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 modifié des images, des "vignettes" artistiques.

Objectifs pédagogiques : Apprendre la programmation shell ; premiers contacts avec le HTML.

Comment s'y prendre ? Il est fortement conseillé de faire une première lecture complète du sujet. En particulier, la section 3.4 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 en utilisant uniquement des scripts shell.

Les fichiers contenant le code de votre générateur seront stockés en utilisant le gestionnaire de version `git`.

On demande un compte-rendu en HTML, équivalent à une page de texte maximum, donnant une vue d'ensemble rapide de votre programme.

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 notre générateur utilisant des scripts shell. 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 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 départ 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. Il faudra reprendre cette méthode pour vos autres scripts.

3.1.1 Génération du fragment de HTML pour une image (`generate_img_fragment`)

On va maintenant écrire, dans `utilities.sh`, une fonction `generate_img_fragment`, 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.

Notation annoncée : `-2`, si le fichier `utilities.sh` contient autre chose que des fonctions.

3.2 Appel des fonctions et génération des vignettes (`galerie-shell.sh` et `galerie_main`)

En utilisant les fonctions définies dans `utilities.sh`, on demande d'écrire un script `galerie-shell.sh`, en langage « shell » (sh ou bash). Ce script prend en argument certains paramètres et appelle la fonction de traitement `galerie_main` avec les arguments adéquats. Le script traite 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`.

Pour générer l'ensemble de la galerie, le script va faire un appel à une nouvelle fonction de `utilities.sh`, `galerie_main` en lui passant les paramètres adéquats.

Notation annoncée : -2, si le script `galerie-shell.sh` contient autre chose que, le chargement de la bibliothèque de fonction, le traitement des arguments et l'appel de la fonction `galerie_main`.

3.2.1 `galerie_main`

La fonction `galerie_main` 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 artistique est une version transformée et réduite de l'image)

— Appeler `generate_img_fragment` pour générer le morceau de HTML correspondant.

Cette fonction appelle notamment `html_head`, concaténe les résultats de `generate_img_fragment`, puis appelle `html_tail` pour créer le fichier `index.html`.

3.3 Tests semi-automatisés

Pour vérifier que votre script fonctionne correctement, nous vous demandons une 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/`.

3.4 Indications, recommandations

Voici quelques recommandations :

— Pour visualiser le fichier `index.html`, faites simplement `firefox index.html`

— Pour la création des vignettes utiliser la commande `gmic` : `gmic` *`source`* `-cubism , -resize 200,200 -output` *`cible`* (les deux virgules sont significatives).

— 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, comme la date de prise de vue.

— 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`), comme dans les scripts fournis. C'est une bonne idée d'ajouter des tests pour les différents cas à votre base de tests (cf. 3.3).

— 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 Améliorations des scripts

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

Complétez 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 *identify -verbose* (du packetage ImageMagick), commande à essayer pour connaître les informations fournies. En principe, une modification de la fonction `generate_img_fragment` devrait permettre ceci.

4.2 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 ».

4.3 Pour aller plus loin : Génération des vignettes en parallèle (avec `xargs -n 1 -P 4`)

Enfin on peut faire une 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 programmes en même temps. Notre première version n'était pas capable d'exploiter ce parallélisme.

Néanmoins, si votre galerie comporte des dizaines de milliers d'images, le lancement de dizaines de milliers de `gmic` simultanément pourrait poser des problèmes.

La commande `xargs` permet de lancer simplement en parallèle des commandes, tout en limitant leurs nombres à un instant donné.

Pour donner un exemple, la suite de commandes `time ( echo 1 2 3 4 | xargs -n 1 -P 4 sleep )` s'exécute pendant 4 secondes : `sleep 1`, `sleep 2`, `sleep 3` et `sleep 4` s'exécutent en même temps. Tandis que la commande `time ( echo 1 2 3 4 | xargs -n 1 -P 1 sleep )` s'exécute pendant 10 secondes car les 4 `sleep` sont réalisés les uns après les autres.

En utilisant `xargs`, vous devriez pouvoir créer les vignettes et générer les entêtes en exploitant le parallélisme de votre machine.

Vous rajouterez à vos scripts une option `-n X`, ou `X` est un entier qui sera passé à l'option `-P` de xargs. Sa valeur par défaut sera 4.

5 Utilisation de `git`

Les fichiers contenant le code de votre générateur seront développés et stockés en utilisant le gestionnaire de version `git` (si vous ne savez pas encore l'utiliser, vous le verrez en cours pendant la séance 3).

Votre rendu devra inclure à la fois les fichiers de travail (`utilities.sh`, `galerie-shell.sh`, ...) et le répertoire `.git/` qui contient l'historique de votre projet. Nous vous imposons les bonnes pratiques de base de l'utilisation de Git : (1) votre historique doit contenir plusieurs commits (au moins un au nom de chaque étudiant de l'équipe), et (2) la dernière version doit être correctement commitée : pas de fichier non-tracké, pas de modification non-commitées (en d'autre terme, `git status` doit afficher `On branch master`, `nothing to commit, working tree clean` et rien d'autre). Inversement, (3) seuls les fichiers sources pertinents doivent être versionnés (i.e. pas de `git add` sur des fichiers générés).

Notation annoncée L'évaluation de votre usage de `git` comptera pour 4 points dans la note :

0 pas de répertoire `.git` dans le rendu,

1 un répertoire `.git` dans le rendu,

$>$ **1** au moins 1 commit non vide,

$+$**1** `git status` n'indique rien (condition (1) ci-dessus),

+1 tous les membres de l'équipe ont un *commit* à leur nom (2),

+1 aucun fichier généré, ou fichier de backup, n'est versionné (3).

6 Le compte-rendu

On demande un compte-rendu en HTML, équivalent à une page de texte maximum, donnant une vue d'ensemble rapide de votre programme. Le compte rendu sera placé dans un sous-répertoire de votre projet nommé `rapport/`.

Vous préciserez ce que vos tests vérifient. Vous indiquerez aussi comment lancer chacun de ces tests. Le cas échéant, vous préciserez les extensions implantées.

Enfin, dans une partie annexe du rapport qui ne compte pas dans sa longueur maximum, vous inclurez la sortie de l'utilitaire `shellcheck` associée à chacun de vos fichiers shell.

7 Résumé des livrables

En résumé, votre rendu contiendra un fichier TAR, éventuellement compressé, contenant l'ensemble des éléments suivants :

— La version séquentielle (ou parallèle) du générateur de galerie (scripts shell) dans son répertoire de travail git (répertoire `.git/` compris).

— Les scripts de tests permettant de vérifier que cette version marche, y compris sur des cas tordus (répertoire `tests/`)

— Le rapport en HTML sera inclus et géré dans le répertoire de travail git (répertoire `rapport/`).

8 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), 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. 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`