TP Unix : Générateur de « galerie d'images » en HTML

1 Vue d'ensemble du TP

Objectif pratique : à partir d'un répertoire contenant des images en format JPEG, construire une page HTML avec un aperçu des images, des "vignettes".

Objectifs pédagogiques : Apprendre la programmation shell et la notion de Makefile ; premiers contacts avec le HTML et avec le parallélisme.

Comment s'y prendre ? Il est fortement conseillé de faire une première lecture complète du sujet. En particulier, les sections 3.5 et 4.7 devraient vous aider (mais vous n'en comprendrez le contenu que si vous avez au moins parcouru le reste).

Le TP propose d'écrire un générateur de galerie d'images de deux manières différentes, l'une en utilisant uniquement des scripts shell, l'autre en utilisant également des Makefiles de manière à paralléliser certaine tâches.

Ces deux versions devront réutiliser au maximum le même code (i.e. pas de copier-coller, le Makefile et les scripts doivent être dans le même répertoire, et le Makefile doit appeler les scripts autant que possible).

On demande un compte-rendu, de deux pages maximum (i.e. 1 feuille recto-verso), donnant une vue d'ensemble rapide de votre programme, et répondant aux questions posées ci-dessous.

2 Premiers contacts avec HTML

La plupart des pages web sont des documents au format HTML, un langage de balises permettant de coder dans le même document le contenu texte, la structure (titres, sous-titre, ...) et la mise en forme (italique, gras, ...). L'utilisateur final ne voit en général que le résultat de la mise en forme par le navigateur, mais on peut aussi voir le code source HTML de n'importe quelle page (Menu « View », « Page Source », raccourcis clavier Control-U sous Firefox). On peut comparer HTML à LaTeX, qui a aussi cette notion de code source (`.tex`) et de version mise en forme (PDF, DVI ou PS).

L'utilisation la plus classique de HTML est le web : les pages sont stockées sur un serveur et téléchargées par un navigateur web via le protocole HTTP. On peut aussi écrire du HTML dans un fichier (avec l'extension `.html`, ou `.htm`), et charger ce fichier dans le navigateur (par exemple avec la commande `firefox toto.html`).

Nous vous proposons un apprentissage (partiel) de HTML par l'exemple : visitez la partie « Exemples de pages HTML et de galeries d'images » du site web de ce cours, et suivez les indications que vous y trouverez.

3 Version séquentielle du générateur (shell-script uniquement)

Dans cette partie, on va écrire une première version du générateur qui utilise des scripts shell uniquement. Pour simplifier l'écriture du script shell qui génère l'ensemble de la galerie, on va écrire quelques scripts utilitaires effectuant des parties du travail. L'avantage est que l'on pourra réutiliser certains de ces scripts utilitaires dans la seconde version de notre outil.

3.1 Fonctions auxiliaires pour générer du HTML

On va commencer par écrire un fichier `utilities.sh` (cf. squelette fourni) qui contiendra les fonctions utilitaires (« helper functions ») qui nous faciliteront la tâche par la suite. Ces fonctions sont très simples, il suffit d'appeler la fonction `echo` dans chaque fonction. On demande au minimum ces 3 fonctions :

`html_head` : affiche un en-tête HTML (i.e. le début du fichier à générer, jusqu'à la balise `<body>` incluse) sur la sortie standard. On peut prendre un argument `$1` donnant le titre de la page (i.e. le texte placé dans la balise `<title>`).

`html_tail` : affiche un pied de page HTML (de la balise `</body>` jusqu'à la fin du fichier, avec éventuellement un texte ou un logo en bas de page)

`html_title` : affiche son premier argument (`$1`) sous forme de titre HTML (i.e. encadré par des balises `<h1>` et `</h1>`).

Le fichier `test-html.sh` fourni vous permet de tester ces fonctions. Vous pouvez lire son code source pour voir comment le fichier `utilities.sh` est inclus dedans.

3.2 Génération du fragment de HTML pour une image (`generate-img-fragment.sh`)

On va maintenant écrire un script `generate-img-fragment.sh`, qui prend en premier argument le chemin vers une image (le nom de fichier se termine obligatoirement par `.jpg`), et qui affiche sur sa sortie standard un fragment de code HTML du type :

```
<img src="image.jpg">
```

ou quelque chose de plus évolué (cf. les exemples de galeries à votre disposition). L'attribut `src="..."` contient le nom du fichier sans le répertoire. En effet, ce fragment de HTML sera intégré dans une page qui référencera les vignettes (générées dans le même répertoire que le fichier HTML) et non l'image source.

Ce script pourra être réutilisé sans modification dans la seconde version de notre générateur (d'où l'intérêt d'avoir décomposé le travail en plusieurs petits scripts).

3.3 Appel des autres scripts et génération des vignettes (`galerie-shell.sh`)

En se basant sur les fonctions définies dans `utilities.sh` et sur le petit script `generate-img-fragment.sh`, on demande d'écrire un script `galerie-shell.sh`, en langage « shell » (sh ou bash), prenant au minimum les paramètres suivants :

`--source REP` : Répertoire contenant les images JPEG à miniaturiser.

`--dest REP` : Répertoire cible (où on va générer les vignettes et le fichier HTML).

`--verb` : mode "verbeux" pour mise au point : on affiche les commandes (convert, cp, ...) appelées par le script avant de les exécuter. Regardez ce que font les options `--verb` de commandes comme `mv` ou `cp` si vous ne comprenez pas de quoi il s'agit. La sortie du script doit rester courte et lisible (typiquement, les solutions à base de `set -x` et `set -v` ne sont pas acceptables).

`--force` : forcer la création de vignette, même si la vignette existe déjà.

`--help` : Afficher la liste des options disponibles, et quitter.

`--index FICHIER` : générer la galerie dans le fichier spécifié au lieu de générer un fichier `index.html`.

Le script devra générer un fichier `index.html` dans le répertoire cible, et pour chaque image du répertoire source :

— Si (et seulement si) la vignette n'existe pas encore, la créer (la vignette est une version réduite de l'image)

— Appeler `generate-img-fragment.sh` pour générer le morceau de HTML correspondant.

Pour générer l'ensemble de la galerie, le script va faire un appel à `html_head`, concaténer les résultats de `generate-img-fragment.sh`, puis appeler `html_tail` pour créer le fichier `index.html`.

3.4 Tests semi-automatisés

Pour vérifier que votre script fonctionne correctement, nous vous demandons une petite base de tests, c'est à dire un ensemble de scripts qui eux-mêmes vont lancer le générateur de galerie dans des conditions différentes. A priori, ces scripts ne vérifieront pas par eux-mêmes si la galerie est correcte, il faudra le vérifier à la main.

Pour vous aider, nous vous fournissons :

— Un script `make-img.sh`, qui prend comme argument un nom de fichier, et crée une image avec ce nom. Par exemple, `make-img.sh toto.jpg` crée une image `toto.jpg`, et `make-img.sh 'mon image.jpg'` crée une image `mon image.jpg`.

— Quelques exemples de scripts de tests (`tests/simple.sh`, `tests/star.sh`). Essayez-les, lisez-les, et écrivez-en d'autres plus complets.

Cette base de tests est à compléter en ajoutant des scripts dans le répertoire `tests/`.

3.5 Indications, recommandations

Voici quelques recommandations :

— Pour visualiser le fichier `index.html`, faites simplement `firefox index.html`
— Pour la création des vignettes utiliser la commande `convert` (qui fait partie de la suite ImageMagick) : `convert -resize 200x200` *`source cible`*
— Pour la forme générale du fichier HTML à produire, vous trouverez plusieurs exemples de code HTML de galerie sur la page du cours.
— On peut utiliser le répertoire d'images `/matieres/3MMUNIX/exifImages/` qui contient quelques fichiers JPEG avec données EXIF.
— Utiliser la redirection d'entrées/sorties de façon astucieuse
— Pour la redirection d'entrées/sorties et l'analyse des arguments sur la ligne de commande, on pourra s'inspirer du petit script fourni en annexe.
— Pensez bien au cas où `--source` et/ou `--dest` ne sont pas le répertoire courant ; d'une manière générale, les chemins relatifs sont source de problèmes dans les scripts, c'est une bonne idée de les rendre absolus au début du script (la commande `pwd` peut aider, si on la combine astucieusement avec la commande `cd`). C'est une bonne idée d'ajouter des tests pour les différents cas à votre base de tests (cf. 3.4).
— Structurez votre code avec des fonctions.
— Pour la mise au point, on pourra exécuter le script avec `sh -x ./galerie-shell.sh` pour une trace d'exécution détaillée
— Utilisez l'utilitaire `shellcheck` pour vérifier votre code. Il est disponible pour votre distribution linux ou directement en ligne à l'adresse `http://www.shellcheck.net`.

4 Version parallèle du générateur (en utilisant `make -j N`)

On va maintenant faire une autre version du générateur de galerie, qui permette d'exploiter le parallélisme de la machine sur laquelle il tourne. La majorité des PC de l'Ensimag ont 4 cœurs (salles E303, E200, E201, D200, D201). Les PC les plus puissants sont ceux de la E303 (8 Go de RAM), mais cela ne changera pas les performances pour ce TP. Ces machines peuvent donc exécuter jusqu'à 4 fils d'exécution en même temps. Notre première version n'était pas capable d'exploiter ce parallélisme.

L'utilitaire `make` permet d'exploiter de manière simple le parallélisme des machines : avec l'option `-j`, `make` va s'autoriser à lancer plusieurs processus en parallèle, mais en respectant les dépendances (i.e. l'action d'une règle ne sera exécutée qu'*après* que toutes actions correspondant aux dépendances soient terminées). En général, c'est utilisé pour compiler plus rapidement des logiciels, mais nous allons en faire une utilisation un peu détournée pour générer notre galerie HTML. Si on utilise `make -j 10`, on autorise `make` à lancer jusqu'à 10 processus en parallèle. Si on utilise `make -j` sans argument, on ne limite pas le nombre de processus parallèle, **ce qui va en général faire ramer la machine**, et qu'il faut donc **éviter** autant que possible.

On ne pourra pas (ou du moins pas simplement) avec `make` obtenir une belle interface en ligne de commande avec les options (`--verb`, `--source`, ...), donc on se contentera d'indiquer les paramètres source et destination directement dans le fichier `Makefile`. L'affichage, et la régénération ou non de miniatures existantes sont contrôlés par `make`, et on ne cherchera pas a priori à implémenter les équivalents de `--force` ou `--verb` pour la

version du générateur utilisant `make`.

4.1 Génération des fragments de HTML

Dans le Makefile, écrire une règle permettant de générer, pour un fichier `*.jpg` donné dans le répertoire source, un fichier `*.inc` dans le répertoire cible. Ce fichier contiendra le fragment de HTML généré par le script `./generate-img-fragment.sh` (qui a été écrit pour la version script-shell de notre outil, mais que l'on peut réutiliser ici) à partir des images sources (qui sera donc utilisé dans l'action de cette règle).

On pourra tester cette règle avec :

```
make dest/image1.inc
cat dest/image1.inc
```

4.2 Génération du fichier `index.html`

Une fois les fichiers `*.inc` générés, il faudra les concaténer, et ajouter un en-tête et un pied de page (générés avec `html_head` et `html_tail`). Cette opération se fait en quelques lignes de shell, on peut au choix écrire ces quelques lignes directement dans l'action du `Makefile` (mais attention, le retour à la ligne a une signification particulière pour `make`, il faut l'échapper avec un \), ou plus simplement écrire un autre script, par exemple `./generate-index.sh` qui réalise les appels à `html_head` et `html_tail`, et la concaténation des `*.inc`.

On peut tester cette règle avec :

```
make dest/index.html
cat dest/index.html
```

Pour l'instant, le fichier `index.html` contient des balises `<img>` qui pointent sur des images qui ne sont pas encore générées, donc le fichier s'affichera mal dans un navigateur.

4.3 Génération des vignettes

Écrire maintenant une règle qui pour chaque fichier `*.jpg` dans le répertoire source, va générer une version miniaturisée (vignette, ou *thumbnail* en anglais) dans le répertoire cible.

4.4 Faire marcher le tout ...

Nous avons maintenant les éléments de base, il reste à écrire une règle pour générer l'ensemble de la galerie (`index.html` et les vignettes). Écrire une règle `gallery` (qui ne correspondra pas à un vrai fichier, donc une « Phony target » dans la terminologie de GNU Make, avec ce que ça entraîne pour l'écriture du Makefile), qui permette de faire ceci. Cette règle n'aura pas nécessairement besoin d'action : il suffit qu'elle ait les bonnes dépendances.

Ajouter maintenant une cible `view` qui lance le navigateur `firefox` sur le fichier `index.html` (après avoir si besoin généré la galerie).

Si vous avez bien travaillé, la commande `make view`, lancée avec un répertoire cible vide ou non, devrait lancer Firefox sur une page correctement générée.

Pour terminer, ajouter une cible `clean` qui permette de supprimer les fichiers générés via `make clean`.

4.5 Vue graphique

Votre Makefile définit en fait un *graphe des tâches*. Une représentation graphique consiste à écrire les cibles du Makefile et/ou les actions dans des rectangles, et de représenter les dépendances par des flèches entre ces rectangles. Faites une représentation graphique dans votre rapport, en précisant les conventions graphiques que vous avez utilisées. Le graphe des tâches devra faire apparaître toutes les cibles et les actions du Makefile.

4.6 Est-ce bien parallèle ?

Si vous avez bien écrit votre code, la commande `make view` devrait donc générer la galerie, puis lancer un navigateur dès que le travail est terminé. La commande `make clean; time make gallery` vous dira combien de temps la génération a pris.

Essayons maintenant la génération en parallèle : `make clean; make -j 10 view`. Si vos dépendances sont correctes, vous devriez avoir le même résultat, mais plus rapidement. Les commandes `convert` et `./generate-img-fragment.sh` devraient pouvoir s'exécuter en parallèle, et la génération du fichier `index.html` ne devrait pas commencer avant que tous les fragments de HTML soient générés. Par contre, on peut tout à fait générer ce fichier `index.html` avant que la génération des vignettes soit terminée.

La commande `make clean; time make -j 10 gallery` vous permettra de voir le temps gagné. Essayez différentes valeurs pour le paramètre `-j`. Avec quelle valeur obtient-on les meilleurs résultats ? Expliquez vos mesures de performances et votre conclusion dans votre rapport.

4.7 Indications et conseils

Pour vous aider à débugger vos Makefiles et à évaluer les performances :

— La commande `make -d` permet de demander à `make` d'afficher beaucoup d'informations sur la raison pour laquelle une action est exécutée. Cette commande est *très* verbeuse, donc en général, on l'utilise avec les options permettant de désactiver les règles par défaut de `make`, comme ceci : `make -rRd`

— Pour voir la valeur de certaines variables, on peut écrire une cible `debug` sans dépendances et avec comme action `echo UNE_VARIABLE=$(UNE_VARIABLE)`. La commande `make debug` permettra alors d'afficher la valeur de cette variable.

— Pour mesurer les performances, ne pas oublier de faire `make clean` avant de faire `time make ...`, sinon, seule une partie de la galerie risque d'être régénérée (c'est le principe de `make` de ne régénérer que ce qui est nécessaire depuis le dernier lancement de `make` ...).

— Quand on utilise la commande time, on obtient trois temps différents : le temps « real » est le temps écoulé entre le début et la fin de la commande, le temps « user » est la somme de temps passé par les processeurs pour exécuter nos processus, et

le temps « sys », ou « system » est celui passé dans le noyau de notre système d'exploitation pour exécuter le tout. Pour un processus séquentiel, le temps « real » est toujours supérieur ou égal à la somme « user + system ». Un système parallèle change en général peu les temps « user » et « system », mais permet de réduire le temps « real ».

— On fera l'hypothèse que les chemins ne contiennent pas de caractères spéciaux (espaces, `$`, ...). En effet, la commande `make` ne permettrait pas de les gérer correctement sans une gymnastique peu élégante ...

5 Améliorations des scripts

5.1 Le minimum : ajout d'une légende sous les images

Votre squelette contient un répertoire `exiftags-1.01/` qui contient les sources de l'utilitaire `exiftags`, qui permet d'extraire des commentaires d'images. Le Makefile fourni dans votre squelette contient une version simplifiée du Makefile officiel d'exiftags, donc un simple `make ./exiftags` dans la racine de votre squelette devrait créer un exécutable `exiftags` dans ce répertoire.

Complétez les deux versions (scripts et Makefile) de votre générateur pour ajouter sous chaque vignette une légende, composée du nom du fichier (sans le préfixe répertoire), de la date de prise de vue, et éventuellement d'autres paramètres fournis par la commande *exiftags*, commande à essayer pour connaître les informations fournies. En principe, une modification du fichier `generate-img-fragment.sh` devrait permettre ceci. Ajustez également les dépendances de votre Makefile pour assurer que l'utilitaire `exiftags` est toujours compilé avant d'être utilisé. Ajoutez les tâches correspondant à la compilation de exiftags sur votre graphe des tâches dans le rapport.

Au passage, regardez le fichier `exiftags-1.01/Makefile`, qui correspond au Makefile fourni avec exiftags. En dehors du fait qu'il est plus compliqué que le notre, vous pourrez remarquer que les dépendances sont incorrectes :

```
cd exiftags-1.01/
make
# modifier un fichier .h de votre choix
# (c'est l'équivalent d'un .ads en Ada)
make
```

Préciser dans votre rapport où se trouve le problème, et expliquez ce qu'il faudrait faire pour le corriger.

5.2 Pour aller plus loin : Navigation d'une image à l'autre

On peut aussi, en plus de la page de vignettes, créer une page HTML par photo contenant l'image en pleine taille (toujours avec la balise `<img>`), et faire pointer la vignette de l'index vers cette page (en utilisant un code HTML de la forme
`<a href="page.html"><img src="photo.jpg"></a>`).

La page HTML créée pourra avoir des boutons « Suivant », « Précédent », et « Retour à l'index ».

6 Résumé des livrables

En résumé, votre rendu contiendra :
— La version séquentielle du générateur de galerie (scripts shell)
— Les scripts de tests permettant de vérifier que cette version marche, y compris sur des cas tordus (répertoire `tests/`)
— La version parallèle (Makefile)
— Le rapport

7 Indication sur le barème

Pour obtenir la note 12/20, il faut avoir un code qui marche, y compris dans des cas un peu tordu (cf. section « indications » ci-dessus) pour la version de base sans légende (version shell-script et version Makefile), une base de tests raisonnable, et un rapport correctement écrit. Pour obtenir une note supérieure à 12, il faut avoir travaillé sur la version avec légende.

La note 16/20 correspond à un TP implémentant la version minimale, la légende, sans bug et avec un code propre. La partie « pour aller plus loin » (ou autres choses en plus) permet de monter au dessus.

Le but du TP est avant tout d'apprendre la programmation en shell, les Makefiles, et d'avoir un premier contact avec le parallélisme. On peut bien sûr s'amuser avec HTML, faire de la mise en forme avec les feuilles de style CSS, ajouter du JavaScript, mais ce n'est pas l'objet du cours. Si vous faites de telles extensions, elles ne seront prises en compte (typiquement par un bonus d'un point sur 20) que si le reste est irréprochable. Produire du code HTML valide (cf. le validateur en ligne `http://validator.w3.org/`) est un plus, mais concentrez-vous surtout sur le code de votre programme.

Toute copie (sur une autre équipe, un projet d'une année passée—y compris votre ancien TP pour les redoublants—ou autre) sera sanctionnée par un 0/20 ou un passage en conseil de discipline. Lire la charte contre la fraude pour les détails : `https://intranet.ensimag.fr/teide/Charte_contre_la_fraude.php`