# Météo 2019 - 2020 – Décryptage

Attention : tous les décryptages ont été réalisé sur la même base. Il y aura de nombreuses répétitions, il est probablement inutile de tous les consulter, peut être juste pour avoir si il y a des différences…

## Déroulement

### Constatations

Dans cette partie du sujet, vous apprenez qu'il faut faire une seule commande qui va prendre en charge toutes les phases qui y sont décrites.

Quand dans le sujet il est écrit "la notion de phase signifie que j'exécute une commande unix avec des options et la commande fait le travail demandé. Si je souhaite exécuter une nouvelle phase j'exécute une nouvelle commande unix". C'est une phrase importante car elle signifie que je peux lancer et relancer la même commande avec des objectifs différents. Comme certaines phases vont dépendre de résultats provenant d'autre phases, il va donc falloir garder des informations en mémoire. Mais comme, la commande est relancée, c'est forcément sur le disque (dans des fichiers) car quand une commande s'arrête il ne reste plus rien de son exécution … sauf les fichiers qu'elle a créés, modifiés ou supprimés.

Quand dans le sujet il est écrit : " il est inutile de converser avec l'utilisateur, il n'y aura personne devant le clavier", cela signifie qu'il n'y a que les options pour donner des indications et s'il y a un souci il faut émettre un message 'erreur et s'arrêter proprement mais il est inutile de demander quelque chose à un utilisateur.

## Commande

Cette partie est importante car elle va donner les différentes options et leurs structures. Ce qui va former le squelette de l'application finale.

### Constatations structure

Plusieurs constatations :

* Chaque option commence comme d'habitude par le tiret et il n'y a qu'une lettre. Ce sont des options courtes, il n'y a pas les options longues.
* Certaines options n'ont pas de paramètre, d'autres 1 paramètre voire 2 paramètres. Dans le déroulement plus loin dans le document on peut voir qu'il est possible de soumettre plusieurs options à la même commande plusieurs fois des options, etc.

### Possibilités structure

Ecrire un script avec une fonction principale qui commence à la ligne 1 et se termine à la dernière.

Architecturer le scripte en utilisant des fonctions qui seront invoquées le moment venu.

Comme il y a un nombre élevé d'options, que le code dans les options n'est pas encore quantifiable mais probablement grand, que certaines options pourraient référencer ou partager des fonctionnalités donc ce n'est pas un script jouet (loin de là).

### Choix structure

Il va falloir organiser et surtout assurer un découpage qui permet facilement de construire une base que l'on enrichit au fur et à mesure des questions à traiter.

Chaque option sera traitée par une fonction interne qui va porter le nom de l'option (-i va appeler option\_i, -s va appeler option\_s, etc…). Initialement les fonctions seront 'vides' et par la suite on les renseigne.

### Constatations arguments

Il va falloir consommer les arguments intelligemment. Il n'y a pas d'ordre entre les options.

Deux possibilités :

* imbriquer savamment le traitement de l'option 1 puis l'option 2 puis l'option 3, ou l'option 1 puis la 2 et enfin la 3, ou l'option 2 suivit de la 1 et de la 3 etc… Cauchemar de lecture, redondances permanentes et … il sera impossible d'atteindre des arguments au-delà du 9 ième car le shell refuse cette notation ($10 ou $11).
* faire une boucle qui traite chaque option et ses paramètres en faisant défiler les arguments.

### Possibilités arguments

Impossible de faire autrement, c'est probablement non négociable quel que soit la commande car si une commande est riquiqui au début elle devient fatalement plus riche et l'imbrication s'effondre., donc il faut faire une boucle, laquelle ?

### Constatations boucle

Il y a deux boucles en shell (la troisième n'est jamais utilisée) : for et while

La boucle for est chouette car elle itère automatiquement sur les arguments mais n'offre pas le contrôle de l'itération. Elle n'est pas appropriée car elle va considérer que tous les arguments peuvent être des options et malheureusement le 'ville' qui est derrière -m n'est pas une option mais un paramètre à cette option. Les novices vont 'trier' (ignorer dans ce cas) les arguments en considérant qu'une option sera toujours un tiret avec une lettre et un paramètre n'aura jamais de tiret. C'est une erreur car une commande pourrait avoir des paramètres avec des tirets et là on entre dans une confusion certaine. Certes ce n'est pas le cas ici mais si déjà on fait un travail autant l'écrire proprement (surtout si l'effort n'est pas immense).

La boucle while est plus primitive mais elle donne le contrôle de l'itération. Le principe est fort simple, on boucle jusqu'à épuisement des arguments ($# <= 0), on traite l'argument $1 comme une option, dans chaque traitement des options on décale (shift) les arguments pour retirer les paramètres des arguments. Une vacherie de la part d'unix (je m'en passerais et vous aussi). Si vous faite un shift 3 et qu'il n'y a pas au moins 3 arguments le shift ne fait rien ce qui n'est pas le comportement souhaité (j'aurais préféré que shift supprime les arguments existant et laisse la liste vide). Il faudrait donc techniquement vérifier que le shift X est possible si le X > 1 ou enchainer plusieurs shifts de 1. En fait comme j'ai 0, 1 ou 2 c'est plus malin de faire deux shifts de 1 quand c'est nécessaire.

### Choix boucle

Avec les constatations précédentes, la conséquence est qu'il n'y a qu'une seule boucle utilisable : while malgré le truc casse pied du shift qui est 'bizarre'

### Constatations interprétation des options

Pour interpréter les options cela devient simple. Il faudra faire une sélection qui va se baser sur la présence du tiret et d'une lettre précise (s'il y a des options longues il faudra tenir compte de double tiret avec un mot).

### Possibilités interprétation des options

Un if est envisageable si le nombre d'option est faible (1 ou 2) mais c'est une erreur sur le long terme.

Un case est toujours envisageable. Chaque entrée sera une option autorisée. Il faudra juste prévoir le cas autre (\*) pour finir avec une fonction d'erreur.

### Choix interprétation des options

Pas de choix : le case.

Comment traiter les paramètres. Relativement facile. Il faut récupérer la valeur des paramètres et les transmettre aux fonctions prévues. Pour éviter des problèmes avec des paramètres contenant des espaces il faudra utiliser des doubles quottes pour chaque invocation d'un "$quelquechose"

### Implémentation

Voici un premier jet du script, avec un dernier décryptage lié aux lignes.

Ligne 1 : je commence par le shebang pour le bash

Ligne 3, 7, 11, etc… J'utilise la manière moderne de bash de déclarer une fonction. Il y a d'autres variantes équivalentes.

Lignes 3 à 5 : une fonction non prévue mais pratique. Elle permet d'imprimer sur la sortie d'erreur (2>&1) un texte en paramètre ($1). Cela va permettre d'éviter de recopier la redirection partout où j'ai besoin de cette fonctionnalité (partout au début).

Lignes 7 à 38 : chaque fonction, comme prévue, avec l'impression du texte et des paramètres. Il faut noter que les paramètres dans la fonction commencent bien par $1, $2, $3… Même si quand on invoque la fonction on transmet des $2 ou $3. C'est la différence classique entre les paramètres dans la fonction et lors de l'appel. Un effet de bord classique et agaçant. Si vous faites une fonction vide vous avez un message d'erreur car une fonction ne peut pas rien faire … Le fait d'avoir invoqué la fonction messageOnStderr dans chaque fonction fait disparaitre ce problème le temps de développer le code indispensable.

Lignes 39 à 51 : mouais je pousse mais autant faire propre, on peut faire plus simple. Je créé un bloc (ligne 40 et 50) qui va supporter une seule fois la redirection.

Ligne 53 : je force une sortie avec erreur (exit 1) pour briser les traitements et j'en profite pour signaler qu'il y a eu un échec quelque part.

Lignes 58 à 72 : la boucle principale (le main en C). Dès qu'il y a un $X il y a des doubles quottes pour protéger des espaces. La ligne 71 contient le shift de l'option et dans chaque option je consomme les paramètres en accord avec le nombre de paramètre (par exemple les lignes 66 et 67). Ligne 69 j'ai le traitement d'une option inconnue.

1. #!/bin/bash
2. function messageOnStderr () {
3. echo "$1" 2>&1
4. }
5. function option\_i () {
6. messageOnStderr "init"
7. }
8. function option\_c () {
9. messageOnStderr "clear"
10. }
11. function option\_e () {
12. messageOnStderr "erase"
13. }
14. function option\_m () {
15. messageOnStderr "meteo of $1"
16. }
17. function option\_s () {
18. messageOnStderr "silent of $1"
19. }
20. function option\_a () {
21. messageOnStderr "analyse"
22. }
23. function option\_w () {
24. messageOnStderr "web for $1 at $2"
25. }
26. function option\_t () {
27. messageOnStderr "cheat with $1"
28. }
29. function help () {
30. {
31. echo "meteo [option]\*"
32. echo " -i"
33. echo " -c"
34. echo " -e"
35. echo " -m town"
36. echo " -s town"
37. echo " -w town day"
38. echo " -t day"
39. echo " -h"
40. } 2>&1
41. }
42. function error () {
43. messageOnStderr "error $1"
44. exit 1
45. }
46. while [ "$#" != "0" ]; do
47. case "$1" in
48. -i) option\_i;;
49. -c) option\_c;;
50. -e) option\_e;;
51. -m) option\_m "$2"; shift;;
52. -s) option\_s "$2"; shift;;
53. -a) option\_a;;
54. -w) option\_w "$2" "$3"; shift; shift;;
55. -t) option\_t "$2"; shift;;
56. -h) help;;
57. \*) error "wrong option $1";;
58. esac
59. shift 1
60. done

Vous noterez l'absence de parenthèses lors de l'invocation des fonctions, il n'en faut pas. Les plus attentifs constaterons qu'à la ligne 66 je transmets $2 et ^$3 mais dans la fonction \_w à la ligne 32 j'imprime $1 et $2. C'est normal dans le 'main' les arguments sont ordonnés d'uen façon et à l'invocation de \_w les arguments sont transmis dans l'ordre d'arrivée, ainsi $2 est le premier argument a transmettre il sera le $1 dans la fonction \_w etc…

L'intérêt de cette structure est qu'à partir de maintenant il faudra essentiellement remplir les fonctions, éventuellement ajouter des variables (après la première ligne), des comportements globaux (des fonctions après les variables globales) et des fonctions plus spécialisées selon les traitements qu'il faut réaliser (traditionnellement juste avant qu'on les invoque).

Dans le zip, le fichier intitulé meteo-initial.bsh devrait illustrer le contenu après cette étape de développement.

## Détails concernant l’initialisation

C'est le code qu'il va falloir écrire pour la fonction option\_i.

### Constatations

Dans le texte on évoque une initialisation qui semble totale, avec même l'effacement d'un cache de donnée qui semble générer des problèmes par la suite. Comme on ne sait pas comment les choses vont évoluer il va falloir prendre des précautions pour faciliter l'écriture de cette fonction.

A la lecture du sujet … Oui des fois il est 'futé' de lire l'intégralité du sujet avant de commencer quelque chose. Les paragraphes suivants traitent de l'effacement des téléchargements et des analyses. Comme l'effacement total va supposer aussi effacer les téléchargements et les analyses il serait sain de prévoir deux fonctions qu'on l'on peut invoquer indépendamment.

Par contre, comme il y a 'couplage' autant commencer par traiter les plus simple et revenir sur cette option quand j'aurais la réponse aux deux autres options. Cela va m'éviter des aller-retour dans l'écriture du code.

## Détails concernant l'effacement des téléchargements précédents

### Constatations

Il est écrit qu'il faut effacer toutes les données liées aux téléchargements et uniquement eux. Je me doute que je vais faire de 'nombreux' téléchargements et qu'il va falloir les différencier des autres types de fichiers.

### Possibilités

Il y a de nombreuses solutions :

* un dossier spécifique au téléchargement,
* des fichiers à la racine du script avec une charte de nommage qui identifieraient les téléchargement,
* etc…

J'aime bien l'idée du dossier spécialement nommé pour stocker les téléchargements. Il devient facile à supprimer (récursivement) et me permet de garder l'idée d'une charte de nommage pour le nom.

Je n'aime pas polluer l'espace où se trouve le script avec des données d'exécution, et cela pour deux raisons : la première il devient illisible de retrouver le script (c'est quand même un fichier important) et en faisant le ménage trop vite on peut le supprimer par inadvertance et la deuxième est qu'en principe le dossier où est stocké la commande n'est pas prévu pour y stocker des données.

### Choix

Pour faciliter la gestion des dossiers spéciaux (il n'y en a jamais qu'un mais en fin de projet c'est de nombreux dossiers/fichiers qu'il faut gérer et les nommer à la volée dans le code n'est pas une bonne idée car ils sont essentiels au bons fonctionnement du scripte. Je vais donc créer une variable avec un nom clair pour identifier son usage et je le fais en début de script comme ça on ne peut pas les ignorer/oublier.

Par la suite, si le scripte entre en production il faudra reloger le dossier principal pour stocker les données. Avec cette approche il suffira d'ajouter le nom d'un dossier dans les variables et le tour sera joué et cerise sur le gâteau, tout est dans les 10 premières lignes du fichier.

### Implémentation

J'ajoute donc après le shebang :

1. DLS="downloads"

La variable est en majuscule pour la différencier visuellement, au pluriel car elle va assurer un stockage de plusieurs fichiers (bon j'exagère je sais) en anglais (en franglais, soyons honnête) car en suomi personne ne va comprendre.

J'ajoute une fonction spéciale (avec un nom qui ne laisse aucun doute) pour l'effacement du dossier.

1. function clearDownloads () {
2. [ -d "$DLS" ] && rm -r "$DLS"
3. }

Vous constaterez que je prends la précaution de vérifier que la variable pointe bien sur un dossier avant de faire la suppression récursive. Notez également toutes les doubles quottes pour éviter un soucis si un espace serait dans le contenu de la variable.

Enfin il faut invoquer la fonction … facile

1. function option\_c () {
2. messageOnStderr "clear"
3. clearDownloads
4. }

## Détails concernant l'effacement des analyses précédentes

### Constatations

Ce paragraphe est identique au précédent mais il traite exclusivement des fichiers qui seraient liés au résultat d'une analyse.

### Choix

Je vais adopter donc la même démarche, un dossier référencé par un nom symbolique, une fonction pour l'effacement une invocation.

### Implémentation

J'ajoute en début de fichier (après le shebang) :

1. ANALYSES="analyses"
2. function eraseAnalyses () {
3. [ -d "$ANALYSES" ] && rm -r "$ANALYSES"
4. }

Et la fonction d'invocation …

1. function option\_e () {
2. messageOnStderr "erase"
3. eraseAnalyses
4. }

Cela semble verbeux. Certes, mais avec cette approche on devient systématique, le code est bien, écrit machinalement et sa maintenance est facilitée car structurellement identique.

## Retour sur Détails concernant l’initialisation

### Constatations

Maintenant que j'ai les deux fonctions de base je peux revenir sur cette option. Il semble qu'il y a encore une table d'association qu'il faut supprimer. Je vais adopter la même approche car je veux continuer à être systématique dans mon codage.

### Implémentation

Je vais donc créer une variable pour les mappings (associations in British) et une fonction pour effacer le dossier.

1. MAPPINGS="mappings"
2. function clearMappings () {
3. [ -d "$MAPPINGS" ] && rm -r "$MAPPINGS"
4. }

Il reste à faire l'option\_i qui va se charger de faire le nettoyage.

1. function option\_i () {
2. clearMappings
3. clearDownloads
4. eraseAnalyses
5. }

Yep c'est minimal et fonctionnel. Néanmoins, je vais avoir un souci par la suite … Les trois variables pointent sur des dossiers qui probablement devront exister pour qu'on puisse y déposer des fichiers sans être obligé de les créer régulièrement. J'aime bien avant la boucle principale invoquer une fonction spéciale nommée before (car elle est exécutée avant la boucle) qui va se charger de faire les créations nécessaires. Ainsi, j'ai les suppressions et les créations de gérée.

1. function before () {
2. [ ! -d "$DLS" ] && mkdir "$DLS"
3. [ ! -d "$ANALYSES" ] && mkdir "$ANALYSES"
4. [ ! -d "$MAPPINGS" ] && mkdir "$MAPPINGS"
5. }

Précaution indispensable, je vérifie si le nom pointe sur un dossier et s'il n'existe pas je le créé. J'invoque cette fonction avant la boucle while du traitement des options.

Dans le zip, le fichier intitulé meteo-after\_ice.bsh devrait illustrer le contenu après cette étape de développement.

## Détails concernant le téléchargement

Beaucoup d'informations dans ce paragraphe. Il va falloir trier et avancer pas à pas.

### Constatations téléchargement

Il faut télécharger une page et la stocker sur le disque si elle n'était pas présente. Facile, combien de lignes pour coder ce truc …

### Choix téléchargement

Une fonction pour faire le travail. On lui demande de faire le travail du téléchargement et du stockage et elle fera la vérification voire d'autres trucs par la suite. Mais la certitude est : après l'invocation le fichier est sur le disque où la page n'existe pas/plus. Comment s'appelle le fichier, comment on reçoit les données c'est un problème qu'on résoudra quand on aura le panorama complet.

### Constatations stockage

Comment va-t-on nommer le fichier ? Il faudra trouver une solution simple et efficace car non seulement il faut stocker mais il faudra être capable de vérifier si le fichier a déjà été stocké. Une alerte en rouge (donc faut pas l'ignorer) elle signale qu'il ne faut pas télécharger le contenu si vous l'avez déjà fait. Visiblement l'auteur du sujet est paranoïaque sur le point du téléchargement et vaudrait mieux faire bien.

### Possibilités stockage

On a déjà un dossier ($DLS) pour isoler les téléchargements des autres fichiers qui ont vocation à être stockés sur le disque mais un jour il y a aura les mappings, les analyses. Maintenant, il faut une charte de nommage pour le fichier. Cette charte doit résoudre plusieurs problèmes :

* Contenir que des caractères valides pour un fichier unix
* Le fichier doit être unique pour la page mais il doit pouvoir représenter plusieurs page différentes (faudrait savoir en quoi sont-elles différentes ?)
* Représenter l'URL utilisé avec toutes les informations : domaine, document, paramètres. C'est à cette condition que l'on pourra faire la vérification d'un téléchargement déjà effectué.

On ne peut pas utiliser l'URL directement car elle pose deux problèmes plutôt pénibles :

* Une URL contient le / qui sera réinterprété comme séparateur de dossier et ce ne sera pas pratique car il faudrait les créer à l'avance etc… que du travail inutile.
* Une URL peut contenir des caractères spéciaux (?&[] etc.) qui vont également poser des problèmes.
* On peut traduire les caractères qui posent des problèmes soit en les retirant, soit en les substituant par un séparateur soit en les translatant en séquence %20 pour un espace, %40 pour un @ etc… comme une URL encodé.
* En fait, il est aussi possible de demander à curl de faire ce travail… Mais c'est se reposer sur une implémentation pour résoudre un problème crucial. Et … curl ne fera pas la pré vérification de l'existence ou pas du fichier, il faudra donc deviner comment il a fait la traduction pour retrouver le nom potentiel et faire le test. Ce n'est pas un souci… sauf que si les développeurs décident de changer la manière de traduire les caractères farfelus vous devrez changer un scripte qui marchait. Pas top.

Bon, après avoir longuement réfléchi à cet aspect, je suis tombé sur un dernier point… Comme on va télécharger des fichiers de prévision météo, ils dépendent de la date du jour et cette information est nulle part, ni dans l'URL ni dans les paramètres logiques, elle est dans l'instant courant.

### Choix stockage

Il va donc falloir trouver une solution simple car l'URL est un objet difficile à manipuler à cause des caractères farfelus, c'est un objet clé et il faut avancer dans la résolution des problèmes. Le plus simple … on fabrique une fonction à qui on donne les données nécessaires (URL, paramètres, variantes, etc…) et qui me donne en retour le nom à utiliser en tenant compte de tout.

Déjà je prévois plusieurs cas d'usage :

* Je dois stocker pour météofrance et je dois préserver le fichier : il me faudra le nom de la ville et son code postal donc au moins 2 paramètres pour le nom du fichier, comme précisez précédemment la date du jour devra ne pas être oublié.
* Je dois stocker pour laposte et je dois préserver le fichier : il me faudra le paramètre pour le nom de fichier.
* Je dois stocker pour laposte et je ne dois pas préserver le fichier : il me faudra aucun paramètre pour le nom de fichier car je peux utiliser un nom magique.
* Vais-je avoir d'autres cas… Aucune idée mais il y aura des fichiers pour l'analyse et des fichiers pour d'autres cas.

Pourquoi préserver ou pas. Des fois, on veut télécharger et garder le contenu car le traitement sera fait à une date ultérieure. Ou on peut télécharger et traiter immédiatement le contenu et stocker le résultat du traitement ailleurs et le fichier devient inutile. Il est temporaire et jetable.

Enfin, le dossier cible sera un dernier problème : qui va en tenir compte :

* C'est l'appelant : il va fabriquer le nom de cette façon N="$DLS"/`build\_local "$town" "$zip"`. Dans ce cas la fonction a 2 paramètres et c'est l'appelant qui ajoute le dossier.
* C'est la fonction : on aura le nom N=`build\_local "$DLS" "$town" "$zip"`. Dans ce cas la fonction a 3 paramètres et c'est elle qui ajoute le dossier.
* Si on a X fonctions, elles sont dédiées, donc elles peuvent ajouter le dossier implicite, voir même adapter l'extension au cas traité.
* En fait, il y a une dernière solution… On développe une fonction unique qui reçoit un paramètre supplémentaire 'usage' qui permet de calculer la composition et c'est usage qui permet de savoir si on utilise DLS ou MAPPINGS comme dossier et on peut inventer plein de nouvelles possibilités. C'est chouette car universel mais ça impose de donner un paramètre qui est lié à l'application donc on ne peut pas réutiliser facilement ou alors c'est tellement évident que cette approche est bien.

### Implémentation stockage

J'avoue que si je dois faire très intense je vais aller dans la direction du paramètre supplémentaire qui va donner l'usage. C'est tellement universel, mais couteux à développer surtout que je ne connais pas encore toutes les variantes (et un œil acéré distinguera que cette fonction ne ferait que rassembler les X fonctions si on les développe individuellement). De plus, on est en TP et on peut faire simple. Dans ce cas, je vais opter pour une fonction dédiée pour chaque type de nom de fichier (météo, mappings, analyse, etc.), comme elles sont dédiées elles peuvent aussi prendre en charge le nom du dossier, l'extension etc… Par contre, je vais faire 'court' je ne contrôle pas les données et je considère que le développeur sait développer correctement. Pour la date, elle est générée en interne avec non plus le / comme séparateur mais un \_. Cela permet d'avoir un nom de fichier conforme à Unix et je garde le – comme séparateur dans le nom. Il assure la distinction nom de la ville – code postal.

1. function buildMeteoFranceFilename () {
2. day=`date "+%m\_%d\_%Y"`
3. echo "$DLS/$1-$2-$day.html"
4. }

et pour la fonction associée au mapping

1. function buildMappingFilename () {
2. echo "$MAPPINGS/local.json"
3. }

### Constatations USER-AGENT

Un deuxième paragraphe impose d'utiliser un USER-AGENT pour se faire passer pour un navigateur.

### Choix USER-AGENT

Pas de discussion sur les diverses possibilités, vu le contenu d'un USER-AGENT il va falloir le stocker dans une variable qu'on initialise en début de scripte et qu'on utilise lors du curl.

### Constatations URL

On a une première URL de météo France :

https://meteofrance.com/previsions-meteo-france/<<VILLE>>/<<POSTAL>>

Par la suite dans le sujet on découvre une deuxième URL

https://datanova.legroupe.laposte.fr/api/records/1.0/search/?dataset=laposte\_hexasmal&q=<<PARAM>>&lang=fr&rows=10&facet=nom\_de\_la\_commune&facet=code\_postal

Elles sont moches mais les deux sont paramétriques. Ceux sont des modèles avec des valeurs qu'il faut modifier en fonction de paramètres. J'ai mis le nom des paramètres en majuscule.

### Possibilités URL

Plusieurs solutions selon la structure de l'URL

Il n'y aurait pas de paramètre elle pourrait être en dur dans le code, mais ce n'est pas souhaitable car s'il faut la changer il faut partir à sa recherche.

Il y a des paramètres, il faut probablement combiner deux technologies, une qui fournit l'URL de base et une qui l'adapte aux paramètres.

La fourniture, il y a au moins deux solutions :

* L'URL pourrait être dans une variable initialisée en début de scripte et utilisée lors de l'adaptation. Approche raisonnable quand l'URL est unique et n'a surtout pas vocation à devenir un 'pool' d'URL dans lequel il faut piocher selon une logique.
* L'URL pourrait être dans un fichier, extraite du fichier et adapté. Complètement générique, puisque vous avez une collection et vous y puisez librement. Cette approche pose des questions supplémentaires, quand créé-t-on ce fichier, comment le modifie-t-on et enfin quand le supprime-t-on. Rien de majeur mais encore des questions et … faudrait avancer. Enfin, pour une URL c'est probablement contreproductif sauf si on sait déjà qu'on va passer à X (et on ne le sait pas – pour ce sujet).

L'adaptation il y a à nouveau deux solutions :

* L'URL est 'simple', grossièrement il suffit de changer la partie terminale en bidouillant et en collant les paramètres. Cette solution est simple, très simple, mais elle ne permet pas une évolution vers des manipulations plus complexes.
* L'URL est plus complexe, on y inscrit des emplacements que l'on remplace lors de l'adaptation. C'est moins intuitif mais en définitive beaucoup plus générique.

Clairement quel que soit le choix, il faudra une fonction pour faire ces bidouilles et cacher la misère.

Une fonction, sous unix, peut poser de nouvelles questions comme : où sont les données comme l'URL, les nom les paramètres, etc. et où va-t-on fournir le résultat ? Selon le choix c'est plus ou moins pratique, propre, efficace (et il n'y a pas de solution unique, selon la fonction un choix est meilleur que l'autre ce qui veut dire que toute les fonctions doivent être implémentée en se posant des questions). Mais quels sont les choix :

* On peut fournir des informations via l'entrée standard. Cela signifie que l'on utilise une commande pour fabriquer un contenu que l'on enchaine à la fonction en utilisant le pipe (|)
* On peut fournir des informations via les arguments. Cela signifie qu'il faut les traiter via des $1, $2, etc.
* Bien sûr, on peut combiner l'entré standard et les paramètres… Pourquoi faire simple.
* Le résultat peut être écrit sur la sortie standard, cela suppose alors que le résultat sera traité par une autre commande enchaînée via un pipe (|) ou stocké dans une variable via la redirection `` ou $().
* Le résultat peut être écrit dans un fichier, cela suppose que l'utilisateur sache comment ce fichier s'appelle et où il est (c'est la stratégie utilisée pour le téléchargement)

Ce n'est pas aussi simple, car cette fonction va devoir abstraire l'application des paramètres et malheureusement :

* Si les bidouilles (bricolage par concaténations) sont illisibles, elles sont faciles.
* Une fonction ultra générique va couter cher à développer car il faut tenir des paramètres sous la forme de paires clé/valeur, de l'URL initial etc… Bref non seulement un passage de paramètres long et verbeux mais l'exploitation dans la fonction n'est pas triviale
* Développer deux fonctions (une pour météofrance et l'autre pour laposte) ce n'est pas top mais nettement plus simple. Pourquoi, on connait le nombre de paramètre (1 ou 2) et on peut se contenter de la position pour les identifier. Pour 1 à 2 c'est largement suffisant, 3…4 je pense qu'il faut abandonner cette idée.

Pour cette (ou ces) fonction(s) une question reste encore en suspens où trouvera-t-elle l'URL.

### Choix URL

Pour l'URL vers laposte sans aucun doute, je choisis la variable globale en début de fichier. Cette URL est unique et si elle doit être modifiée elle est facile à modifier.

Pour l'URL vers météo france, j'aurais tendance à perdre 1s pour hésiter mais la raison du plus simple l'emporte (ce n'est pas toujours le cas). Je ne vois pas (et c'est moi qui ai inventé le sujet-je triche car depuis j'ai trouvé pourquoi) sous quelles injonctions on devrait utiliser un pool d'URL et donc recommander l'usage d'un fichier.

Concernant l'adaptation de l'URL avec le paramétrage demandé… Sincèrement, coller des contenus je sais faire mais la limite du 'en fin d'URL sinon c'est moche' me pousse vers la solution substitution. La fonction générique me tente mais sa rédaction est contreproductive car elle va couter cher en temps pour une fonctionnalité qui n'est pas indispensable (on pourra très bien la développer plus tard si besoin est). Je vais donc me diriger vers deux fonctions une pour meteo et une pour laposte. La structure sera la même pour les deux : l'URL arrive via l'entrée standard, les paramètres dans les arguments selon un ordre déterminé et le résultat est sur la sortie standard. L'entrée standard car cela permet d'avoir des filtres pour sélectionner la bonne URL et la sortie cela permet de coller le résultat dans une variable via les `…`

### Implémentation URL

Il va me falloir 2 variables initialisées en tête de scripte.

1. URL\_MAPPING="https://datanova.legroupe.laposte.fr/api/records/1.0/search/?dataset=laposte\_hexasmal&q=PARAM&lang=fr&rows=3&facet=nom\_de\_la\_commune&facet=code\_postal"
2. URL\_METEOFRANCE="https:// meteofrance.com/previsions-meteo-france/TOWN/ZIP"

J'ai fait simple pour marquer les paramètres : un nom en majuscule. Les majuscules ne sont pas fréquentes dans une URL normale donc pas trop de risque de me 'tromper' dans la substitution. Maintenant si le risque existe il suffit d'encadrer les paramètres avec une ponctuation quelconque, alors j'aime bien utiliser le % ou le !.

1. function buildMeteoFranceURL () {
2. sed -e "s/TOWN/$1/" -e "s/ZIP/$2/"
3. }
4. function buildMappingURL () {
5. param=""
6. param=${param:-$1}
7. param=${param:-$2}
8. sed -e "s/PARAM/$param/"
9. }

Oui elles sont simples, et alors ? Elles cachent la misère qui n'était pas aussi dramatique (mais aurait pu le devenir avec une fonction générique. La buildMappingURL s'autorise à recevoir deux paramètres la ville ou le code postal et elle va prendre la première non vide. J'utilise les substitutions de chaines du bash qui permet de simplifier les if/then/else en 1 ligne magique.

Pour l'utilisation, il suffira de faire

url=`echo "$URL\_METEOFRANCE" | buildMeteoFranceURL "$TOWN" "$ZIP"`

Plus tard, le echo pourra être un sed qui sélectionne une ligne spécifique dans un fichier ou …

### Constatations paramètre à option\_m

La fonction a un paramètre qui peut être soit un nom de ville soit un code postal. Il va falloir savoir si on a un texte ou un nombre.

### Possibilités paramètre à option\_m

Le décryptage est très simple, soit :

* On fabrique l'expression régulière qui gouverne un des types de données (ici le nombre) et un if avec l'opérateur de matching du bash va permettre de vérifier facilement. Avec cette approche on se repose sur son cerveau qui n'est pas infaillible.
* On peut essayer d'appliquer une fonction si elle marche c'est bon, si elle échoue c'est bon aussi (on connait la bonne décision). Avec cette approche on se repose sur une fonction qui marche/échoue … Si le développeur change l'implémentation l'échec peut devenir un succès.

### Choix paramètre à option\_m

Etrangement j'aime bien l'idée du succès/échec, surtout sur le nombre. Je pense que le développeur ne va plus changer le comportement et j'aurais un résultat pérenne dans le temps. Si ce n'était pas un nombre/texte je prendrais la première approche. Reste à trouver un comportement qui a un succès sur un nombre et échoue dans les autre cas … l'addition, les calculs, les comparaisons, …

### Implémentation paramètre à option\_m

Je fais donc souvent :

1. if expr "$2" + "0" 2>/dev/null >/dev/null; then
2. # un nombre
3. else
4. # un texte
5. fi

Oui c'est redoutablement simple, pourquoi faire autrement ? Il reste à inscrire le code spécifique à la place des commentaires.

### Constatations laposte

Pas facile de comprendre où et comment utiliser cette URL. Mais on peut résumer par :

* L'utilisateur ne donne qu'une donnée, il faut trouver l'autre
* Il ne faut pas télécharger la ville ou le code postal depuis laposte si on l'a déjà. Oups on demande Colmar on trouve 68000 mais si on demande 68000 on ne télécharge pas car on la connait même si on avait demandé avec la ville. Cela veut dire qu'on ne vérifie pas si un fichier existe avec le paramètre transmit mais on vérifie si on connait déjà une association.
* Donc, il semble donc que ce n'est pas le téléchargement qui est important mais la présence de l'association et si elle est absente on télécharge.
* Ce qui va poser d'autres questions comme qu'elle structure de fichier d'association et comment y accède-t-on ?

### Choix laposte

Je vais donc définir une fonction qui se charge de me trouver le mapping qui m'intéresse. Je prévoie de lui transmettre deux paramètres : le premier sera le code postal et le deuxième le nom de la ville. Ainsi, je renseigne celui que je connais et si un jour je veux compliquer je peux. Le résultat sera sur la sortie standard sous la forme d'une paire avec un séparateur. Le : est un bon choix car il ne devrait pas trainer dans les codes postaux ni le nom des villes. Inutile de donner le nom du fichier à utiliser, il sera défini globalement (comme les URL).

Au passage, cette fonction va télécharger un nouveau mapping et en faire l'insertion s'il était manquant. Je vais donc utiliser deux nouvelles fonctions : l'une pour télécharger et une autre pour trouver le contenu dans le json. La durée de stockage du fichier téléchargé n'est pas précisée mais une fois que j'ai extrait l'information je peux probablement considérer qu'il n'est plus utile (je me protège des téléchargements multiples via le fichier de mapping). On pourrait donc le réutiliser pour chaque téléchargement.

Comme cela ne suffit pas, je définis la structure du fichier de mapping. C'est facile y a deux champs ville et code postal faut les avoir un par ligne et faut un séparateur. Unix a l'habitude du : car il n'est pas autorisé dans les contenus, comme ici, autant faire comme Unix. Au cas où, j'ajoute deux fonctions minimaliste: une pour requêter le fichier et une pour ajouter dans le fichier. Ainsi, les 95% du boulot de l'encodage de l'association sera réalisé et modifiable facilement.

### Implémentation laposte

La variable globale (oui elle va dans le dossier des mappings) :

KNOWN\_MAPPINGS="$MAPPINGS/mappings.txt"

Les deux fonctions de manipulation du fichier des mappings :

1. function selectFromKnownMappings () {
2. zip=${1:-[^:]\*}
3. town=${2:-.\*}
4. grep -i "^$zip:$town\$" < "$KNOWN\_MAPPINGS"
5. }
6. function insertInKnownMappings () {
7. [ ! -z "$3" ] && [ -z `selectFromKnownMappings "$1" "$2"` ] && echo "$3" >> "$KNOWN\_MAPPINGS"
8. }

L'insert est 'standard' car il insère que si le mapping existe actuellement ($3) et n'existait pas auparavant. Le select est tordu mais, en fait, il remplace le paramètre inconnu donc vide par une expression régulière qui permet de spécifier un champ a une valeur quelconque.

La fonction getMapping maintenant. J'extrait un mapping. Si j'ai trouvé, le if est ignoré et je retourne le mapping. Je n'ai encore rien. Je calcule le nom du fichier cible du téléchargement (unique car je traite de suite derrière) Je télécharge la page. J'extraite les données du json, j'inscris le mapping dans le fichier et je retourne le résultat.

1. function getMapping () {
2. mapping=`selectFromKnownMappings "$1" "$2"`
3. if [ -z "$mapping" ]; then
4. target="`buildMappingFilename`
5. downloadMapping "$1" "$2" "$target"
6. mapping=`extractMappingFromJson "$1" "$2" "$target"`
7. insertInKnownMappings "$1" "$2" "$mapping"
8. fi
9. echo "$mapping"
10. }

Il n'y aura pas de paragraphes pour discuter des nombreuses façons de faire un analyseur de json. Sniff. En fait, ici le job de json est simple il doit trouver quelque chose dans le fichier et retourner une chaine zip:town pour qu'on puisse l'insérer dans le fichier. Le pire c'est que les données sont dans une zone qui ressemble à "nom\_de\_la\_commune": "<<town>>", <<bruit>>, "code\_postal": "66666",. J'ai donc bidouillé deux expressions régulières pour matcher le buffer \(…\) avec les contenus que j'ai ou que je recherche et une troisième pour consommer le bruit [,]\*. Les lignes 2 et 3 sont magiques Si les champs sont vides j'utilise une expression régulière qui aspire un contenu adapté. Avec ces expressions régulières, j'ai toujours un contenu dans le buffer et des fois j'impose un contenu (c ritère de la recherche) des fois je prends n'importe quoi (résultat de la recherche). J'avoue, je me suis fait plaisir et il y a d'autres solutions moins rusées.

1. function extractMappingFromJson () {
2. zip=${1:-[^\"]\*}
3. town=${2:-[^\"]\*}
4. sed -n 's!.\*"nom\_de\_la\_commune": "\('"$town"'\)",[^,]\*, "code\_postal": "\('"$zip"'\)",.\*!\2:\1!pI' < "$3" | tr [A-Z] [a-z]
5. }

OMG c'est quoi ce truc. J'avoue elle est rusée. Pour faire simple le sed va chercher la ligne en matchant un code postal et une ville et réimprime le résultat séparé par un : et passe tout en minuscule. C'est pour cela qu'il y a (illisible mais bien présent) un buffer \( … \) qui est créé autour de $zip et $town. Maintenant la construction des variables. Deux cas : la variable a un contenu je l'utilise car il sera à cet emplacement dans le fichier et grâce au buffer je vais ressortir le contenu que je connaissais mais c'est plus simple comme ça. La variable n'a pas de contenu, je remplace par un aspirateur de texte jusqu'à une double quotte [^"]\* qui va aspirer le contenu réel et inconnu tout en le stockant dans le buffer. Oui, je connais mes classiques.

### Choix option\_m

Je crois que maintenant cette fonction va devenir évidente… même s'il y a encore des trous

* Elle reçoit un paramètre
* On recherche le bon mapping selon le type (entier, texte)
* Si le mapping existe on télécharge sur météoFrance

### Implémentation option\_m

1. town=""
2. zip=""
3. param="$1"
4. if expr "$param" + "0" 2>/dev/null >/dev/null; then
5. zip="$param"
6. else
7. town="$param"
8. fi
9. mapping=`getMapping "$zip" "$town"`
10. if [ ! -z "$mapping" ]; then
11. zip=`echo $mapping | cut -d: -f1`
12. town=`echo $mapping | cut -d: -f2`
13. downloadFromMeteoFrance "$town" "$zip"
14. fi

Remarque : on aurait pu cacher le if numérique/test dans getMapping cela simplifie encore plus cette option\_m. Mais, … on ne pourra plus faire un getMapping avec par exemple un 'début' de code postal ou de nom de ville. Ne soyons pas puriste c'est déjà bien.

### Constatations téléchargements

Si je fais le compte il me reste à gérer les téléchargements.

* J'ai déjà planifié deux fonctions l'une pour laposte et l'autre pour météofrance.
* Les deux fonctions reçoivent les paramètres à utiliser dans les arguments
* Elles sont chargées de déposer la page sur le disque
* En testant l'unicité pour météofrance, plus de test pour laposte car j'ai anticipé avec le test dans le mapping.
* J'ai deux fonctions build\_\* pour construire l'URL selon le site ciblé.

### Implémentations téléchargements

Je peux fabriquer cette fonction pour météofrance

1. function downloadFromMeteoFrance () {
2. FILENAME=`buildMeteoFranceFilename "$1" "$2"`
3. if [ ! -f "$FILENAME" ]; then
4. URL=`echo "$URL\_METEOFRANCE" | buildMeteoFranceURL "$1" "$2"`
5. curl -A "$USER\_AGENT" "$URL" > "$FILENAME"
6. fi
7. }

En fait … elle est ridiculement simple (maintenant).

Et celle-là pour laposte

1. function downloadMapping () {
2. URL=`echo "$URL\_MAPPING" | buildMappingURL "$1" "$2"`
3. curl -A "$USER\_AGENT" "$URL" > "$3"
4. }

### Constatations

Je n'en ai plus pour cette option\_m. C'était long mais c'est le premier cœur du script donc pas illogique.

En fait, y a une dernière opération qu'il ne faut pas oublier de faire. Quand vous n'avez aucune association il faut créer un fichier vide pour éviter que le scripte ne fasse des erreurs. Dans la function before, il suffit d'ajouter :

[ ! -f "$KNOWN\_MAPPINGS" ] && cat < /dev/null > "$KNOWN\_MAPPINGS"

Dans le zip, le fichier intitulé meteo-after\_m.bsh devrait illustrer le contenu après cette étape de développement.

## Détails concernant la simulation du téléchargement

### Constatations

C'est une fonctionnalité qui ne devrait pas impacter fortement le code précédent. En effet, vu les explications, il faut changer l'URL pour la dérouter sur un autre site. C'est globalement autour du build\_meteofrance qu'il y a la modification. Comme il était prévu de transformer une URL lue sur l'entrée standard c'est donc le code qui génère le contenu qu'il faut modifier. On le trouve dans download\_meteofrance. C'est un echo dans le code actuel.

### Possibilités

Il va falloir injecter le choix de la source réelle dans le download\_meteofrance. Un paramètre de plus. Puis il va falloir décider de la bonne URL. On peut continuer à créer des variables avec de URL ciblant un meteofrance réel, faux etc… Il reste plus qu'à choisir. Un case ou plus futé une fonction get\_url\_ meteofrance qui va résoudre ce problème.

### Choix

Bien sûr je vais opter pour la fonction getUrlMeteoFrance avec le paramètre pour choisir la meilleure dans les constantes prédéfinies. Au passage, je vais étendre avec une ruse vraiment utile. Héberger un site pour redistribuer les fichiers c'est plutôt long à mettre en œuvre (chez vous) par contre il y a une troisième solution, les déposer sur le pc et au lieu de foncer sur le réseau pour chercher le contenu on peut le chercher sur le disque… Ce n'est qu'une question d'URL.

Le plus simple sera de recopier le code d'option\_m dans une fonction spéciale (processDownloadFromMeteoFrance). Elle reçoit un paramètre supplémentaire ($2 pour éviter une renumérotation hasardeuse) qui est le mode qui sera transmis à la fonction getUrlMeteoFrance toujours en fin de liste d'arguments donc le $3 pour elle et on remplace le echo "$URL\_METEOFRANCE" par l'invocation de la fonction qu'on vient de créer avec le paramètre de mode.

Les fonctions option\_m et option\_s deviennent élémentaires car il suffit d'invoquer la fonction processDownloadFromMeteoFrance avec le premier argument qui est le paramètre initial de la fonction et un deuxième argument qui est le mode : real ou safe.

### Implémentation

1. function getUrlMeteoFrance () {
2. case "$1" in
3. real) echo "$URL\_REAL\_METEOFRANCE";;
4. safe) echo "$URL\_SAFE\_METEOFRANCE";;
5. fake) echo "$URL\_FAKE\_METEOFRANCE";;
6. esac
7. }

La variable URL\_REAL\_METEOFRANCE est l'ancienne URL, la SAFE pointe sur mon faux site web don c'est une url type http://localhost/… et la dernière pointe sur le disque local donc une file:///home/<<user>>/…

Chouette ? non ! en plus curl sait gérer des file:/// donc rien à faire de plus c'est réellement transparent. Il faut juste dans l'URL coder le chemin en absolue vers le fichier.

Voilà la nouvelle variante du code.

1. function processFromMeteoFrance () {
2. param="$2"
3. mode="$1"
4. town=""
5. zip=""
6. if expr "$param" + "0" 2>/dev/null >/dev/null; then
7. zip="$param"
8. else
9. town="$param"
10. fi
11. mapping=`getMapping "$zip" "$town"`
12. if [ ! -z "$mapping" ]; then
13. zip=`echo $mapping | cut -d: -f1`
14. town=`echo $mapping | cut -d: -f2`
15. downloadFromMeteoFrance "$town" "$zip" "$mode"
16. fi
17. }
18. function option\_m () {
19. messageOnStderr "meteo of $1"
20. processFromMeteoFrance "real" "$1"
21. }
22. function option\_s () {
23. messageOnStderr "safe of $1"
24. processFromMeteoFrance "safe" "$1"
25. }

Avec cette approche :

Un premier truc 'chouette' on peut inventer l'option –f qui va utiliser une copie sur le disque. Elle aurait cette structure :

1. function option\_f () {
2. messageOnStderr "fake of $1"
3. processFromMeteoFrance "fake" "$1"
4. }

Un deuxième truc, je reviens sur le fichier d'URL pour météofrance. J'avais retiré cette idée car elle ne semblait pas correspondre à mon besoin et me coutait trop cher en développement. Maintenant, je pourrais reconsidérer la situation car j'ai au moins 3 sources d'URL en les stockant dans le fichier avec un index je pourrais sélectionner depuis le fichier et non pas depuis un ensemble de variables.

Dans le zip, le fichier intitulé meteo-after\_ms.bsh devrait illustrer le contenu après cette étape de développement.

## Détails concernant l’analyse

### Constatations générales

Les premières informations m'expliquent d'une façon hautement complexe que je dois traiter des fichiers téléchargés s'ils existent et ne pas me poser de questions si j'ai déjà fait ou pas.

### Implémentation générales

Donc je vais devoir avoir une fonction qui va boucler sur les fichiers et une fonction qui va se charger d'extraire les données de la page et de les stocker dans un format adapté. Pour faire simple, je vais écrire :

1. function option\_a () {
2. messageOnStderr "analyse"
3. for file in `echo $DLS/\*.htm`; do
4. analyseOneFile "$file"
5. done
6. }

La fonction analyseOneFile sera chargée de faire l'extraction de toutes les données demandées (voire plus) et de les stocker quelque part sur le disque.

### Constatations date de téléchargement

Le paragraphe qui suit est sibyllin. Mais si je décrypte, prendre la date du jour de l'analyse ne semble pas la bonne solution il faut celle du téléchargement. Chouette elle est dans le nom du fichier il y a qu'à l'exploiter.

### Constatations données (premières)

Puis, il y a une 'longue' liste de données qu'il faut extraire d'une page avec des indications plus ou moins complexes voire incompréhensibles (car on ne connait pas le contenu réel – il faudra ouvrir la page sur un notepad++ ou autre pour pouvoir lire et comprendre la structure. Mais, la lecture me fait savoir que je vais avoir plusieurs étapes à réaliser pour chaque fichier. Globalement, il faut trouver un lot de lignes qui contiennent les données recherchées, de ce lot en garder la ligne intéressante ou les lignes intéressantes puis dans chaque ligne retirer le bruit ou sélectionner le contenu cherché.

Avant de sélectionner une approche, il faut aussi s'interroger sur comment stocker ce que l'on extrait de la page. Nul part dans le sujet il est fait état de la structure du fichier issu de l'analyse et selon le choix cela pourrait être plus ou moins simple par la suite.

### Possibilités logique de stockage des données extraites

On sait que les données issues de l'analyse seront stockées dans des fichiers dans le dossier $ANALYSES. Ce qu'il va nous rester à décider c'est la politique du stockage :

1. Un dossier par ville, dans ce dossier, un dossier par jour, dans ce dossier un fichier par données.
2. Un dossier par ville, dans ce dossier un fichier par jour qui contient toutes les données.
3. Un fichier par couple ville et jour qui contient toutes les données pour une ville et un jour.
4. Un fichier par ville qui contient toutes les données pour tous les jours.
5. On peut permuter ville et jour si l'on considère que le jour est plus important que la ville.

La 1 va simplifier le script car une fois que l'on a calculé le dossier cible il suffit d'y ajouter des fichiers pour chaque contenu, mais ça fait quand même beaucoup de micro fichiers.

La 2 (2' si on permute ville et jour), moins de micro fichiers que pour la 1. C'est sympa, les données utiles (ville et jour) sont dans un fichier. On veut un couple on prend le fichier, on veut ajouter l'analyse du jour on ajoute un fichier. Plutôt élégant. En plus c'est top si on a besoin de fabriquer des pages pour une ville, les fichiers dans le dossier permettent un accès simple. Si on veut plutôt avoir toutes les villes pour un jour donnée c'est 2'. Si on veut les deux alors il faut la 3

La 3, carrément plus de micro fichiers, mais si on fait tourner le script pendant 1 an sur 10 villes… 3560 fichiers. Ce n'est pas grave mais j'ai un doute sur le long terme.

La 4 et 4', je ne la sens pas car ajouter des données, en fin de fichier mais comment on va séparer les données de J-1 de celles de J. J'ai l'intuition que je vais avoir des problèmes.

Quoiqu'il arrive faudra une charte de nommage pour les dossiers (s'ils existent) et pour les fichiers et une définition des contenus comment différencier chaque données extraites d'une page.

Si on opte pour un fichier qui agrège toutes les données, il y a trois possibilités :

* on écrit chaque donnée sur une ligne selon un ordre implicite (1er le nom, 18 ème la température du 3ème jour, etc…). Solution légère si le nombre de donnée est raisonnable et la valeur courte (si la valeur devient un 'vent latérale de 10km/h avec rafales à 30Km/h' ce n'est plus utilisable). Quand on a la donnée on peut l'exploiter immédiatement.
* on fabrique une immense chaine séparée par un séparateur avec les contenus ordonnés. Solution légère si le nombre de donnée est très réduit (3 à 5) et la valeur est courte comme un nombre ou une abréviation. Il faut découper la chaine pour trouver la donnée.
* on stocke sous la forme d'une ligne par donnée, une paire clé/valeur et un séparateur. La lisibilité est top, l'extraction marche en deux temps : on trouve la ligne (sed/grep) on retire la clé c'est pas le plus difficile. Peut-être laborieux s'il y besoin d'extraire de très nombreuses données mais … on ne va jamais les mélanger contrairement aux deux précédentes.

### Choix logique de stockage des données extraites

J'avoue que ce n'était pas mon premier choix lors de la réalisation du TP mais à la rédaction de ce document je privilégie la solution avec un dossier pour la ville avec le nom de la ville et un fichier par jour avec le nom de la ville, le code postal et la date (bref comme dans le téléchargement).

Clairement, stocker des données en se reposant sur l'ordonnancement du fichier html avec aucune garanti que les données figurent dans les pages et toujours dans le même ordre (il faut donc combler les trous) j'aime pas du tout car je sais que la solution va craquer. Je vais donc privilégier une solution, à la Unix, un fichier de paires clé/valeur et un processus d'extraction très dirigé.

Une question qui est apparu lors de la confection des extracteurs… Je vais extraire les mêmes 'types' de données pour plusieurs jours ou pour plusieurs instants dans un jour, par exemple l'état du ciel le matin, l'après-midi, le soir et la nuit ou pour J, J1, J2 etc… J'ai l'intuition que créer une clé qui combine tout ne sera pas idéale, deux clés sera plus pratique. Donc le fichier aura plutôt une structure en triplet : [Instant, Grandeur, Valeur].

Je sens, mais je ne pourrais pas prouver, que disposer de la ville du code postal et de la date à l'intérieur du fichier va me simplifier les extractions par la suite (source unique, le contenu du fichier).

Par contre, pour cacher toute cette misère, il me faudra de nombreuses fonctions.

* une fonction pour construire le nom du fichier, probablement comparable à celle du téléchargement, à la différence que la date est fournie par l'utilisateur de la fonction.
* des fonctions d'extractions de contenu(s) qui vont automatiquement insérer les paires clé/valeur dans le fichier (il suffira de faire dans chaque fonction un >> "$analyse" si analyse contient le nom du fichier.
* des fonctions d'extractions de bloc qui vont piocher le bon bloc dans la page pour faciliter les traitements pour les fonctions précédentes.
* etc…

Ce qui va reposer la question où et comment trouver les données pertinentes.

### Implémentation logique de stockage des données extraites

A la louche on va finir avec ces éléments d'architectures :

1. function buildAnalyseFilename () {
2. mkdir –p "$ANALYSES/$1-$2"
3. echo "$ANALYSES/$1-$2/$1-$2-$3.txt"
4. }
5. function analyseOneFile () {
6. input="$1"
7. town=`expr "/$input" : '.\*/\(.\*\)-.\*-.\*\..\*'`
8. zip=`expr "/$input" : '.\*/.\*-\(.\*\)-.\*\..\*'`
9. day=`expr "/$input" : '.\*/.\*-.\*-\(.\*\)\..\*'`
10. analyse=`buildAnalyseFilename "$town" "$zip" "$day"`
11. cat < /dev/null > "$analyse"
12. echo "::TOWN:$town" >> "$analyse"
13. echo "::ZIP:$zip" >> "$analyse"
14. echo "::DAY:$day" >> "$analyse"
15. }

La fonction buildAnalyseFilename va me créer le nom du fichier selon mon choix (on pourrait changer facilement). La fonction analyseOneFile va traiter un seul fichier dont le nom est dans le premier argument. On y extrait du nom du fichier les 3 informations clés (ville, code postal, jour) ce qui permet de créer le fichier analyse à la ligne 11 et de vider définitivement son contenu. Les lignes 13 à 15 insère avec les triplés initiaux. Le premier champ est vide car il ne dépend pas d'un instant particulier.

### Constatations générale données (dans le fichier)

Cette partie n'est pas traité dans le sujet car elle est liée à la structure de la page html…

* Il faut télécharger une page
* Faire l'analyse visuelle du contenu avec un outil type éditeur de texte
* Trouver ce qui est essentielle pour extraire du contenu. Il faut retrouver dans quelles structures html se trouve le contenu qui est visualisé dans la page du navigateur. Dans des pages on peut avoir des répliquas qui cachent la bonne place.
* Inventer le filtre (et le tester individuellement) qui est basé sur la structure html.
* Tester individuellement chaque filtre sur un fichier de test (inutile de lancer le script à chaque fois)
* Collecter les algorithmes dans le script
* Quand les algorithmes sont déterminés, quelques dry run pour valider, basculer sur une autre page etc…

Le filtre le plus générique, est de penser à quatre couteaux qui vont passer dans le fichier.

* Les deux premiers couteaux (horizontaux) vont délimiter le bloc de lignes (l'un au-dessus du bloc, l'autre en dessous du bloc). Des fois, il suffit de dire la ligne, voire la ligne et X lignes en plus, voire de la ligne X1 à X2. S'il y a des lignes en trop, mais c'est simple à écrire, alors pas grave, on peut passer derrière pour ôter le superflu. Ces couteaux sont souvent basés sur les balises h1, h3, ul, li, div, ou des classes vraiment explicites, etc… C'est très empirique.
* Les deux autres couteux (verticaux) vont extraire le contenu noyé dans le formatage html. Ils passent à gauche du début et à droite de la fin. A nouveaux, ils vont s'appuyer sur des caractères spécifiques, les <, >, ", ' qui délimitent l'html.
* Chaque extraction est probablement indépendante donc ne pas trop chercher à généraliser mais plutôt à adopter une approche systématique.

### Constatations absences données (dans le fichier)

Une première description de ce qui est constaté dans un fichier type et la première constatation c'est qu'il n'y pas/plus de données (en 2022-2023).

Je vais sur le site et je sauvegarde la page depuis le navigateur et … un bloc nommé atmograme vient d'être rempli avec des données. Ce n'est carrément pas top, le TP est HS car ils sont passés sur un modèle dynamique de page. Le contenu est chargé à l'aide de requêtes ajax.

Mais …. une belle solution est de charger la page, cliquer sur la bonne entrée du menu, de sauvegarder la page sur le disque et d'utiliser soit un site web interne pour avoir la page pour continuer le script, soit de stocker dans le fameux dossier qui pourra servir de sources fausses. La partie téléchargement devra être revue ☹ mais pas pour le TP. Il est 'important' de cliquer sur la bonne entrée du menu car cela permet de charger le contenu dynamique qui se substitue au précédent, ainsi par défaut on a la prévision 2 jours et si on va sur 7j ou 15j ou … on a les autres prévisions (mais plus la prévision 2j). Donc si on veut faire un traitement faut être rigoureux.

### Constatations formatage données (dans le fichier)

Maintenant qu'on a une page avec des données 'réelles' on peut les examiner et en déduire des conditions. Déjà, il est où le contenu : ligne 855 à la ligne 942 on a le bloc cherché. Pfff 100 lignes, et en plus elles ne sont pas faciles à exploiter car il y a souvent de nombreuses données sur une seule ligne et Unix déteste ces lignes.

On peut déjà récupérer la partie utile du fichier, une centaine de lignes contre 2300 et après on va consulter ce contenu après l'avoir reformaté. La sélection est très simple, un block qui a comme id atmogramme est évidemment le block qui nous intéresse il peut être à la ligne 5 ou 2200 (la position numérique est sans importance), c'est le bloc nominatif qu'il faut extraire. La taille, c'est plus complexe à déterminer car selon le contenu il pourrait y avoir des lignes en plus ou moins, mais en repliant le block atmogramme, dans l'éditeur, je vois apparaitre un block qui a comme id bulletin\_tendance. Clairement c'est entre ces deux que se trouve le contenu recherché. Pour garder le contenu un sed fera le job.

sed -n '/id="atmogramme"/,/id="block-mfbulletintendance"/p'

Pour reformater le fichier, selon l'outil utilisé pour générer la page, on peut avoir besoin de faire des opérations différentes, mais à la base c'est

* rien
* insère un saut de ligne avant l'ouverture d'une balise ouvrante
* insère un saut de ligne avant l'ouverture d'une balise fermante
* insère un saut de ligne avant l'ouverture d'une balise
* insère un saut de ligne après la fin de l'ouverture d'une balise ouvrante
* insère un saut de ligne après la fin de l'ouverture d'une balise fermante
* insère un saut de ligne après la fin de l'ouverture d'une balise
* insère un saut de ligne avant et après les chevrons

Plus on insère des sauts de ligne, plus le contenu est isolé sur la ligne et les extractions suivantes deviennent fastidieuse car il faut prendre un bloc de plusieurs lignes car la première marque l'intérêt, la ou les suivantes le contenu et la dernière marque une fin. Puis il faut prendre le contenu et supprimer des textes etc. etc. Distinguer l'ouverture d'une balise ouvrante d'une balise fermante n'est pas trivial (il faut écrire une expression régulière qui exclue le / sans consommer le caractère).

Avec le bon reformatage, le fichier fait maintenant plus que 250 lignes mais elles sont plus facilement analysables.

### Implémentation formatage données (dans le fichier)

Dans les pages de météofrance il semble qu'une insertion avant l'ouverture des balises ouvrantes (donc le < pour l'ouverture uniquement). La commande est complexe mais elle vaut le coup :

sed 's/<\([^/]\)/\n<\1/g'

Je demande d'exclure le / de la substitution mais comme j'ai lu le premier caractère de la balise je dois le réinsérer en sortie. Le g permet de faire la substitution plusieurs fois sur la ligne.

On pourra donc fabriquer une fonction formatInput :

1. function formatInput () {
2. sed -n '/id="atmogramme"/,/id="block-mfbulletintendance"/p'   
    | sed 's/<\([^/]\)/\n<\1/g'
3. }

Ce sera l'utilisateur qui va donner sur l'entrée standard le fichier à traiter et en sortie il redirigera vers un fichier temporaire complètement jetable en fin d'extraction.

### Constatations données journalières (dans le fichier)

Voilà un extrait d'une entrée dans les descriptions journalières :

<li class="weather-item" style="width: calc(33.3333%);">

<div class="top">

<div class="h2">après-midi</div>

<p>Ciel voilé</p></div>

<div class="weather\_temp">

<img src="colmar-7j\_fichiers/p4j.svg" alt="Ciel voilé" title="Ciel voilé" class="icon shape-weather">

<strong>19°</strong></div>

<div class="wind\_container">

<p class="wind-speed">

<img src="colmar-7j\_fichiers/SO.svg" alt="Vent SO" title="Vent SO" style="width: 25px;">

<strong>15</strong>

km/h

</p>

<div class="rafales-vent-details">

<span class="rafales\_text is-visible">Rafales</span>

<span class="rafales\_speed is-hidden">55 km/h</span></div></div>

<!----></li>

Qu'en déduis-je ?

* La class "weather-item" permet d'identifier une entrée dans les prévisions.
* La class="h2" est suivi de la période concernée : matin/après-midi/soir/nuit pour les 2 prochains jours (et plus 7 comme à l'époque)
* Sur la ligne en dessous il y a l'état du ciel dans un <p></p>
* La température est 2 lignes après un weather\_temp et/ou dans un <strong></strong>.
* La vitesse du vent est 2 lignes après un wind-speed et/ou dans un <strong></strong>
* Quand il y a des rafales une class "rafales-vent-details" avec des informations dans les 2 lignes suivantes.

Il n'y a pas de mini/maxi mais le reste est disponible. La rafale est en plus, elle n'est pas demandée dans le sujet, mais c'est normal il n'y a pas tous les jours des vents en rafales qui justifient l'alerte de météofrance.

### Choix données journalières (dans le fichier)

Pour me faciliter les traitements, je vais extraire les contenus de bloc weather-item dans un fichier temporaire et les traiter les uns après les autres. Cette manière me permet d'éviter de mélanger des données. Je pourrais tenter de récupérer toutes les données avec un filtre sed unique magique mais d'une part, je n'y crois pas et d'autre part, si des données manquent vais-je arriver à remplacer les trous par un contenu vide (ou en tenir compte) ou si des données supplémentaires sont présentes comment mon filtre va-t-il les traiter. Ces interrogations sont cruciales dans la conception du filtre. Si tout marche bien j'ai un joli filtre, si j'ai une boulette je vais devoir compenser et je risque de perdre beaucoup de temps (j'ai déjà eu cet effet dans un autre TP). Enfin, j'anticipe le traitement des données hebdomadaires, je pourrais répliquer cette approche. Etre systématique c'est bien aussi.

Je vais donc subdiviser le fichier pour avoir juste le bloc en exemple et c'est sur celui-ci que je vais appliquer les extracteurs. Cerise sur le gâteau, je vais pouvoir extraire l'intitulé dans le h2 qui va permettre de tagger correctement les données dans le fichier. Il va falloir tenir compte de h2 car en fin de journée j'ai plus que la nuit (m'enfin je ne sais pas mais ce serait prévisible) et c'est une prévision pour la nuit pas la première de la journée … le matin. C'est important par la suite (ou pas).

### Implémentation données journalières (dans le fichier)

La fonction qui extrait le bloc considéré est moins facile que prévu. Si je ne me repose pas sur un nombre de lignes mais un encadrement <li> </li> c'est mieux mais sed va me les sortir tous en une commande, moi je veux le premier, le deuxième… Une piste simple est de demandé à grep le numéro des lignes où se trouve les <li qu'on recherche puis on utilise une boucle et on demande à sed d'extraire du numéro jusqu'au </li>. Techniquement, ce n'est pas </ mais un < 'n'importe quoi' car on ne peut écrire facilement le / dans cette expression.

1. function getInDayLines () {
2. grep -n '<li class="weather-item"' | cut -d: -f1
3. }
4. for line in `getInDayLines < input.htm`; do
5. sed -n "$line,/<.li>/p" < input.htm > output.htm
6. done

Le fichier intitule input.htm est le fichier reformaté par une des fonctions précédentes et output.htm est un fichier temporaire jetable. Le code d'extraction de donnée sera incrusté à partir de la ligne 11.

Je vais souvent avoir besoin de ciseaux verticaux qui vont extraire des contenus soit dans le contenu texte d'une balise comme dans <div class="h2">**après-midi**</div> ou dans des arguments comme <img src="colmar-7j\_fichiers/SO.svg" alt="**Vent SO**" title="Vent SO" style="width: 25px;"> ou la très bizarre (en réalité sur 3 lignes) <strong>**15**</strong> **km/h** </p> aussi je vais définir trois fonctions pour ce job :

1. function getTagContent () {
2. sed 's![^>]\*>\([^<]\*\)<.\*!\1!'
3. }
4. function getAltAttribute () {
5. sed 's!.\*alt="\([^"]\*\)".\*!\1!'
6. }
7. function getNoisyContent () {
8. tr '\r\n' ' ' | sed -e 's/<[^>]\*>//g' -e 's! \*! !g' -e 's/$/\n/'
9. }

Rq : la function GetNoisyContent pourrait remplacer définitivement la getContent. Mais je préfère avoir mes trois helpers. En plus cette fonction a un comportement différent des autres, car elle remplace les \r et \n par des espaces (du coup il n'y a plus qu'une ligne) puis retire les balises (le sed avec la première expression) et enfin remplace les répétitions d'espace par un seul espace (la deuxième expression du sed) et enfin j'ajoute un saut de ligne à la fin.

Une dernière fonction pour translater l'instant du jour en une clé pour mon fichier. Elle est pratique car elle va me garantir que la clé pour un jour sera toujours bien inscrite dans le fichier. Plusieurs manières de l'écrire, mais je vais supposer que le texte arrive par l'entrée standard, que je vais donc devoir lire, je traduis et j'écris la traduction sur la sortie standard. Les | et `` pourront être utilisés habilement.

1. function textAsKey () {
2. read text
3. key=""
4. case "$text" in
5. matin) key="MORNING";;
6. après-midi) key="AFTERNOON";;
7. soirée) key="EVENING";;
8. nuit) key="NIGHT";;
9. esac
10. echo "$key"
11. }
12. function getInstant () {
13. sed -n -e '/h2/,+1p' | head -1 | getTagContent | textAsKey
14. }

Enfin l'extraction des quatre données intéressantes (selon le sujet) à l'aid de 4 fonctions dédiées à cette extraction (daily par opposition weekly) et après le nom de la donnée :

1. function getDailySky () {
2. sed -n -e '/h2/,+1p' | tail -1 | getTagContent
3. }
4. function getDailyTemperature () {
5. sed -n -e '/weather\_temp/,+2p' | tail -1 | getTagContent
6. }
7. function getDailyWindDirection () {
8. sed -n -e '/wind-speed/,+1p' | tail -1 | getAltAttribute
9. }
10. function getDailyWindSpeed () {
11. sed -n -e '/wind-speed/,+4p' | tail -3 | getNoisyContent
12. }
13. function getDailyWindGusts () {
14. sed -n -e '/rafales\_speed/p' | getTagContent
15. }

Maintenant cela va devenir simple pour extraire juste la partie des prévisions journalières. $analyse référence le fichier résultat dans lequel j'écris mes extractions.

1. for line in `getLines < "$input"`; do
2. sed -n "$line,/<.li>/p" < "$input" > tmp
3. instant="TODAY+`getInstant < tmp`"
4. echo "$instant:SKY:` getDailySky < tmp`" >> "$analyse"
5. echo "$instant:TEMP:`getDailyTemperature < tmp`" >> "$analyse"
6. echo "$instant:WIND\_DIRECTION:`getDailyWindDirection < tmp`" >> "$analyse"
7. echo "$instant:WIND\_SPEED:`getDailyWindSpeed < tmp`" >> "$analyse"
8. echo "$instant:WIND\_GUST:`getDailyWindGusts < tmp`" >> "$analyse"
9. done

Ce code s'insère habilement dans la fonction analyseOneFile déjà entrevue.

Rq : On peut faire mieux (bon ok il est vraiment pourri comme code), car il y a de nombreux >> "$analyse" qui peuvent être compacté avec une écriture comme dans la fonction help. Je vous proposerais cette solution dans le récapitulatif (ou pas).

### Constatations hebdomadaires (dans le fichier)

Voilà un extrait d'une entrée dans les descriptions hebdomadaires :

<li class="period\_item" style="min-width: 103.143px;">

<i class="triangle"></i>

<div class="inside">

<div class="period upper">

mardi 14

</div>

<div>

<img src="colmar-7j\_fichiers/p12bisj.svg" alt="Rares averses" title="Rares averses" class="icon shape-weather "></div>

<div class="weather\_temp">

<p>

<span class="min">5°</span>

/

<span class="max">12°</span></p></div>

<div class="wind">

<p>

<img src="colmar-7j\_fichiers/SSO.svg" alt="Vent SSO" title="Vent SSO" style="width: 15px;">

<strong>30</strong>

km/h

</p>

<div class="rafales-vent-details">

<span class="rafales\_speed is-hidden">50 km/h</span></div></div></div></li>

Qu'en déduis-je ?

* La class "period-item" permet d'identifier une entrée dans les prévisions. Ce n'est pas la même que pour la journalière et le balisage html n'est pas exactement le même.
* La class="period" est suivi de la période concernée sur la prochaine ligne en nom de jour et numéro pas le mois ni l'année. Utiliser cette donnée va demander un reformatage car elle est trop précise (Mardi…) et trop imprécise (Mars 2023).
* L'état du ciel n'est plus lisible mais inscrit dans les annotations d'une balise img, alt servant aux outils pour les personnes mal voyantes je pense qu'il faudra se rabattre sur cette donnée.
* Les températures min et max y sont, plutôt facile, un contenu dans une balise avec une classe adaptée.
* La vitesse du vent est 2 lignes après un wind dans un <p></p>
* Il y a une direction qui est indiquée par une image mais l'alt permet de la connaitre en texte.
* Quand il y a des rafales une class "rafales-vent-details" avec des informations dans la ligne suivante.

Rien de majeur, par contre, vu que le balisage est différent du journalier, il va falloir écrire des extracteurs horizontaux spécialisés dans l'hebdomadaire. Probablement, qu'un ou deux extracteurs horizontaux seront identiques mais faut considérer que c'est un hasard. Pour les extracteurs verticaux, ce serait bien de reprendre les précédents.

Par contre, je ne trouve pas/plus l'indice de confiance et le blabla sur les moyennes je ne sais pas quoi en faire, il n'y a rien qui s'y rapporte. Une nouvelle fois, un magnifique sujet qui a perdu des plumes car les webmestres de météofrance ont changé les données affichées.

### Implémentation hebdomadaires (dans le fichier)

Je vais faire rapide (maintenant). Il va me falloir une fonction pour extraire le bloc considéré. Récupérer l'instant de la période est sans intérêt, car connaitre le nom du jour et son numéro dans le mois n'apporte rien. Je vais plutôt inventer le miens en fabriquant une clé DAY+1, DAY+2, etc. Ce n'est pas trop complexe, une variable qui s'auto-incrément et un préfixe. Je peux avoir la date des prévisions en relatif car le fichier contient la date de début et le nom contient aussi cette information, inutile de recalculer la date des prévisions en clair.

1. function getInWeekLines () {
2. grep -n '<li class="period\_item"' | cut -d: -f1
3. }

Les fonctions d'extraction horizontales :

1. function getWeeklySky () {
2. sed -n -e '/shape-weather/p' | getAltAttribute
3. }
4. function getWeeklyMinTemperature () {
5. sed -n -e '/class="min"/p' | getTagContent
6. }
7. function getWeeklyMaxTemperature () {
8. sed -n -e '/class="max"/p' | getTagContent
9. }
10. function getWeeklyWindDirection () {
11. sed -n -e '/wind/,+2p' | tail -1 | getAltAttribute
12. }
13. function getWeeklyWindSpeed () {
14. sed -n -e '/wind/,+5p' | tail -3 | getNoisyContent
15. }
16. function getWeeklyWindGusts () {
17. sed -n -e '/rafales\_speed/p' | getTagContent
18. }

Une remarque mais j'avoue, ne pas savoir quoi en faire : getWeeklySky devrait aussi marcher pour getDailySky et get\*WindGusts sont effectivement les mêmes.

La partie extraction de la prévision hebdomadaire :

1. instant=0
2. for line in `getInWeekLines < "$input"`; do
3. sed -n "$line,/<.li>/p" < "$input" > tmp
4. ((instant=instant+1))
5. instant="DAY+$instant"
6. echo "$instant:SKY:`getWeeklySky < tmp`" >> "$analyse"
7. echo "$instant:TEMP\_MIN:`getWeeklyMinTemperature < tmp`" >> "$analyse"
8. echo "$instant:TEMP\_MAX:`getWeeklyMaxTemperature < tmp`" >> "$analyse"
9. echo "$instant:WIND\_DIRECTION:`getWeeklyWindDirection < tmp`" >> "$analyse"
10. echo "$instant:WIND\_SPEED:`getWeeklyWindSpeed < tmp`" >> "$analyse"
11. echo "$instant:WIND\_GUST:`getWeeklyWindGusts < tmp`" >> "$analyse"
12. done

Dans le zip, le fichier intitulé meteo-after\_a.bsh devrait illustrer le contenu après cette étape de développement.

## Détails concernant la page web d'une ville à un jour donné

### Constatations source des donnés

Le sujet est clair, vous devez piocher dans le dossier utilisé pour stocker le résultat des analyses. Structure que vous seul connaissez.

### Constatations argument localité

Un premier argument qui est soit la ville, soit le code postal. Merci d'avoir laissé dans les noms de dossiers ou de fichier ces deux données. Elles vont permettre de facilement trouver les fichiers juste avec un nom employant des caractères génériques.

Si $1 est le nom d'une ville un ls $1-\* me donne les fichiers ou les dossiers recherchés

Si $1 est le code postal un ls \*-$1 me donne les fichiers ou les dossiers recherchés

Comme l'écriture des codes postaux et des villes ne se recouvrent pas on peut même écrire une combinaison irréfléchie des deux (en fait seule celle qui marche va trouver des fichiers l'autre ne servira pas) : ls $1-\* \*-$1

Vous noterez l'absence que quottes habituelles lorsque j'utilise une variable. Elle est intentionnelle. Si vous mettez les quottes, le shell va considérer que c'est un texte qu'il va utiliser tel quel, si les quottes sont omis, le shell va réaliser l'expansion des caractères génériques avec le nom de dossiers et des fichiers donc une il va chercher le fichier que je souhaite exploiter.

### Constatations argument date

Il va falloir décoder si l'utilisateur introduit que le jour, le jour et le mois ou le jour le mois et l'année à deux ou quatre chiffres. Cela ferait 4 formats différents.

Je vais me permettre d'ajouter que le séparateur : semble bien pratique pour l'utilisateur (quoique le / aurait été plus logique) et nous nous avons le \_ comme séparateur dans les noms de fichiers. Inventer une fonction pour faire ce travail sera probablement pratique.

Rq: L'informatique c'est quand même spécial … 3 format pour une date 'courte' je ne parle même pas des autres possibilités. Juste une date est déjà l'enfer.

### Constatations sélection des fichiers

Là il y a encore un truc bizarre dans le sujet "Attention, le jour peut correspondre à la date de capture ou à une date de capture antérieure + le décalage correspondant." Comment traduire le Hassenforder en problème à résoudre ? En se souvenant qu'on ne fait pas juste un TP mais une 'application' qui aspire des prévisions à des jours donnés.

Si vous aspirez avec X jours de prévision : les 1, 2 3 et 4 janvier. Si vous demandez les prévisions pour le 4 Janviers vous pouvez prendre les données du jour pour le 4 janvier mais également la prévision à J+1 du 3 janvier, et la prévision à J+2 du 2 janvier et enfin la prévision à J+3 du 1 janvier. Donc déterminer la date du jour est une première étape… mais il faudra aussi déterminer les prévisions des jours antérieurs.

### Possibilités argument date

Pour ce traitement, on peut imaginer plusieurs approches :

* on fait des if avec des expressions régulières et on recombine
* on fait un case avec des caractère générique, il nous donne le format puis on extrait les valeurs et on injecte les valeurs par défaut
* on fait passer un sed magique qui détermine les valeurs puis un deuxième qui injecte les defauts
* on calcule les trois valeurs supportables par l'argument, on calcule les trois valeurs par défaut et on combine avec les trous pour finir par faire une date standard "dd/mm/yy" qui sera réutilisable pour la commande date
* etc

### Possibilités sélection des sources

Il va falloir ruser sinon on part dans de nombreuses lignes de codes toutes aussi inutiles les unes de autres. Si je résume les constatations :

* le nom de ville ou le code postal on s'en sert pour filtrer les dossiers
* la date on s'en sert pour filtrer le fichier dans le dossier
* de la date initiale il faudra 'reculer' dans les jours pour trouver les autres jours qui peuvent proposer des prévisions
* quand on va extraire des données, on peut soit extraire les données de la date du jour soit les données à J+X d'une date antérieure.

Il faut trouver un mécanisme simpliste qui va faire le travail, le petit détail est dans l'extraction des données, elle n'est pas basée sur la date mais un couple (date, instant) et il est impossible d'inférer l'instant si vous n'avez que la date et inversement. Il faudra donc générer des couples avant de penser traitement. Après, il suffit de parcourir les couples et d'appliquer les critères pour trouver les fichiers puis les données.

### Choix argument date

* J'avoue que 4 if j'aime pas les écrire,
* Le case me plait mais je dois rajouter les expressions régulières donc c'est faire 2 fois le travail ça ne me passionne pas.
* J'ai le sed et j'aime sa puissance, mais le code n'est pas suffisamment beau à mon gout (je le laisse dans ce fichier pour info mais je ne retiens pas son usage).
* Il reste les bidouilles sur des expressions régulières

### Implémentation argument date abandonnée

1. day=`date "+%d"`
2. month=`date "+%m"`
3. year=`date "+%y"`
4. two='[0-9]\{2\}'
5. echo "$1" | sed "s!:!/!g" | sed \
6. -e "s!^\$!DD\_MM\_YY!" \
7. -e "s!^\($two\)\$!\1\_MM\_YY!" \
8. -e "s!^\($two\)/\($two\)\$!\1\_\2\_YY!" \
9. -e "s!^\($two\)/\($two\)/\($two\)\$!\1\_\2\_\3!" \
10. -e "s!^\($two\)/\($two\)/$two\($two\)\$!\1\_\2\_\3!" \
11. | sed -e "s/DD/$day/" -e "s/MM/$month/" -e "s/YY/$year/"

Quelques explications …

Lignes 1 à 3 : je récupère les jour, mois et année courantes.

Ligne 4 : je créé une variable avec l'expression régulière qui impose deux chiffres obligatoire. Inutile mais elle simplifie les sed qui suivent.

Ligne 5 : un premier sed me permet de passer sur un format de date plus classique (je remplace les : par des /). Je fais ça au cas où mais ce n'est pas précisé dans le sujet.

Ligne 7 : détection du jour uniquement et je le remplace par lui-même avec MM et YY comme placeholder pour les éléments de la date du jour.

Lignes 8, 9 et 10 : la même chose avec les autres formats.

Ligne 11 : un dernier sed qui va remplacer les placeholders par la valeur des variables. NB : j'aurais pu les introduire directement dans les transformations des lignes 6 à 10 mais …

Ligne 6 : pas demandé mais sans paramètre on fait la date du jour.

Etrangement, je reconnais que cette fonction vue comme ça elle est bizarre mais les autres codages sont carrément laids et étriqués.

### Implémentation argument date conservée

1. function correctDay () {
2. day=`expr "$1" : '\([0-9]\*\).\*'`
3. month=`expr "$1" : '[0-9]\*[:/]\([0-9]\*\).\*'`
4. y1=`expr "$1" : '[0-9]\*[:/][0-9]\*[:/]\([0-9][0-9]\)$'` && y1="20$y1"
5. y2=`expr "$1" : '[0-9]\*[:/][0-9]\*[:/]\([0-9][0-9][0-9][0-9]\)$'`
6. year="$y1$y2"
7. day=${day:-`date "+%d"`}
8. month=${month:-`date "+%m"`}
9. year=${year:-`date "+%Y"`}
10. echo "$month/$day/$year"
11. }

Lignes 2 à 6 récupère la date depuis l'argument. Les expressions sont calibrées pour le champs recherché (premier pour le jour, le deuxième pour le mois, le dernière pour l'année). L'année pouvant être sur 2 ou 4 chiffres il y a deux extracteurs le premier réinjecte le 2000 manquant. Les lignes 7 à 9 remplacent les variables vides par le contenu adapté depuis la date courante. Et enfin la ligne 10 reformate la date. Bizarrerie on est avec le / et on est au format anglais …. C'est pour être compatible avec date pour le recule dans le temps.

### Implémentation sélection des sources

On peut faire chouette si on prend le temps de fabriquer les bonnes fonctions. Il m'en faudra une pour reculer dans le temps, une première qui va me générer toutes les dates et les clés qui peuvent être utilisées (si je demande le 31 janvier : je dois avoir TODAY & 01/31 puis DAY+1 & 01/30 puis DAY+2 & 01/29 etc.), puis une fonction qui va vérifier si le fichier existe et s'il a le contenu souhaité. Bref, j'espère pouvoir écrire ce pipe entre les 2 dernières fonctions:

buildSearchList "$initialDay" | filterSearchList "$locality"

Cela suppose bien entendu que les contenus utiles circulent dans les flux d'entrées et de sorties.

1. function backInPast () {
2. date --date="$1 $2 days ago" "+%m\_%d\_%Y"
3. }
4. function buildSearchList () {
5. initialDay="$1"
6. echo "TODAY+" "$initialDay" | sed 's!/!\_!g'
7. for i in `seq 1 5`; do
8. echo "DAY+$i" `backInPast "$initialDay" "$i"`
9. done
10. }
11. function filterSearchList () {
12. locality="$1"
13. while read key day dummy; do
14. file="$ANALYSES/$locality-\*/\*-$day.txt"
15. [ -f $file ] && grep -sq "^$key" < $file && echo $file $key
16. file="$ANALYSES/\*-$locality/\*-$day.txt"
17. [ -f $file ] && grep -sq "^$key" < $file && echo $file $key
18. done
19. }

BackInPast: ben oui elle est simple car j'utilise date pour faire le travail (impossible d'écrire cette fonction correctement). Au passage, je fabrique la date au format filesystème pour pouvoir l'utiliser rapidement dans mon script.

buildSearchList: redoutable car elle génère le cas particulier du jour courant à la ligne 7 et les autres à la ligne 9 avec une simple boucle (limitée à 5 mais on peut écrire ce qu'on veut à la ligne 8.

filterSearchList: on lit en boucle les différents couples de sélection et on écrit en sortie ceux qui sont utilisables. Pour cela, à la ligne 16, je vais créer une variable qui modélise l'accès au fichier si la localité est le nom de la ville. Les lignes 17 et 19 vont d'abord vérifier si le fichier existe, si oui, si la clé est dans le contenu, si oui, on imprime le nom du fichier et la clé. La ligne 18 construit le nom du fichier en considérant que le paramètre serait un code postal.

### Constatations génération de la page

Si on est sur une ville unique pour une date unique un h2 avec ces informations est parfait et après une table quatre colonnes : instant, jour, grandeur, valeur devrait faire l'affaire. Probablement qu'il faudra juste traduire les 'lignes' dans le fichier de stockage en un format html lisible.

Si ce n'est pas pour un jour ou une ville unique… il faudra probablement imbriquer le format précédent dans une boucle selon la variable d'itération. Avec une fonction et une boucle ou pas on doit y arriver.

### Implémentation génération de la page

Avec l'aide des fonctions précédentes il ne reste plus que :

1. function processDataFile () {
2. echo "<h2> Prévision pour la ville $1 et le jour $2 </h2> "
3. echo "<table border='1'>"
4. while read file key dummy; do
5. grep "^$key" < "$file"
6. done | sed -e 's!^!<tr><td>!' -e 's!:!</td> <td>!g' -e 's!$!</td></tr>!'
7. echo "</table>"
8. }
9. function generateForOneEntry () {
10. locality="$1"
11. initialDay="$2"
12. buildSearchList "$initialDay" \
13. | filterSearchList "$locality" \
14. | processDataFile "$locality" "$initialDay"
15. }

Le sed à la ligne 6 est riche car il injecte les <td></td> dans les : qui servent de séparateurs sans oublier au début d'y coller un <tr> et à la fin le </tr>. C'est rustique mais amplement suffisant. Le vrai reproche c'est d'imprimer les clés des champs dans le vocabulaire de la base de données (oups du fichier), il serait sain de les traduire dans la langue de l'utilisateur. Bon c'est un sed avec 50 substitution qu'il faut faire passer. Pour le TP c'est déjà bien car ce sed c'est peanuts à faire.

### Implémentation génération de la page avec une entrée unique

L'option w doit encore mettre le décors html et reformater la date qui vient des arguments. Mais avec ce qui a été déjà fait c'est redoutablement simple.

1. function option\_w () {
2. messageOnStderr "web for $1 at $2"
3. locality="$1"
4. initialDay=`correctDay "$2"`
5. {
6. echo "<html>"
7. echo "<head>"
8. echo "</head>"
9. echo "<body>"
10. echo "<h1> Les prévisions cumulatives offertes par l'ENSISA </h1> "
11. generateForOneEntry "$locality" "$initialDay"
12. echo "</body>"
13. } > page.html
14. }

### Constatations valeur spéciale

Euh non je vais passer sur ce tour quoique l'idée était d'utiliser les caractères génériques, donc au lieu de laisser la variable à 'rien' ou '-' ou … avec une '\*' à la place le shell devrait sélectionner 'tout' les fichiers. A la base la modification va porter sur la ligne 11.

Dans le zip, le fichier intitulé meteo-after\_w.bsh devrait illustrer le contenu après cette étape de développement.

## Détails concernant la triche

### Constatations

Cette commande semble très simple… Elle l'est car reculer la date de X jours on sait déjà faire et traiter tous les fichiers … Ben il faut juste itérer dessus. Bon ok, il y aura bien une difficulté, il faut non renommer le fichier en changeant la date dans le nom mais également changer la date dans le contenu. Ce changement n'est pas indispensable, mais nécessaire uniquement si on veut rester cohérent. En fait, on peut aussi la laisser, car en définitive je ne m'en sers pas, elle permettrait alors de savoir que la date du fichier c'est la date 'virtuelle' et la date dans le contenu c'est la date 'réelle', ce qui est pas mal aussi.

### Possibilités

Itérer sur tous les fichiers, cela va dépendre de la structure de fichiers adoptée.

* Si on est à plat, un simple : ls et c'est bon,
* Si on a deux niveaux, un simple : ls de ls ou un ls \*/\* et c'est bon,
* Si on a des niveaux des ls de ls de ls (oups que je ne vais pas apprécier)
* Et plus simplement un find qui va abstraire les dossiers.

### Choix

Clairement le find c'est la solution la plus générique de toutes les solutions. Sur sa sortie standard il y le nom des fichiers à traiter et une simple boucle va faire le taff.

### Implémentation

1. function cheatOne () {
2. oldname="$1"
3. amount="$2"
4. name=`expr "$oldname" : '\(.\*\)-.\*\.txt'`
5. olddate=`expr "$oldname" : '.\*-\(.\*\)\.txt' | sed 's!\_!/!g'`
6. newdate=`date --date="$olddate $amount days ago" +"%m\_%d\_%Y"`
7. newname="$name-$newdate.txt"
8. echo mv "$oldname" "$newname"
9. }
10. function cheatAll () {
11. find "$ANALYSES" -name '\*.txt' -print | while read file dummy; do
12. cheatOne "$file" "$1"
13. done
14. }

Vous noterez juste que le mv n'est pas actif (au cas où il y aurait une grosse bêtise), je ne fais qu'afficher ce que je dois faire.

### Implémentation de l'option –t

1. function option\_t () {
2. messageOnStderr "cheat with $1"
3. amount="$1"
4. [ -z "$amount" ] && before="1"
5. cheatAll "$amount"
6. }

Oui c'était trop simple.

Dans le zip, le fichier intitulé meteo-final.bsh devrait illustrer le contenu après cette étape de développement.