ou presque par un commentaire donnant des informations sur le fichier. Écrivez un script `sed` qui enlève ces quelques lignes.

### Du perl?

- Codage en base64. Pour des raisons remontant à la nuit des temps, les fichiers binaires inclus en attachement dans les emails ne sont pas inclus tels quels, mais codés en base64. Le problème est que les caractères non affichables (comme le `0x07`, voir `man ascii`) ne passent pas. On remplace donc 3 octets ( $3 * 8$  bits) du flux d'entrée par 4 octets en sortie en ne choisissant les octets de sortie que parmi 64 octets "imprimables" (lettres majuscules et minuscules, chiffres, + et /) avec le = comme symbole de remplissage. Voir la RFC 3548 pour plus d'informations. Un alphabet de 64 caractères permet de représenter 6 bits d'information ( $2^6 = 64$ ) et  $6 * 4 = 24$  donc le compte est bon.

Le script perl suivant est un filtre qui code l'entrée en base64:

```
use MIME::Base64;

while (<>) {
print encode_base64($_);
}
```

et celui-ci fait le décodage:

```
use MIME::Base64;

while (<>) {
print decode_base64($_);
}
```

- Écrivez ces deux scripts perl dans des fichiers que vous rendrez exécutables. N'oubliez pas d'indiquer en première ligne le chemin vers l'interprète perl.
- Un codage révolutionnaire: un code base64 valide reste valide après passage par rot13. Écrivez un filtre qui fait la transformation suivante:

$$\text{clair} \rightarrow \text{base64}(\text{clair}) \rightarrow \text{rot13}(\text{base64}(\text{clair})) \rightarrow \text{base64}^{-1}(\text{rot13}(\text{base64}(\text{clair})))$$

- Écrivez un script qui demande à l'utilisateur un nom de fichier et effectue le codage décrit ci-dessus en ajoutant `.sec` au fichier produit (essayez de ne pas planter salement en cas d'entrée incorrecte...).
- Et pour le décodage? :-)
- Le codage par rot13 n'étant pas d'une sûreté exceptionnelle, on va utiliser mieux. On suppose qu'une clef est une permutation (avec éventuellement des points fixes) de l'alphabet des 64 caractères définis pour la base64, et cette permutation va remplacer la fonction rot13 dans le diagramme ci-dessus.. La clef est donnée dans un fichier comme une suite de lignes contenant deux champs: un caractère et son image. Écrivez un script shell permettant à l'utilisateur d'indiquer un fichier clef, un fichier à coder et si l'opération à effectuer est le codage ou le décodage.

Comparez vos résultats avec le script perl solution suivant:

```

use MIME::Base64;

use strict;
use warnings;

my $clef = shift || die "Veuillez entrer une clef.";
my $sens = shift || die "Veuillez entrer un sens (1 ou 2).";
my %perm;

open KEY, $clef;
while (<KEY>) {
    if (/s*(\w)s*(\w)/) { $perm{$1} = $2; }
}

sub permute {
my ($in) = (@_);
if (defined($perm{$in})) {
return $perm{$in};
}
return $in;
}

close KEY;
%perm = reverse(%perm) if ($sens eq "1");
my $ligne;
while ($ligne = <STDIN>) {
    print decode_base64(join('', map { permute($_) }
split(/ */, encode_base64($ligne))));
}

```

aussi disponible sur <http://nowwhat.fousse.info/tmp/code>.