Explications du code

Fonctions

Comme dans d'autres langages de programmation, on peut écrire des fonctions, série d'instructions, qu'on peut ensuite appeler plusieurs fois dans le code sans avoir à tout réécrire. Cela permet de rendre le codage plus pratique et plus clair.

Ici une fonction d'affichage de l'aide, appelée lignes 37 et 66, et une fonction d'affichage en cas d'erreur, appelée lignes 25, 31, 61, 86, 92, 121, 126 et 131.

La commande exit induit une interruption du script.

Récupération des options et gestion des erreurs de saisie

- Concernant le choix de la manière de procéder ici, voir la note « Les commandes 'getopt' et 'getopts' » en fin de document -

Lignes **23** à **26** : on vérifie qu'au moins un argument a été passé au script, si ce n'est pas le cas on interrompt avec un message d'erreur spécifique.

Ligne **28** : on teste le premier caractère de l'argument en tête de liste, si c'est un '-' il s'agit d'une option et on entre dans la boucle (Cette boucle permet le traitement des options écrites séparément, par exemple : *./langstat.sh -l -t nom_du_fichier*).

Lignes **29** à **32** : on vérifie que '-' est bien suivi d'au moins un caractère ; ce test n'est pas en soi nécessaire (le cas pourrait être traité par le '*' dans le 'case' à venir) mais ainsi cela permet d'afficher un message d'erreur plus précis.

- Concernant la commande 'sed' voir la note correspondante en fin de document -
Ligne **34** : on va passer en revue un à un, tous les caractères de l'argument en cours de traitement, sauf bien sûr le '-' départ (Ceci permet de traiter les cas où les options sont collées comme par exemple : *./langstat.sh -ioy nom_du_fichier*).

Ligne **35** à **94** : en fonction des caractères passés en revue, on définit certaines variables qui serviront par la suite au traitement du fichier, chacune par leur existence indique que l'option correspondante a été activée.

Lignes **57** à **88** : on traite les cas des options longues. Là on récupère tout ce qu'il y a dans l'argument après '--', de ce fait on ne va plus traiter les caractères un à un, donc une fois l'option longue vérifiée il faut sortir de la boucle 'for' entamée ligne 34. Ceci se fait à l'aide de l'instruction 'break'. Il est donc à noter qu'on ne peut coller une autre option à la suite d'une option longue. Aussi comme pour les lignes 29 à 32, le test des lignes 59 à 62 n'est pas nécessaire mais permet plus de précision sur le message d'erreur.

Exemples :

./langstat.sh -io --lower-only nom_du_fichier → *valide*
./langstat.sh -io--lower-only nom_du_fichier → *invalide*
./langstat.sh -io-lower-only nom_du_fichier → *invalide*
./langstat.sh -iolower-only nom_du_fichier → *invalide*
./langstat.sh --plus-spec -i -t nom_du_fichier → *valide*
./langstat.sh --plus-specit nom_du_fichier → *invalide*

Ligne **96** : le 'shift' indique qu'on a traité le premier argument de la liste, celui-ci est passé et son suivant devient le nouveau premier argument ($2 devient $1) qu'on teste à son tour, ligne 28. Et ainsi de suite jusqu'à arriver à un argument qui n'est pas une option (qui ne commence pas par '-') ou qu'il n'y ait plus d'arguments.

Préparation des caractères à analyser

La variable 'alphabet' est celle qui contient les caractères à analyser.

Ligne **102** : Par défaut, 'alphabet' prend la valeur 'uppercase', les majuscules de A à Z

Ligne **104** à **106** : si 'iflag' n'est pas vide (c'est-à-dire l'option -i est activée) alors 'lflag' est vidé (c'est-à-dire l'option -l est désactivée). Ceci permet d'éviter des doublons dans les statistiques si par mégarde l'option -i est demandé en même temps que -l. En effet, par sa définition l'option -i inclut l'option -l, dans une implémentation parallèle.

Ligne **108** à **110** : si l'option -l est activée alors on fait une concaténation de 'alphabet' et 'lowercase', cette dernière contenant les minuscules de a à z

Ligne **112** à **114** : si l'option -L est activée alors 'alphabet' prend la valeur 'lowercase', on n'analysera donc que les minuscules

NB : l'ordre des conditions sur 'iflag', 'lflag' et 'Lflag' n'est pas anodin. Le test 'iflag' doit absolument venir avant le test 'lflag' pour annuler ce dernier en cas de conflit. Le test 'Lflag' vient après 'lflag' par choix de donner la priorité à l'option -L sur -l, ainsi -L annule également -l, d'une manière différente à -i. Il n'y a pas de conflit entre -i et -L (*voir les notes de la documentation doc_langstat.pdf pour détails à ce sujet)*.

Ligne **116** : on fait une concaténation de 'alphabet' avec toutes les variables d'ajout possibles, venant de l'option -y et des options longues --plus-… Dans le cas où ces (ou certaines de ces) options n'ont pas été activées, les variables correspondantes n'ont pas été définies au préalable, de fait elles sont vides et n'ajoutent rien à 'alphabet' comme voulu.

Traitement

Comme expliqué plus haut, on sort de la boucle de récupération des options si il n'y a plus d'option à analyser. Arrivé ici, $1 est donc sensé contenir le nom du fichier à analyser. Il s'agit de le vérifier.

Lignes **119** à **132** : Un premier test pour vérifier que l'argument n'est pas vide. Un second pour vérifier que l'argument désigne bien un fichier valide. Un troisième pour s'assurer des droits d'accès au fichier.

Lignes **134** à **136** : On regarde si il y a des arguments restants au-delà de $1, cela n'empêche pas le script de fonctionner à partir du moment où on a déjà vérifié qu'on avait un nom de fichier valide, donc on n'interrompt pas le script mais on signale que les arguments suivants ne sont pas pris en compte, laissant sous-entendre qu'il y a une erreur non fatale de syntaxe.

Lignes **138** à **146** : on prépare la création de deux fichiers temporaires, le premier pour récupérer le flux d'entrée, le second pour le flux de sortie. On vérifie le cas improbable (restes d'une erreur majeure survenu pendant un précédent appel au script, type d'erreur qui a très peu de chances d'être liée au script en lui-même) où les fichiers existent déjà et on les supprime le cas échéant pour repartir sur des bases propres.

Lignes **148** à **156** : on teste l'activation de l'option -o et de l'option -t. A noter que le test de l'option -t ne s'effectue que si l'option n'est pas active, c'est-à-dire que l'option -o outrepasse l'option -t en cas d'activation simultanée.

→ *Cas -o et -t non activée* → le fichier temporaire d'entrée est une copie directe du fichier passé en argument

→ *Cas -o non activée et -t activée* → on manipule le fichier à analyser avant d'envoyer les données dans le fichier temporaire d'entrée (ajout d'un espace après chaque apostrophe et trait d'union, puis remplacement des espaces par des sauts de ligne - *voir notes sur 'sed' plus bas*, et *les notes de la documentation doc_langstat.pdf concernant une petite limitation sur l'option -t*).

→ *Cas -o activée* → là aussi on manipule le fichier à analyser (ajout d'un saut de ligne après chaque caractère)

- 5 petites lignes qui constituent le cœur du script, et font le plus gros du travail -
Ligne **158** : on parcourt un à un, les caractères à analyser contenus dans 'alphabet'

Ligne **159** : Pour chacun de ces caractères, on ajoute une ligne *'NOMBRE-X'* dans le fichier temporaire de sortie où

→ *NOMBRE* est le nombre de lignes du fichier d'entrée dans lesquelles le caractère est apparu.
'grep' (avec ou non l'option -i définie par 'iflag') cherche le caractère 'char' dans le fichier temporaire d'entrée 'intmp' et envoie ce qu'il trouve par le pipe | à 'wc -l' qui détermine le nombre de lignes.

→ *X* est le caractère qui vient d'être analysé

Ligne **162** : le fichier temporaire de sortie contient alors les lignes correspondant à chacun des caractère qu'on voulait analyser, 'sort' trie ces lignes qui commencent toutes par un nombre, avec l'option -n pour le tri numérique et -r pour un tri décroissant. Le 'sed' n'est pas nécessaire ici, il ne sert qu'à changer la pagination en inversant le caractère et le nombre, en ajoutant des espaces, et en remplaçant le tiret par '='. *Les lignes prennent la forme 'X = NOMBRE'.*

Ligne **163 -164** :on supprime les fichiers temporaires avant de sortir.

Notes

> La commande sed

Cette commande est un éditeur de flux et semble relativement complexe dans son ensemble, mais l'utilisation que j'en fais ici est somme toute assez simple. Il s'agit de comprendre ce que signifie l'argument et pour cela, de bien en repérer la forme :

s/donnée_en_entrée/donnée_en_sortie/g

s indique qu'on procède à une substitution, c'est-à-dire que *donnée_en_sortie* va remplacer *donnée_en_entrée*.

g indique qu'on veut opérer cette substitution de manière récursive, c'est-à-dire sur l'ensemble des données d'entrée.

/ sert simplement de séparateur.

donnée_en_entrée et *donnée_en_sortie* peuvent prendre des formes très diverses, cela peut être un caractère, une chaîne de caractères, une expression rationnelle...

Dans le script ci-présent, en commençant par les formes les plus simples :

Ligne **152** (1) (2) :

sed "s/'/' /g"

→ *l'apostrophe en entrée devient apostrophe suivi d'un espace en sortie et avec la commande récursive g cela se traduit par* "ajout d'un espace après **toutes** les apostrophes"

sed 's/-/- /g'

→ *de la même manière* "ajout d'un espace après **tous** les tirets"

(À noter, dans le second cas l'argument est entre ' ', alors que dans le premier on a " ", c'est simplement que ' ' suffit dans le second cas, et dans le premier " " est nécessaire pour traiter les apostrophes sans antislash supplémentaire)

Ligne **152** (3) :

sed 's/ /\n/g'

→ *l'espace en entrée devient \n, retour à la ligne, en sortie, et récursivement avec g cela se traduit par* "remplace ***tous*** les espaces par des retours à la ligne" *ou* "place un et ***un seul mot par ligne***"

(À noter que pour un simple échange de caractères seuls, la commande 'tr' aurait pu faire ça aussi avec *tr ' ' '\n'*)

Formes un peu plus complexes :

Lignes **34**, **155** et **158** :

sed 's/\(.\)/\1\n/g'

donnée_en_entrée : \(.\)
→ il s'agit d'une expression rationnelle, \(et \) ne sont en fait que des délimiteurs, des balises qui définissent un champ numéroté en fonction de sa position dans l'expression, ici il n'y a qu'un seul champ qui aura le numéro 1. Le contenu du champ ici est '.' un métacaractère qui signifie "n'importe quel caractère seul".
Un début de traduction pourrait être "remplace chaque caractère quel qu'il soit par…"

donnée_en_sortie : \1\n
→ \1 désigne tout simplement le champ numéro 1 défini en donnée d'entrée c'est-à-dire dans le cas présent qu'on garde ce qu'on avait en entrée, ce à quoi on ajoute ce qui suit, soit ici \n qui est toujours le retour à la ligne.

Traduction complète "ajout d'un retour à la ligne après chaque caractère" ou "place un et ***un seul caractère par ligne***"

Ligne **162** :

sed 's/\([0-9]\)-\(.\)/\2 = \1/g'*

donnée_en_entrée : \([0-9]\)-\(.\)*
→ champ \1 vaut \([0-9]\) → [0-9] désigne "n'importe quel caractère seul parmi 0-9" soit "n'importe quel chiffre". * est un autre métacaractère qui s'applique au morceau d'expression placé juste avant lui et signifie "ce qui précède en un nombre quelconque de fois ou jamais", donc ici "[0-9] un nombre quelconque de fois ou jamais", soit "n'importe quelle série de chiffres ou rien", soit "n'importe quel nombre ou rien".*
→ champ \2 vaut \(.\) vu plus haut
Ceci mis bout à bout, un début de traduction pourrait être "Quand les données en entrée sont de la forme *NOMBRE-X*, remplace par…"

donnée_en_sortie : \2 = \1
→ place le champ 2 en premier suivi d'un espace puis '=' puis un autre espace et enfin le champ 1, c'est-à-dire que le nombre et le caractère en entrée sont intervertis et séparés par ' = ' au lieu de '-'

Traduction complète "remplace toute donnée de la forme *NOMBRE-X* par la forme *X = NOMBRE*"

> Les commandes getopt et getopts

Ces deux commandes permettent de récupérer les options dans la ligne de commande, j'ai choisi de ne pas me pencher plus avant sur leur fonctionnement pour les raisons suivantes :

→ *getopts* semble d'utilisation assez simple, mais ne gère pas les options longues, et surtout n'est pas présente nativement sur ma distribution, et donc potentiellement absente aussi ailleurs.

→ *getopt* paraît plus compliquée à utiliser et nécessite l'appel à une autre commande, et en l'état de mes connaissances, m'aurait à mon sens plus compliqué que facilité la tâche. De plus, bien que présente nativement sur ma distribution, je n'ai pas la certitude qu'elle le soit sur les autres.