Alias, redirections & réécritures d'URL

> Your wish is my command. But be careful what you wish for.
>
> Le neuvième docteur

Alors que la majeure partie des URL [1] reçues par un serveur Web désignent un chemin relatif par rapport à la racine du serveur (directive `DocumentRoot` pour Apache, `root` pour Nginx), il est également possible d'associer certaines URL à des emplacements qui ne suivent pas cette règle. Dans ce contexte :

— Un *alias* permet d'associer un emplacement non standard à une URL. Tout est alors réalisé directement sur le serveur, l'opération étant invisible [2] pour le client.
— Une *redirection* consiste à associer une nouvelle URL à une URL reçue en requête. C'est alors au client de suivre la nouvelle URL… ou pas : une redirection peut être interne. C'est alors au serveur de se débrouiller.

Ces fonctionnalités sont disponibles via deux modules Apache différents, idem pour Nginx, à ceci près que l'un des deux est `ngx_http_core_module` [3], c'est à dire celui qui fournit la possibilité d'utiliser Nginx comme serveur web.

NB : dès que vous avez besoin d'un module, prenez soin d'aller lire la documentation du module au moins une fois. Ça ne fait pas de mal et vous aurez des réponses avant de vous poser les questions (du genre, « Dans quelle contexte puis-je utiliser cette directive ? ») [4].

1 Alias

1.1 Apache

NB : les redirections sont traitées avant les alias. Les alias sont traités dans l'ordre d'apparition dans la configuration et seule la première correspondance est appliquée.

Le module `mod_alias` [5] est a priori le plus utilisé pour gérer les alias et les redirections. Il est généralement assez flexible pour répondre à la majorité des besoins et doit être utilisé en priorité lorsque son usage permet de résoudre un problème donné.

La gestion des alias s'appuie sur les directives suivantes :

— `Alias`, permettant d'associer un nom de dossier à une URL, sachant que ce dossier n'est pas obligatoirement sous la racine du serveur (le *DocumentRoot*). Son emplacement est désigné par un chemin absolu.
Attention : si l'URL déclarée par `Alias` se termine par un `/`, les chemins devront également en comporter un.
Syntaxe et exemples d'utilisation :

```
Alias <URL> <chemin absolu d'un répertoire>
Alias /images /usr/local/share/images
Alias /errors/include/ /usr/local/share/apache2/errors/
```

— `AliasMatch`, qui fonctionne de la même manière que la directive `Alias`, mais qui permet d'identifier les URL concernées à l'aide d'expressions régulières.
— `ScriptAlias`, fonctionnant également de la même manière de `Alias`, mais désignant un répertoire dans lequel des scripts CGI pourront être exécutés [6].

1. https://grisebouille.net/url-uberlu/
2. Je hais quand on dit *transparent* dans ce genre de contexte : un verre est transparent, pourtant vous le voyez, non ? Grmpf.
3. http://nginx.org/en/docs/http/ngx_http_core_module.html
4. Je rabâche, je sais :P
5. http://httpd.apache.org/docs/2.4/mod/mod_alias.html
6. Voir http://httpd.apache.org/docs/2.4/mod/mod_alias.html#scriptalias pour plus de détails

— `ScriptAliasMatch`, qui fonctionne selon le même principe que `ScriptAlias` en utilisant des expressions régulières.

1.2 Nginx

Deux directives peuvent être utilisées pour créer des alias dans Nginx : `alias` et `root`.

On les utilisera pour cela à l'intérieur d'un contexte `location` (**NB** : n'oubliez pas que `root` peut être utilisé dans d'autre contextes) :

```
location /images {
    alias /usr/local/share/img/;
}
location /errors/include/ {
    root /usr/local/share/apache2/errors/;
}
```

Il y a une différence de comportement entre `root` et `alias` :

```
location /foo {
    root /var/www;
    # alias /var/www;
}
```

Si l'on demande le fichier `https://exemple.org/foo/bar/index.html`, avec `root` dans la configuration, on aura le contenu de `/var/www/foo/bar/index.html` et avec `alias`, on aura `/var/www/bar/index.html`.

Avec `root`, le fichier fourni sera `<dossier fourni en argument à root>/<chemin complet de la requête>`, avec `alias`, ce sera `<dossier fourni en argument à root>/<chemin complet de la requête moins le chemin spécifié dans le contexte location>`.

La documentation d'`alias` indique qu'il est préférable d'utiliser `root` quand la `location` correspond à la dernière partie de l'alias (`/images/` et `/usr/local/share/images/` par exemple).

Si la `location` est utilisée avec une regex, celle-ci pourra comporter des captures qui seront utilisées dans l'alias (ou avec `root`) :

```
location ~ ^/users/(.+\.(?:gif|jpe?g|png))$ {
    alias /data/w3/images/$1;
    # root /data/w3/images/$1;
}
```

Notez que les chemins passés en arguments de `root` et `alias` peuvent contenir des variables[7] (sauf `$document_root` et `$realpath_root`).

2 Redirections

2.1 Apache

La gestion des redirections s'appuie quant à elle sur les directives suivantes :

7. Voir https://nginx.org/en/docs/varindex.html

— `Redirect` (qui est aussi fourni par `mod_alias`), qui permet d'effectuer des redirections « simples », avec une syntaxe basée sur les préfixes : `Redirect [état] ancien_chemin nouveau_chemin`. Ainsi, si l'URL d'une requête commence par `ancien_chemin`, toute l'URL correspondant à la requête est transformée en substituant dans la requête d'origine la base de l'URL et `ancien_chemin` par `nouveau_chemin`. Le `nouveau_préfixe` peut être une URL absolue, avec protocole et nom d'hôte ou être un chemin URL commençant par un `/`, auquel cas ce seront le protocole et nom d'hôte du serveur qui seront utilisés.
Les états utilisables correspondent à une des valeurs suivantes : `permanent` (`301`), `temp` (`302`, cet état est l'état par défaut qui sera utilisé si on ne précise pas l'état), `seeother` (`303`), `gone` (`410`, pas de nouveau préfixe d'URL fourni dans ce cas) ou à un code de retour numérique. Dans ce dernier cas, l'argument `nouveau_chemin` ne doit être défini que si on utilise un code entre 300 et 399.

```
# redirection de http://localhost/support/find.php
# en http://support.example.org/find.php
Redirect 301 /support http://support.example.org/
Redirect permanent /support http://support.example.org/
# redirection de http://localhost/contact/index.html
# en http://contact.example.org/index.html
Redirect /contact http://contact.example.org/
```

— `RedirectMatch`, qui fonctionne comme la directive `Redirect`, mais en considérant des expressions régulières. Les expressions capturées (entre parenthèses) peuvent être utilisées dans la dernière partie de la directive grâce à `$1`, `$2`, `$3`, etc.

```
RedirectMatch permanent (.*)\.gif$ http://example.org/icons/$1.png
```

— `RedirectTemp` : permet de faire une redirection temporaire, renvoyant donc un code 302, et est strictement équivalente à `Redirect temp ...`
— `RedirectPermanent` : idem que pour la directive précédente, avec `permanent` à la place de `temp`.

Pourquoi utiliser les deux dernières directives si elles sont équivalentes à `Redirect xxx` ? Sans doute pour la lisibilité (un terme comme `temp` étant plus parlant qu'un code de retour, même si on finit par les connaître par cœur) et sans doute pour l'économie d'un caractère à taper :-).

Blague à part, Les redirections permanentes et temporaires sont suffisamment souvent utilisées pour avoir leur propre directive, qui indique d'un coup d'œil de quelle redirection il s'agit.

2.2 Nginx

On utilisera la directive `return` du module `ngx_http_rewrite_module`[8], avec généralement en argument un code de retour (en chiffres) et une URL (absolue ou relative).

```
return 301 https://support.example.org/;
return 301 /foo/bar;
return http://localhost/baz/qux;
```

Il est possible de retourner uniquement un code (`return 404;`), un code accompagné d'un texte (`return 401 "foo";`) ou juste une URL (`return http://localhost/baz/qux;`), auquel cas le code de retour est 302 (redirection temporaire) et l'URL doit commencer par `http://`, `https://` ou `$scheme`.

Pour spécifier dans quel cas on redirige, on placera `return` dans un contexte `location` ou `if` :

```
if ($http_user_agent ~* 'googlebot') {
    return 401;
}
location /support {
    return 301 http://support.example.org/;
}
```

8. http://nginx.org/en/docs/http/ngx_http_rewrite_module.html

3 La réécriture d'URL

NB : Certaines règles de réécriture peuvent pourrir les performances du serveur, bien que la plupart n'auront qu'un impact limité. Tout dépend de la complexité des réécritures successives.

3.1 Apache

Tout comme `mod_alias`, le module `mod_rewrite`[9] permet de gérer les alias et les redirections, mais de façon beaucoup plus fine. Il est ainsi beaucoup plus puissant et beaucoup plus complexe. À titre d'exemples, il est ainsi possible avec ce module de :

- Faire tout ce que `mod_alias` permettait déjà de faire en étendant ses possibilités.
- Définir des conditions permettant d'activer ou non certaines règles en fonction de la condition testée. Les conditions peuvent porter sur des variables internes du serveur, comme sur des valeurs systèmes ou des valeurs liées aux requêtes reçues.
- Faire des comparaisons entre chaînes de caractères.
- Tester l'existence de pages et, dans la négative, d'en créer une de toute pièce.
- De faire appel à des programmes extérieurs pour assister la réécriture.
- Enfin, ses fonctionnalités peuvent être configurées au niveau du serveur, d'un répertoire (conteneur `<Directory>` ou contexte htaccess) ou d'un hôte virtuel.

C'est un module très complet et très complexe, à utiliser lorsque la situation le nécessite.

NB : Bien que les règles de réécriture soient permises du point de vue de la syntaxe dans les sections `<Location>` et `<Files>` (y compris leurs versions sous forme d'expression régulière `*Match`), elles n'y sont pas prises en compte, et n'y sont à priori d'aucune utilité.

Pour commencer

Pour activer le moteur de réécriture dans un hôte virtuel, vous devez y écrire `RewriteEngine On` : les hôtes virtuels n'héritent pas des configurations de réécriture. Pour activer le moteur dans un contexte répertoire ou htaccess, vous devez *en plus* y écrire `Options FollowSymLinks` (et vous devez faire en sorte que cette option soit autorisée, voir la directive `AllowOverride`).

Comme les règles de réécriture sont relativement complexes à mettre en place, il est bon d'activer dans la configuration de votre hôte virtuel (le temps des tests ! C'est gourmand en IO disque) un niveau spécial de journalisation :

```
LogLevel alert rewrite:trace3
```

Créer une réécriture

La syntaxe d'une règle de réécriture est :

```
RewriteRule Modèle Substitution [drapeaux]
```

Le *Modèle* est une expression régulière compatible Perl.

Ce modèle est comparé au chemin de l'URL lorsque la règle de réécriture est placée dans un contexte d'hôte virtuel : pour `https://example.org/foo/bar.html?baz=1`, le *modèle* sera comparé à `/foo/bar.html`.

Si la règle est dans un contexte de répertoire ou htaccess, le chemin où la règle est définie est supprimé du chemin correspondant du système de fichier avant comparaison : si vous placez la règle dans un contexte correspondant à `%{DOCUMENT_ROOT}`, la requête `https://example.org/foo/bar.html?baz=1` aboutira à

9. http://httpd.apache.org/docs/2.4/mod/mod_rewrite.html

une comparaison de la régle avec `foo/bar.html` (notez la suppression du premier `/`). Si la règle est dans un contexte correspondant à `%{DOCUMENT_ROOT}/foo/`, la comparaison sera faite avec `bar.html`.

La *Substitution* peut être :

- un chemin du système de fichier : Les substitutions ne sont traitées en tant que chemins du système de fichiers que si la règle est configurée dans un contexte de serveur (serveur virtuel), et si le premier composant du chemin dans la substitution existe dans le système de fichiers ;
- un chemin d'URL : un chemin relatif à la valeur de `DocumentRoot` vers la ressource qui doit être servie ;
- une URL absolue : si le nom d'hôte n'est pas celui de l'hôte virtuel, il y aura redirection du client, sinon le protocole et le nom d'hôte sont supprimés et traité comme un chemin d'URL ;
- un tiret `-` : aucune substitution n'est effectuée. Utile quand on veut juste appliquer des drapeaux sans modifier le chemin.

Quelques *drapeaux* (voir la documentation de `mod_rewrite` pour tous les avoir) :

- `END` : stoppe le processus de réécriture, plus aucune règle de réécriture n'est appliquée, pas même les règles de réécriture des contextes de répertoire et de fichier .htaccess ;
- `last|L` : stoppe le processus de réécriture mais les règles des contextes de répertoire et de fichier .htaccess peuvent encore être appliquées ;
- `nocase|NC` : rends la comparaison insensible à la casse
- `next|N` : provoque un redémarrage du traitement des règles depuis le début, mais avec le résultat des règles déjà exécutées. Attention aux boucles infinies ;
- `redirect|R[=code]` : force une redirection externe, avec un code de statut HTTP optionnel. Si le code n'est pas spécifié, ce sera le code 302 qui sera renvoyé.

On peut combiner les drapeaux en les séparant par une virgule

Voici un exemple basique de réécriture :

```
RewriteRule "^/index\.html$"  "/welcome.html" [L]
```

À partir de là, Apache traite la requête comme `/welcome.html`

```
RewriteRule "^/index\.html$"  "/welcome.html" [NC,R=301]
```

Là, on indique une redirection au client, en faisant une comparaison insensible à la casse.

RewriteBase

Cette directive ne s'applique que lorsque la règle est définie un contexte répertoire ou `.htaccess`.

Elle permet de spécifier le préfixe d'URL à utiliser dans un contexte de répertoire (htaccess) pour les directives `RewriteRule` qui réécrivent vers un chemin relatif.

Exemple :

```
DocumentRoot "/var/www/site-web"
<Directory "/var/www/site-web/">
    RewriteEngine On
    RewriteBase "/foobar/"
    RewriteRule "^index\.html$"  "welcome.html"
</Directory>
```

Une requête `GET /` va être traitée comme suit :

- apache rendre dans le dossier `/var/www/site-web/`

— apache compare la chaîne vide `''` à la regex `^index\.html$` qui ne correspond pas, pas de réécriture
— apache tente à présent avec la chaîne `index.html` (puisque c'est ce fichier qui doit normalement répondre à la requête `/`, à cause de la directive `DirectoryIndex`)
— la réécriture s'opère : `index.html` devient `welcome.html`
— à cause de la directive `RewriteBase`, au lieu d'être réécrite en `/welcome.html` (`/` étant le préfixe du chemin de l'URL, soit l'URL sans le fichier), la requête est réécrite en `/foobar/welcome.html`

NB : `RewriteBase` ne s'applique que pour les `RewriteRule` qui réécrivent vers un chemin relatif. Si nous avions eu `RewriteRule "^index\.html$"  "/welcome.html"`, `RewriteBase` ne se serait pas appliquée.

RewriteCond

Cette directive définit une condition qui devra être satisfaite pour que la réécriture soit effectuée. La réécriture ne sera effectuée que si la condition est rencontrée ET si l'URL correspond à la règle de réécriture.

Sa syntaxe est :

```
RewriteCond chaîne_de_test expression_de_comparaison [drapeaux]
```

Son utilisation basique est de tester sur du texte simple ou une variable du serveur, mais on peux aussi utiliser des références arrières de règle de réécriture, des références arrières de condition de réécriture ou des extensions de règles de réécriture (usage avancé non traité ici, voir la documentation [10]).

Exemple :

```
RewriteCond /var/www/%{REQUEST_URI} !-f
RewriteRule ^(.+) /other/archive/$1 [R]
```

La redirection ne sera effectué que si le fichier `/var/www/%{REQUEST_URI}` n'existe pas. On a utilisé ici une variable du serveur dans la chaîne de test (voir la documentation [11] pour les variables disponibles) et le test `-f` *est un fichier* avec un `!` de négation.

Lorsque qu'il y a plusieurs `RewriteCond`, il faut que toutes les conditions correspondent pour appliquer la réécriture (drapeau `[AND]` implicite) mais on peut utiliser le drapeau `[OR]` :

```
RewriteCond "%{REMOTE_HOST}"  "^host1"  [OR]
RewriteCond "%{REMOTE_HOST}"  "^host2"
RewriteRule ^(.+) /other/archive/$1 [R]
```

À noter : l'existence du drapeau `[NC]` pour rendre la condition insensible à la casse.

Pour utiliser plusieurs drapeaux, on fait comme pour les règles de réécriture, en les séparant par des virgules : `[NC,OR]`

Voir la documentation pour les différentes expressions de comparaison.

3.2 Nginx

Ici, nous utiliserons la directive `rewrite`. Elle utilise en argument une regex qui sera comparée à l'URL demandée, l'URL de redirection (qui pourra réutiliser des termes capturés dans la regex) et éventuellement un *flag*.

10. http://httpd.apache.org/docs/2.4/mod/mod_rewrite.html#rewritecond
11. http://httpd.apache.org/docs/2.4/mod/mod_rewrite.html#rewritecond

```
server {
    rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3;
    rewrite ^(/download/.*)/audio/(.*)\..*$ $1/mp3/$2.ra  last;
    ...
}
```

Les réécritures sont traitées séquentiellement et peuvent donc s'exécuter l'une après l'autre. Lorsque l'URL de redirection commence par `http://`, `https://` ou `$scheme`, le traitement s'arrête et la redirection est retournée au client.

Pour stopper le traitement, on utilisera les *flags* suivants :

- `last` : les autres `rewrite` ne sont pas regardés et Nginx regarde comment traiter la requête avec la nouvelle URL ;
- `break` : les autres `rewrite` ne sont pas regardés mais Nginx cherche à traiter la requête sans repartir de zéro (contrairement à `last`) ;
- `redirect` : redirection temporaire (avec le code 302) ;
- `permanent` : redirection permanente (code 301).

Notez qu'on peut placer des `rewrite` dans des `if`, permettant ainsi de singer de façon simpliste la fonctionnalité `RewriteCond` d'Apache.

Pour débuguer les réécritures dans Nginx, on mettra le log d'erreur au niveau `notice` et on activera la journalisation des réécritures avec `rewrite_log on` :

```
rewrite_log on;
error_log /var/log/nginx/site.error.log notice;
```