Apache : la réécriture d'URL

Table des matières

Liste des tableaux

Apache : la réécriture d'URL

julp

14/12/2012

Résumé

J'ai toujours voulu consacrer un article à la fonctionnalité incontournable qu'est la réécriture d'URL pour Apache. Il faut cependant admettre que c'est un sujet très difficile à aborder tant il est technique et tant il est complet. Il y a en effet beaucoup de choses à dire pour couvrir de manière correcte ce vaste sujet. J'espère remplir cet objectif au travers du présent document par une description technique abordable, complétée de différents cas pratiques courants.

---

1. Avant propos

1.1. Ce qu'est et n'est pas la réécriture

Avant de débuter, il est important d'être clair sur la définition de "réécriture d'URL", faute de voir trop souvent une erreur d'interprétation quant à cette expression : **la réécriture d'URL ne modifie pas les liens de la source HTML que vous générez !** La réécriture d'URL, à la base, est un processus qui redirige de façon purement interne une URL virtuelle (= qui n'existe pas) sur une URL réelle. "Interne", au sens où, la redirection n'est pas une redirection HTTP : le

client ignore tout de ce processus, rien n'apparaît de son côté ; c'est le serveur HTTP lui-même qui se charge d'effectuer cette traduction. En conséquence, bien que la page servie au final soit toute autre, l'adresse vue par le client, dans sa barre de navigation, reste inchangée à ce qu'il a saisi ou au lien qu'il a suivi.

Pourquoi utiliser la réécriture d'URL ?

- pratiquer l'obscurantisme, c'est-à-dire cacher aux utilisateurs l'URL réelle donc les éventuels détails techniques qui se cachent derrière, ceci pour éviter :
  - qu'ils ne jouent, manuellement, avec les URLs (les paramètres en query string notamment) ;
  - qu'ils ne devinent de quelles technologies dépendent votre site. Avec des extensions de pages en .php, .asp ou .jsp, ceux-ci peuvent aisément émettre quelques hypothèses ...
- améliorer votre référencement. Un article traitant des sessions en PHP et ayant pour URL article.php?id=2 se trouvera surement dans les bas fonds de toute recherche avec un moteur de recherche quelconque quand article-2-les-sessions-en-php.html sera certainement mis un peu plus en avant.

Tout d'abord, d'un point de vue purement théorique, la réécriture n'entrera en jeu que si vous modifiez vos liens. Pour reprendre l'exemple de l'article :

```
<a href="article.php?id=2">Les sessions en PHP</a>
```

Doit à présent devenir :

```
<a href="article-2-les-sessions-en-php.html">Les
sessions en PHP</a>
```

Suite à cela, quand le client suivra le lien réécrit, article-2-les-sessions-en-php.html, il ignorera totalement que c'est le script article.php?id=2 qui est finalement invoqué, comme à l'origine. La traduction article-2-les-

sessions-en-php.html => article.php?id=2 étant réalisée de manière interne par le serveur.

Il existe quelques moyens valables pour réellement modifier vos liens initiaux sans votre intervention, il faut cependant noter que :

- cette tâche peut s'avérer techniquement fastidieuse à mettre en place et pire à maintenir
- le procédé mis en œuvre a un coût qui peut s'avérer non négligeable

C'est pourquoi, dans la mesure du possible, il est préférable de prévoir la réécriture dès le départ de façon à avoir des liens (adresses) qui font directement intervenir la réécriture, sans besoin d'intervenir globalement sur le code HTML produit.

Toutefois, à titre purement illustratif, voici un moyen en PHP couplant bufferisation de sortie et DOM (les expressions régulières sont à bannir pour une telle tâche) :

```
<?php
$bdd = new PDO(/*...*/);

function slugify($string) {
    /* voir implémentation ci-dessous */
}

function implode_url($path, $qs, $anchor) {
    $url = $path;
    if ($qs) {
        $url .= '?' . (is_string($qs) ? $qs :
http_build_query($qs, '', '&'));
    }
    if ($anchor) {
        $url .= '#' . $anchor;
    }
    return $url;
}

/**
```

```
 * Notes :
 * - DOM renvoie de l'UTF-8 indépendamment du jeu du
document de départ
 * - Le principe est simplifié : les ports HTTP(S) non
standard ne sont pas gérés entre autres
 **/
function rewrite_links(PDO $bdd, $content) {
    $dom = new DomDocument;
    if (!$dom->loadHTML($content)) {
        return $content;
    }
    $xpath = new DomXPath($dom);
    foreach ($xpath->query('//a') as $link) {
        if (!$link->hasAttribute('href')) { # pas
d'attribut href
            continue;
        }
        $href = $link->getAttribute('href');
        if (FALSE === ($parts = parse_url($href))) { #
parse_url a échoué
            continue;
        } else {
            $args = array();
            $parts = $parts +
array_fill_keys(array('path', 'host', 'scheme', 'query',
'fragment'), FALSE);
            if ($parts['scheme'] &&
!in_array($parts['scheme'], array('http', 'https'))) { #
le protocole n'est ni http ni https
                continue;
            }
            if ($parts['host'] && $parts['host'] !=
$_SERVER['HTTP_HOST']) { # le lien désigne un serveur
extérieur
                continue;
            }
            if ($parts['query']) {
                parse_str($parts['query'], $args);
            }
        }
        # C'est ici qu'il faut traiter/modifier les
liens
        # Exemple: article.php?id=2 => article-2-
```

```
<titre>.html
        if ($parts['path'] == 'article.php' && FALSE !==
($id = filter_var($args['id'], FILTER_VALIDATE_INT))) {
            $stmt = $bdd->query('SELECT titre FROM
billets WHERE id = ' . $id));
            if (FALSE !== ($titre = $stmt-
>fetchColumn())) { # On ne réécrit pas l'adresse d'un
article inexistant
                unset($args['id']);
                $link->setAttribute('href',
                    implode_url(
                        sprintf('article-%d-%s.html',
$id, slugify($titre)),
                        $args,
                        $parts['fragment']
                    )
                );
            }
        }
    }

    return $dom->saveHTML();
}

ob_start();
?>

<!-- Le contenu normal de la page -->

<?php
$content = ob_get_clean();
echo rewrite_links($bdd, $content);
```

Par ailleurs, je vous recommande de prendre soin d'étudier la mise en place d'une réécriture, quitte à prendre, s'il le faut, un papier et un crayon : non seulement, il vaut mieux modifier le moins possible des règles de réécriture qui sont en production mais tout conflit entre vos différentes règles pourrait avoir des répercussions néfastes ! Petite anecdote à ce sujet, on voit souvent des gens, qui se sont lancés dans la mise en place d'une réécriture sans prendre le temps de l'étudier : ils ont choisi le tiret (-) en guise de séparateur

des valeurs qui sont concaténées ensemble pour finir par se rendre compte bien plus tard que ces valeurs peuvent en fin de compte comporter un tiret ... Dès lors, il est impossible de retrouver les valeurs attendues ou bien l'URL conduit à une 404 parce qu'il y en a plus que prévu.

1.2. PHP/SQL pour les besoins d'illustration autour d'un cas pratique dynamique

Pour le cadre de cet article et ses quelques extraits de code illustratifs, nous définissons les pré-requis suivants :

- disposer d'une base de données MySQL (toute adaptation ne demande que quelques minutes vu que le SGBD est exploité via PDO)
- posséder une version de PHP >= 5.4.0
- extension intl activée et, de fait, nous travaillerons exclusivement en UTF-8
- pour simplifier et rendre les codes moins longs, que l'ensemble de nos scripts sont directement à la racine du site

Nous définirons une table minimaliste nommée billets, décrivant un blog fictif, décrit par la structure suivante :

```
CREATE TABLE billets(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    titre VARCHAR(255) NOT NULL,
    -- date, auteur, etc non montrés
    PRIMARY KEY(id),
    UNIQUE KEY(titre)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Les fonctions PHP, à reproduire, intervenant tout au long de l'article sont :

```
<?php
function slugify(/*UTF-8*/ $string) {
    return transliterator_create_from_rules("::Latin;
::Lower; ::Latin-ASCII; ([^a-z0-9])+ > \-")-
>transliterate($string);
}

function http_not_found() {
    header('HTTP/1.1 404 Not Found', 404, TRUE);
    exit;
}

function http_redirect_permanent($to) {
    header('HTTP/1.1 301 Moved Permanently', 301, TRUE);
    header('Location: http://' . $_SERVER['HTTP_HOST'] .
$to);
    exit;
}
```

2. Bref rappel des bases de la syntaxe des expressions rationnelles

Note

PCRE, la bibliothèque greffée à Apache qui se charge des expressions régulières, est bien plus complète que ce que je décris là. Beaucoup de ses aspects sont passés sous silence non pas parce qu'ils ne sont pas intéressants mais pour la simple et bonne raison qu'ils ne sont pas applicables dans ce contexte. Citons :

- le support (partiel) d'Unicode
- certaines classes n'ont pas réellement de sens dans une URL (espaces verticaux ou caractères non imprimables devant être urlencodés, voir RFC 3986)

2.1. L'alternative (|)

Pour entrer dans le vif du sujet, commençons par le plus simple : l'alternative. Comme son nom l'indique, elle ne désigne rien d'autre que le fait de savoir si une sous-chaîne A ou une sous-chaîne B (ou une sous-chaîne C ou une sous-chaîne D ...) est, ou non, contenue dans une chaîne d'origine. L'alternative est représentée par la barre verticale, symbole |, pour avoir sens de "ce qui la précède **ou** ce qui la suit".

Ainsi déterminer qu'une chaîne contient une voyelle, revient à chercher le caractère 'a' ou 'e' ou 'i' ou 'o' ou 'u' ou 'y'. Le problème étant clairement exposé, on peut constater que sa traduction de notre langue naturelle au motif correspondant, `a|e|i|o|u|y`, est triviale.

2.2. Les ancres (^ et $)

Les ancres sont, à mon sens, le point clé à la compréhension des expressions régulières :

- le symbole ^ caractérise le début de la chaîne : il contraint la chaîne à débuter par un motif. Exemple : `^article` accepte toute chaîne débutant par "article". On notera que marquant le début de la chaîne, on ne devrait rien trouver qui ait un sens littéral à sa gauche.
- à l'inverse, le métacaractère $ désigne la fin de la chaîne : il force la chaîne à se terminer par un motif. Exemple : `php$` est vrai pour toute chaîne finissant par "php". Étant donné que $ marque la fin de la chaîne, on ne devrait rien trouver qui ait un sens littéral à sa droite.

Il est important de comprendre que sans ancre, l'expression régulière peut trouver une sous-chaîne **n'importe où** :

- soit l'expression régulière `part`, ancrée en aucune façon, elle validera les chaînes suivantes : "**part**icule", "re**part**i", "rem**part**", "**part**"
- à présent, avec `^part`, ancrée sur le début, seuls "**part**icule" et "**part**" du point précédent satisfont l'expression
- maintenant avec `part$`, ancrée sur la fin de la chaîne, il ne nous reste plus que "rem**part**" et "**part**"
- enfin, avec `^part$`, ancrée sur le début et la fin, nous n'avons plus que "**part**"

On peut remarquer que dans le dernier cas, avec le double ancrage, et pour un motif aussi simple, seule la chaîne littérale elle-même peut être validée. Quand nous n'avons pas d'autres outils que les expressions régulières afin de valider une chaîne précise, c'est là la seule solution.

2.3. Les quantificateurs (répétitions et/ou omission)

Un quantificateur permet de dénombrer très précisément le caractère qui le précède par un minimum et/ou un maximum. D'une part, ils sont bien plus commodes que les équivalents que vous pourriez écrire : l'expression de 3 à 5 caractères 'a' resterait simple à retranscrire en aa(a|aa|aaa) mais l'est beaucoup moins sur de bien plus larges plages. D'autre part, comment représenteriez-vous (théoriquement) l'infini ? Enfin, autre avantage, c'est que ce minimum en nombre d'occurrences peut être nul, ce qui permet alors de rendre la présence du caractère facultative.

Suivant les valeurs données à ces deux bornes, il existe plusieurs quantificateurs, qui se distinguent par leur syntaxe :

- ? : un caractère **optionnel**, il peut apparaître 0 ou 1 fois. Exemple : ab?c accepte les sous-chaînes "abc" ou "ac", autrement dit le caractère 'b' est facultatif entre 'a' et 'c'.
- * : un caractère qui peut être répété **un nombre quelconque de fois**, zéro compris. Exemple : ab*c accepte les sous-chaînes "ac", "abc", "abbc", "abbbc", etc. Le caractère 'b' peut apparaître 0, 1, 2, 3, 4, ..., 100, ... fois entre 'a' et 'c'.
- + : un caractère qui doit apparaître **au moins une fois**. Exemple : ab+c accepte les sous-chaînes "abc", "abbc", "abbbc" mais pas "ac". Le caractère 'b' doit apparaître au moins une fois entre les caractères 'a' et 'c'.
- {x} : le caractère qui précède doit être présent **exactement** x fois consécutives.
- {x,y}, avec y > x : le caractère devant doit être présent **de** x, minimum, **à** y, maximum, fois.
- {x,} : un caractère qui doit être présent **au moins** x fois.

Note

En réalité, les minimum et maximum ne seront réellement honorés que si vous avez une contrainte supplémentaire avant et après. Par exemple, si je prends l'expression régulière `b{2}`, qui consiste à trouver deux caractères 'b' consécutifs, une chaîne comme a**bb**ba satisfait ce masque sur les deux premiers caractères 'b' consécutifs trouvés dans la mesure où l'expression n'impose rien sur le contenu avant ou après ceux-ci. Ici, ces deux caractères 'b' peuvent être trouvés n'importe où, la notion de minimum et maximum ne s'applique pas. On pourrait indépendamment utiliser `b{2,}` dans ce cas de figure.

Si maintenant j'ajoute des tirets comme séparateur en guise de contexte avant et arrière, l'expression devient `-b{2}-`, alors seuls exactement deux 'b' consécutifs, précédés et suivis d'un tiret établissent la correspondance (exemple : a-b-bbb-**bb**-a).

On peut établir les équivalences de forme suivantes :

- ? est synonyme de {0,1} ;
- * est théoriquement équivalent à {0,+∞} - l'infini ne pouvant être représenté, il correspondrait aux limites de notre machine ;
- + est identique, dans l'idée, à {1,+∞} ;
- de même, {X,} a sens de {X,+∞} ;
- {1} est toujours superflu, ce quantificateur est implicitement associé, par défaut, à toute composante d'un motif.

2.4. Métacaractères contre caractères

Un métacaractère est un caractère qui a une signification particulière au sein d'une expression régulière. Nous avons vu pour le moment les métacaractères :

- d'alternative : | ;
- d'ancrage : ^ et $;

- de quantification : ?, *, +, { et }.

Mais, puisqu'ils ont un sens particulier, comment leur faire reprendre leur sens de caractère littéral ? Comme dans beaucoup de langages, il faut, dans ces circonstances échapper c'est-à-dire faire précéder un tel caractère d'un antislash (\) pour que ce caractère perde toute signification spéciale. Par contre, du coup, l'antislash lui-même devient un métacaractère. Si nous voulions un antislash dans le motif, il sera nécessaire de le doubler (\\).

Dans les faits, par rapport au protocole HTTP, la plupart de ces caractères ne peuvent figurer tel quel dans une URL et sont alors (url)encodés. Cependant, laissons de côté le domaine d'application et ses limites, si nous voulions chercher quelque chose qui s'apparente aux variables dynamiques de PHP, de la forme `${ ... }`, un motif possible pourrait être le suivant : `\$\{[^}]+\}`. Remarquez que le caractère $ a été échappé, sans quoi il aurait sens de fin de chaîne, dès lors nous n'obtiendrions jamais de résultat ; de même pour les accolades, qui rendraient invalides l'expression puisque, sans, c'est une ou des quantités numériques qui seraient attendues entre elles, pour contenu.

2.5. Les classes de caractères

Nous avons précédemment vu l'alternative mais que se passe-t-il si je veux décrire une large plage de caractères ? Admettons que je veuille représenter une consonne minuscule ('b' ou 'c' ou 'd' ou 'f' ou ...) : que devrais-je écrire ? L'expression régulière correspondante serait :

```
b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|z
```

Plutôt laborieux n'est-ce pas ? Dans cette situation, une classe est normalement bien plus adaptée.

Une classe est introduite par des crochets : le crochet ouvrant ([) marque son début et le crochet fermant (]), sa fin. Elle a sens de : **un caractère parmi ceux qui**

forment son contenu, chaque caractère est uni par un **ou logique**, comme l'alternative, d'où l'analogie initiale. Ainsi, le précédent exemple peut avantageusement être réécrit sous la forme d'une classe que voici :

```
[bcdfghjklmnpqrstvwxz]
```

Toutefois, les classes permettent de représenter les caractères par un ou plusieurs intervalles en séparant le caractère de code ASCII le plus faible du plus fort par un tiret. Au sein d'une classe, `a-z`, désigne alors l'intervalle mathématique [a;z], soit du caractère 'a' à 'z', tous deux compris (la représentation d'une lettre minuscule quelconque en somme). Par conséquent, notre expression régulière visant à représenter une consonne minuscule peut évoluer en une forme où les groupes de consonnes consécutives peuvent être écourtés sous la forme d'un intervalle :

```
[b-df-hj-np-tv-xz]
```

Les classes ne s'arrêtent pas là et possèdent un avantage que les alternatives n'ont pas : celui de pouvoir être niées. En ajoutant un chapeau (^) juste derrière le crochet ouvrant d'une classe, la classe devient négative. Elle prend alors le sens de : un caractère qui n'est pas parmi ceux qu'elle contient. Dès lors, si nous admettions que l'alphabet était limité aux seules lettres minuscules, une consonne minuscule ne serait-elle pas, en ces circonstances, un caractère qui n'est pas une voyelle minuscule ? Ce qui nous aurait permis de nouveau de simplifier notre motif de départ par une classe négative :

```
[^aeiouy]
```

Attention : il ne faut pas confondre l'usage du métacaractère ^ qui sert d'ancre de début de chaîne lorsqu'il est en dehors d'une classe et de négation de

classe quand il suit immédiatement le crochet ouvrant d'une classe.

Note

Tous les métacaractères deviennent de simples caractères littéraux au sein d'une classe, donc nul besoin de les échapper, à l'exception de :

- le tiret (-) est promu métacaractère sauf s'il est situé en première (négation - ^ - exclue) ou dernière position de la classe. Ainsi, pour qu'un tiret reste littéral à l'intérieur d'une classe et s'il n'est pas en première ou dernière position, il devra être échappé ;
- l'accent circonflexe seul, en tête de classe, indique une négation. S'il doit être considéré comme littéral, et positionné en première position, il est nécessaire de l'échapper ;
- le crochet fermant marque la fin de la classe. Pour être littéral, il doit également être échappé ;

Pour savoir si une chaîne contient un métacaractère quelconque, il est possible d'utiliser la classe ci-dessous :

```
[-^$|?*+{}[\]]
```

Si les conditions de position sont respectées pour le tiret et l'accent circonflexe (^), comme ici, seul le crochet fermant nécessite un échappement.

Notons que les expressions régulières POSIX, complétées par PCRE, prévoient un certain nombre de classes nommées pour les cas usuels :

Tableau 1. Une partie des classes nommées et autres raccourcis PCRE

Description	Notation abrégée	Quelques autres formes équivalentes

Description	Notation abrégée	Quelques autres formes équivalentes
\d	Un chiffre décimal	[0-9] ou [[:digit:]]
\D	Un caractère qui n'est pas un chiffre décimal	[^0-9] ou [^\d] ou [^[:digit:]]
[[:xdigit:]]	Un chiffre hexadécimal	[0-9a-fA-F]
\w	Un caractère composant un "mot"	[0-9a-zA-Z_]
\W	Un caractère ne composant pas un "mot"	[^0-9a-zA-Z_] ou [^\w]
[[:alpha:]]	Une lettre	[a-zA-Z]
[[:alnum:]]	Une lettre ou un chiffre décimal	[a-zA-Z0-9] ou [[:digit:][:alpha:]] ou [\d[:alpha:]]
[[:lower:]]	Une lettre minuscule	[a-z]
[[:upper:]]	Une lettre majuscule	[A-Z]
.	Tout caractère excepté \n. Noter que, de fait, . est un métacaractère. Pour faire référence à un simple caractère point dans un motif, il vous faudra l'échapper (\.).	[^\n]

Avertissement

- Nous aurons l'occasion de détailler la question plus tard, mais sachez qu'Apache ne gère que les caractères de l'ASCII non étendu (points de code dont la valeur se situe dans l'intervalle [0;127]). Par conséquent, toutes ces classes ne comprennent notamment pas les caractères accentués.
- Une classe, positive comme négative, ne correspond qu'à un caractère à la fois. Une expression régulière

telle que `[^abc|def]` n'interdit pas les sous-chaînes "abc" ou "def". Cette dernière a sens de : **un** caractère qui n'est ni a, ni b, ni c, ni |, ni d, ni e, ni f. Dans la même idée, on pourrait être tenté d'écrire `[^d][^e][^f]` en pensant désigner une sous-chaîne qui n'est pas "def", ce qui est faux. Ce motif est beaucoup plus large, il interdit bien la sous-chaîne "def" mais aussi toutes les variantes contenant au moins un de ses caractères ("dxy", "xey", "xyf", etc seraient également rejetées, ce qui est bien différent). La solution pour interdire une sous-chaîne n'est pas une classe mais une assertion négative !

- Lorsque l'on désigne un intervalle par la forme X-Y au sein d'une classe de caractères, ceci implique que le caractère Y possède un code ASCII supérieur à celui de X. Si cette condition n'est pas respectée, vous déclencherez une erreur 500.

2.6. Groupement et mémorisation

J'ai volontairement raisonné en terme de caractères jusqu'ici dans la mesure où les métacaractères, de quantifications notamment, sont "fainéants". Ils ne s'appliquent qu'à ce qui les précède directement, donc, par rapport à mes précédents exemples, à un caractère ('b' pour `ab*c`). Heureusement, les métacaractères peuvent avoir un effet bien plus étendu qu'un simple caractère, ce qui nous amène à la notion de groupement. Il faut alors entourer de parenthèses, comme nous le ferions en mathématiques en fait, la partie du motif concernée. Par exemple : `-(\d{3})+-` accepte une sous-chaîne composée d'un multiple de 3 chiffres (et comprenant au moins 3 chiffres) entourée de part et d'autre par des tirets.

Toutefois, cette notion de groupement va quelque peu au-delà, puisqu'il existe une nuance selon que l'on veuille, ou non, conserver en mémoire la partie correspondant à la sous-expression contenue entre ces parenthèses :

- avec mémorisation, forme (...) : la sous-chaîne qui satisfait la sous-partie du motif entre de telles parenthèses est conservée en mémoire. Ceci permet de s'y référer, à l'intérieur de l'expression régulière en cours comme en dehors, sous certaines limites, pour usage ultérieur. Apache en permet jusqu'à 9.
- sans mémorisation, syntaxe (?: ...) : bien que la forme avec mémorisation soit plus largement utilisée sans distinction, celle-ci possède l'avantage de ne pas consommer inutilement de mémoire, même si cette consommation s'avère négligeable mais, d'autre part, vu que le nombre de captures est très limité, elle peut s'avérer nécessaire sur un motif très complexe et long.

Il est parfaitement possible de mélanger ces deux types de parenthèses dans une même expression, ce serait même conseillé pour ne capturer que ce qui est ensuite réellement exploité. Prenons le motif `^(?:topic|sujet|thema)-(\d+)-(?:(\d+)-)?.*\.html$` soit le mot "sujet" en anglais, français ou allemand (sans le capturer car cette information est supposée inutilisée), un tiret, un nombre (l'identifiant du sujet) que l'on mémorise (capture n°1, première parenthèse capturante en partant de la gauche), un tiret, éventuellement un autre nombre (le numéro de la page) que l'on capture (en 2) plus un tiret, un nombre quelconque de caractères (le titre du sujet pour le référencement) et l'extension .html.

Avec la chaîne "sujet-1234-creer-un-espace-membres-en-php.html", qui ne comporte pas de numéro de page, les captures suivantes vont être obtenues :

Tableau 2.

Numéro de la capture	Texte capturé
0	sujet-1234-creer-un-espace-membres-en-php.html
1	1234
2	∅ (inutilisé : rien)

Numéro de la capture	Texte capturé
3	∅
4	∅
5	∅
6	∅
7	∅
8	∅
9	∅

Quand, pour "topic-473-4-installer-un-bundle-avec-composer.html", avec un numéro de page, ces captures donneront :

Tableau 3.

Numéro de la capture	Texte capturé
0	topic-473-4-installer-un-bundle-avec-composer.html
1	473
2	4
3	∅
4	∅
5	∅
6	∅
7	∅
8	∅
9	∅

Aux 9 captures possibles, s'en ajoute automatiquement toujours une spéciale numérotée 0, dont le but est de reprendre la sous-chaîne qui satisfait l'intégralité de l'expression régulière. Nous verrons par la suite comment exploiter concrètement ces captures.

Avant de poursuivre, évoquons à cette occasion le piège de l'alternative (|), en rappelant que son sens est "ce qui précède la barre verticale ou ce qui la suit". Ainsi, l'expression rationnelle `^abc|def$` ne signifie pas la chaîne (exacte) "abc" ou "def" mais bien une chaîne qui commence par "abc" ou qui se termine par "def". En effet, chacun des ancrages (^ et $) est propre à la sous-partie de l'alternative où il figure. Elle équivaut à `(^abc)|(def$)`. Pour écrire, je veux la chaîne (exacte) "abc" ou "def", il aurait fallu écrire : `^(abc|def)$`. Autre exemple commun : je veux désigner un fichier d'extension .html ou .php, on écrira `\.(html|php)$` et non `\.html|\.php$` où l'ancre $ ne s'applique qu'à la partie \.php.

2.7. Les assertions négatives

Bien que les assertions soient peu souvent utilisées au sein de la réécriture, je tenais à évoquer le sujet pour au moins signaler leur existence. Une assertion négative consiste à spécifier l'absence d'une sous-chaîne avant ou après la position qu'elle occupe.

- (?! ...) : assertion négative avant, possède pour signification "n'est pas suivi de". Exemple : ^articles/(?!fr/) : un chemin commençant par "articles/" et qui n'est pas suivi de "fr/".
- (?<! ...) : assertion négative arrière, a sens de "n'est pas précédé de". Exemple : (?<!^www)\.mondomaine\.fr$: un sous-domaine quelconque de mondomaine.fr excepté (précisément) www.mondomaine.fr.

Je dois avouer qu'elles ne sont que très rarement employées et qu'elles peuvent être remplacées par une règle de non réécriture préalable. Pour illustrer admettons une réécriture ayant pour but d'avoir des

URL fictives du type article/<id>/<titre>.html renvoyant sur article.php?id=<id> mais que pour une raison quelconque, je veuille exclure l'article d'id 34 de cette réécriture, il est possible d'écrire au choix :

```
RewriteRule ^article/(?!34/)(\d+)/.+\.html$ article.php?
id=$1 [L,QSA]
# ou
#RewriteRule ^article/(\d+)(?<!/34)/.+\.html$
article.php?id=$1 [L,QSA]

# Est équivalent à :
# On ne fait rien pour id=34
RewriteRule ^article/34/.+\.html$ - [L]
# Pour tous les autres, on procède à la réécriture
RewriteRule ^article/(\d+)/.+\.html$ article.php?id=$1
[L,QSA]
```

2.8. Épilogue

En combinant ces différents métacaractères, il est possible de décrire bien des choses évoluées suivant un format prédéfini. Quelques exemples :

- Un nombre entier, positif ou négatif, zéros non significatifs autorisés :

```
^[-+]?\d+$
```

- Un nombre quelconque :

```
^[-+]?\d+(?:\.\d+)?$
```

- Une adresse IPv4 :

```
^(?:(?:25[0-5]|2[0-4][0-9]|1\d{2}|[1-9]?\d)\.){3}
(?:25[0-5]|2[0-4][0-9]|1\d{2}|[1-9]?\d)$
```

- etc

3. Partie théorique : activation et explication des directives

3.1. Activation

3.1.1. Pour l'administrateur du serveur

Prétendre à l'usage de la réécriture d'URL implique avant tout d'activer le module, dédié, correspondant qui se nomme mod_rewrite. Éditez votre fichier de configuration d'Apache, usuellement appelé httpd.conf, pour vous assurer que ledit module (dynamique) est chargé par une directive LoadModule (décommentée) :

```
LoadModule rewrite_module modules/mod_rewrite.so
```

Note

- ceux qui auraient compilé Apache à la main et inclus mod_rewrite en module statique n'ont bien évidemment pas besoin de cette ligne, ce module sera (toujours) présent
- les utilisateurs de distribution GNU/Linux Debian et ses dérivées (comme *buntu) sont invités à utiliser la commande **a2enmod** à la place :

  ```
  a2enmod rewrite
  ```

Maintenez ouvert dans votre éditeur le fichier de configuration d'Apache car l'activer ne suffit pas si vous voulez faire usage de la réécriture depuis des fichiers .htaccess. En effet, vous devez vous assurer qu'Apache vous le permet. Ceci requiert en effet des droits.

- D'une manière générale, chaque directive se voit associer une catégorie (à la compilation du module) parmi cinq (`AuthConfig`, `FileInfo`, `Indexes`, `Limit` et `Options`) suivant la fonction qu'elle remplit. Ceci permet éventuellement à l'administrateur de restreindre, via la directive AllowOverride, à ses utilisateurs celles qui peuvent être ou non utilisées dans un fichier .htaccess par la suite. Les directives de réécriture font toutes partie de la catégorie nommée `FileInfo`.
- Cependant, la réécriture est volontairement bloquée de manière artificielle par un mécanisme supplémentaire de façon à pouvoir autoriser l'ensemble des directives de type `FileInfo` sauf la réécriture pour raison de sécurité. En plus d'autoriser les directives qualifiées `FileInfo`, il faut autoriser Apache à suivre les liens symboliques, par l'option `FollowSymLinks` ou `SymLinksIfOwnerMatch`.

La seconde, autoriser Apache à suivre les liens symboliques, vous laisse deux options :

- vous activez globalement, directement depuis le fichier de configuration d'Apache, à ce que les liens symboliques soient suivis en modifiant la ligne Options du ou des répertoires concernés pour y ajouter `FollowSymLinks` ou `SymLinksIfOwnerMatch`. Dès lors, l'utilisateur n'a rien à faire de particulier dans son fichier .htaccess.

  Pour résumer, cette solution implique deux modifications dans le fichier de configuration d'Apache : s'assurer que les utilisateurs puissent recourir à la réécriture (AllowOverride à valeur `FileInfo` au minimum) et Apache doit pouvoir suivre les liens (Options `+FollowSymLinks` ou similaire). En clair :

  ```
  <Directory C:/AMP/www/>
      # Listing de répertoire en l'absence de fichiers d'index
      # Autorisation de suivre les liens symboliques (requis pour la réécriture
  depuis les fichiers .htaccess)
      Options Indexes SymLinksIfOwnerMatch
  ```

```
    # On permet à l'utilisateur de définir toute directive sauf celles de catégorie
Options
    # C'est FileInfo qui permet la réécriture
    AllowOverride AuthConfig FileInfo Indexes Limit

    # Pour permettre l'accès de tous au répertoire (je le mets mais ça n'a rien à
voir avec la réécriture)
    # Apache < 2.4
    <IfModule !mod_authz_core.c>
        Allow from all
    </IfModule>
    # Apache >= 2.4
    <IfModule mod_authz_core.c>
        Require all granted
    </IfModule>
</Directory>
```

- ou, ma préférence, avoir confiance en l'utilisateur en lui laissant la possibilité de tout utiliser. Ainsi, il activerait lui-même le suivi des liens symboliques, ce qui demande alors à leur accorder plus de droits : il faut alors leur donner `Options` en plus de `FileInfo`. Autant dire tout au final : votre directive AllowOverride doit alors passer à valeur `All` mais vous n'avez pas besoin de modifier la partie Options, ce sera à eux de l'ajouter dans leur .htaccess. Pour résumer, le fichier de configuration dans ce cas de figure se présente plutôt ainsi :

```
<Directory C:/AMP/www/>
    # Listing de répertoire en l'absence de fichiers d'index
    # Il reviendra à l'utilisateur d'activer le suivi des liens symboliques
nécessaire à la réécriture
    Options Indexes

    # On permet à l'utilisateur d'utiliser toute directive possible
    AllowOverride All

    # Pour permettre l'accès de tous au répertoire (je le mets mais ça n'a rien à
voir avec la réécriture)
    # Apache < 2.4
    <IfModule !mod_authz_core.c>
        Allow from all
    </IfModule>
    # Apache >= 2.4
    <IfModule mod_authz_core.c>
        Require all granted
    </IfModule>
</Directory>
```

Quelle méthode choisir ? Il n'y en a pas une meilleure que l'autre, ce qui compte ce sont les limites que vous voulez, ou non, imposer et quelle confiance vous accordez à vos utilisateurs. La première, du moins telle que je la vois, présente surtout l'inconvénient de bloquer l'usage des directives de type `Options` à l'usager (si PHP fonctionne en module, vous bloqueriez notamment les directives php_flag et php_value). Est-ce vraiment ce que vous voulez ? Quand la seconde place une plus grande confiance en ceux-ci et leur offre une plus grande liberté.

Avertissement

Si vous ne savez pas ce que vous faites, ne modifiez jamais la partie <Directory />. Elle correspond à la racine du système de fichiers et doit faire office de politique restrictive par défaut !

La ou les parties que vous devez modifier sont celles qui sont publiées donc qui correspondent à votre racine (DocumentRoot) ou plus bas dans l'arborescence (à l'exception des alias). En général, modifier les balises <Directory> qui reprennent les valeurs des DocumentRoot suffit.

Note

Sur la plupart des distributions GNU/Linux et pour une installation d'Apache avec les paquets officiels, la configuration d'Apache est découpée en plusieurs fichiers. Si tel est le cas, c'est le fichier de configuration qui correspond à votre hôte virtuel qu'il faut chercher. Quelques exemples :

- Debian/*buntu : `/etc/apache2/sites-enabled/000-default`
- Gentoo : `/etc/apache2/vhosts.d/default_vhost.include`

À moins d'attendre le (re)démarrage de votre machine, il vous est nécessaire de recharger la configuration d'Apache ou de relancer Apache.

- Sous GNU/Linux, la commande ressemble généralement à :

```
# initscripts
/etc/init.d/apache2 reload
# systemd
systemctl reload apache2
```

- FreeBSD, pour Apache 2.2 :

```
service apache22 restart
# ou
/usr/local/etc/rc.d/apache22 restart
```

3.1.2. En tant qu'utilisateur

Il est nécessaire, partout où vous aurez des règles de réécriture, d'activer explicitement la réécriture par une directive :

```
RewriteEngine on
```

En effet, il ne suffit pas de placer des règles dans un fichier .htaccess, ou même dans le fichier de configuration d'Apache, pour qu'elles soient opérantes. Ceci pour des raisons de performance tout comme cela peut vous permettre de désactiver temporairement votre réécriture.

Bien qu'il soit possible d'activer globalement la réécriture par une directive RewriteEngine `On` au niveau de chaque hôte (VirtualHost) pour ne pas avoir besoin de la repréciser partout, je vous le déconseille fortement de façon à ne pas en subir inutilement le contre-coût. C'est pourquoi, ici, elle sera systématiquement accolée à nos règles.

3.2. Principe général

Avant de savoir comment déclarer ses règles au niveau d'Apache, il est vital de bien assimiler ce qu'est une URL et, surtout, de quoi elle se constitue. Une URL est en effet composée de différentes parties. Pour les détailler, prenons en modèle une URL particulièrement complète :

```
https://www.monsite.fr:8443/forum/admin/task.php?page=db&start=30#p71
```

Elle se décompose comme suit :

1. le nom du protocole qui précède les "://" en tête. En principe, ici, il ne peut qu'être à valeur “http” ou “https” (la version sécurisée - encapsulée par SSL - du protocole HTTP comme dans l'exemple). Il définit la manière dont client et serveur communiquent ;
2. vient ensuite, le nom (ou l'adresse IP directement) (www.monsite.fr dans l'exemple) du serveur à qui le client s'adresse ;

3. optionnellement, si le serveur écoute sur un port non standard par rapport au protocole (80 pour http et 443 pour https), le client devra l'indiquer directement après le nom ou l'adresse du serveur en ajoutant entre les deux un caractère ':'. C'est le cas de notre exemple, avec pour port TCP : 8443 ;
4. puis on trouve la partie la plus importante : le chemin (/forum/admin/task.php dans l'exemple). Elle désigne la ressource que le client veut consulter. Pour schématiser, il faut voir le serveur HTTP comme un disque dur distant où ce chemin indiquerait le nom complet du fichier que vous désirez lire ;
5. éventuellement, on peut ensuite trouver ce que l'on appelle la query string ou, en français, la chaîne de requête. Quand elle est présente, elle est séparée du chemin par un point d'interrogation (?). Elle consiste à fournir des informations sous la forme de couples nom=valeur facultative où chaque paramètre est séparé par un caractère &. Dans notre exemple il s'agit de la partie page=db&start=30 qui correspond à deux paramètres : *page* de valeur *db* et *start* de valeur *30*. Si la ressource invoquée le prévoit, cette page traite ainsi ces données, ce qui peut lui permettre d'avoir un caractère dynamique. Elle est très couramment employée par les langages scripts tel PHP, pour, par exemple, avoir un script unique gérant l'affichage d'une fiche produit plutôt qu'un fichier HTML à maintenir par article : l'identifiant du produit ciblé est passé en query string pour que ce script génère dynamiquement sa description à partir des données de la base de données.
6. en fin d'URL, précédée d'un caractère dièse (#), on peut potentiellement trouver ce que l'on appelle une ancre. Elle désigne, par un nom (p71 ici), une position verticale au sein d'une ressource vers laquelle vous êtes automatiquement déplacé. Cela vous évite de chercher et de scroller si le document est long.

   De fait, notez que cette dernière partie de l'URL n'a de sens que pour un client : le serveur n'aurait que faire d'un emplacement dans une page. **Le client n'envoie pas une ancre au serveur (ce morceau est supprimé de l'URL effectivement transmise au serveur HTTP).** En revanche, il peut être tout à fait pertinent, pour le serveur, d'en fixer une au client.

3.3. Travailler uniquement sur le chemin HTTP

Dans un premier temps nous n'allons considérer et travailler que sur la partie chemin de l'URL. Nous ignorons tout le reste, les protocole, adresse, port et query string.

Tout se joue sur la directive RewriteRule dont je vous présente la forme ci-dessous :

```
RewriteRule <motif> <destination> ([<options>])
```

1. *motif* : une expression régulière que la partie chemin doit satisfaire pour voir la règle appliquée.

   Il faut cependant noter une subtilité par rapport aux chemins suivant l'emplacement de vos directives RewriteRule :

   - dans le(s) fichier(s) de configuration d'Apache, excepté les parties <Directory>, le chemin à partir duquel RewriteRule travaille est le chemin **complet** de la ressource, **qui commence toujours par un slash**. Par exemple, pour des règles directement située dans le fichier de configuration d'Apache et l'URL http://www.mondomaine.ext/forum/profil-3-toto :

     ```
     RewriteRule ^forum/profil-(\d+)-.+ viewprofil.php?id=$1
     ```

     Ne serait jamais appliquée du fait de l'ancrage sur le début de la chaîne et le manque du slash au début.
   - à l'inverse, pour des règles situées dans un fichier .htaccess ou des blocs <Directory>, le chemin sur lequel RewriteRule se base est **relatif** au répertoire du fichier .htaccess. C'est-à-dire que toute la partie qui correspond au répertoire où se trouve le fichier .htaccess est d'abord automatiquement tronquée par Apache. En conséquence, **le chemin ne commence jamais par un slash**.

Prenons l'URL http://www.mondomaine.ext/a/b/c/d avec un fichier .htaccess dans le sous-répertoire b, je dois alors ignorer la partie /a/b/ (dernier slash compris) soit employer ^c/d$ comme motif de ma règle :

```
RewriteRule ^c/d$ c/e [L,R=permanent]
```

2. *destination* :
   - un chemin HTTP absolu (/rss.xml : le fichier rss.xml à la racine du site) ou relatif (forum/topic.php : bien que pas tout à fait exact, le fichier topic.php du sous-répertoire forum du répertoire où se situe le fichier .htaccess) : le chemin de la ressource à atteindre au travers du serveur HTTP.
   - une URL complète (http://www.google.fr/) : une redirection HTTP (temporaire, par défaut) aura lieu vers la nouvelle destination. **Comme il ne s'agit plus du même hôte/domaine, c'est bien une redirection HTTP qui est opérée. Celle-ci est suivie par le client, ce dernier verra son URL changer.**
   - un chemin vers un fichier (tel C:/tmp/maintenance.html), uniquement possible depuis les fichiers de configuration d'Apache, sachant que les droits d'accès sont honorés (par rapport aux directives Allow/Deny, ou Require en versions 2.4, en application).
   - ne rien faire par la valeur spéciale tiret (-). L'URL reste alors inchangée. Comme nous pourrons le voir plus bas, cette valeur est liée à l'usage de certaines options mais permet aussi d'écrire des règles de non-réécriture afin de réaliser des exceptions.
3. *options* : une liste facultative d'options, parmi celles détaillées ci-dessous. Ces options sont encadrées par des crochets et séparées par des virgules. Exemple : [NC,F] pour les options NC, insensibilité à la casse et F, retour d'un code HTTP d'erreur 403.

Avertissement

Veillez à ne pas ajouter d'espace après la virgule sans quoi vous obtiendriez une erreur 500. En effet, un espace marque le passage à un nouvel argument des directives Apache, ce qui rendrait votre règle invalide.

Tableau 4. Les différentes options possibles pour RewriteRule

Abréviation	Nom long	Description	Note
F	forbidden	Interdit virtuellement l'accès à une ressource en renvoyant un statut HTTP "403 Forbidden"	◦ Il n'y a pas lieu de modifier la requête, le deuxième paramètre devrait être à valeur "-" ◦ La lecture du .htaccess est interrompue (l'option Last est implicite) ◦ F est synonyme de R=403 à partir des versions 2.2 ◦ Suivant le contexte, une telle règle peut éventuellement être remplacée par une directive Deny from all (ou Require all denied à partir de la version 2.4.0)

Abréviation	Nom long	Description	Note
G	gone	Indique virtuellement que toute ressource correspondant au motif n'existe plus par un statut HTTP "410 Gone"	◦ Il n'y a pas lieu de modifier la requête, le deuxième paramètre devrait être à valeur "-" ◦ La lecture du .htaccess est interrompue (l'option Last est implicite) ◦ G est synonyme de R=410 à partir des versions 2.2 ◦ Une telle règle peut être remplacée par Redirect ou RedirectMatch
R(=code)	redirect	À l'origine, cette option est prévue pour effectuer une redirection HTTP temporaire ou permanente. Depuis les versions 2.2, il est possible de renvoyer d'autres codes HTTP (403, 404, etc).	Valeurs permises pour code : ◦ R=temp ou R=302 : procède à une redirection temporaire. Il s'agit là du comportement par défaut de cette option en l'absence de toute valeur de code (voir aussi les directives Redirect) ◦ R=permanent ou R=301 : réalise une redirection permanente (voir aussi les directives Redirect) ◦ R=seeother ou R=303 (voir aussi les directives Redirect) : indique que la réponse se trouve à une autre adresse (en méthode GET) ◦ R=un code HTTP numérique valide : permet de renvoyer le code d'erreur HTTP indiqué. Par exemple, avec R=404, nous pourrions faire passer une partie de l'arborescence comme inexistante aux clients Note Le drapeau R n'induit pas implicitement Last dans tous les cas, c'est pourquoi il est vivement recommandé de le préciser sous peine de rencontrer une possible erreur sur les autres règles.
NC	nocase	Rend l'ensemble du motif de la directive RewriteRule courante insensible à la casse	Rappel : elle ne concerne que les caractères de l'ASCII non étendu
NE	noescape	Par défaut, Apache encode tout caractère pouvant s'avérer problématique. Si vous voulez conserver vos caractères tels que vous les avez écrit, ajoutez cette option.	Cette option est notamment requise pour faire figurer une ancre dans la partie destination sans quoi le dièse est encodé en %23. Ce qui conduit à une erreur 404.
QSA	qsappend	Reproduire la chaîne de requête (query string) précédente à la fin du nouveau chemin à suivre	

Abréviation	Nom long	Description	Note
QSD	qsdiscard	Ne pas recopier la chaîne de requête (query string) avant la réécriture courante	Nécessite Apache >= 2.4.0. Il existe cependant un équivalent portable consistant à ajouter un point d'interrogation à la fin du chemin de destination. Il indique à Apache de ne pas recopier la query string (et ce point d'interrogation sera supprimé de la requête).
L	last	Par défaut, Apache continue la lecture des règles après en avoir trouvé une première qui s'applique (c'est alors la nouvelle URL qui doit matcher les RewriteRule restantes). Pour qu'il l'applique immédiatement, sans poursuivre sa lecture, ajouter cette option.	N'espérez pas échapper à une boucle infinie de réécriture avec cette option. En effet, cette option ne stoppe que le processus courant or lors de l'application de toute règle, la nouvelle URL subit à son tour toute éventuelle règle de réécriture suivant où elle aboutit. Lorsqu'il y a une boucle, si vous revenez sur le même répertoire, donc les mêmes règles, le flag Last ne vous sauvera pas.
E	end	Met fin à tout processus de réécriture.	Requiert Apache >= 2.4.0

Voici quelques premiers exemples simples :

- Embellir les URL de mon blog de façon à les produire pour qu'elles aient pour forme : article-<id>-<titre>.html et qu'en réalité elles aboutissent, comme avant, sur article.php?id=<id> :

```
RewriteRule ^article-(\d+)-.+\.html$ article.php?id=$1
```

- Renvoyer tout ce qui n'a pas d'extension et qui n'est pas un répertoire, au sens où le chemin n'est pas terminé par un slash, en query string, en tant que valeur du paramètre page, sur le script index.php :

```
RewriteRule ^[a-z]+$ index.php?page=$0
```

- Faire passer ses scripts PHP pour de bêtes fichiers HTML statiques (ne pas oublier de changer l'extension des fichiers en .html dans vos différents liens - balises <a> notamment) :

```
RewriteRule (.*)\.html$ $1.php
```

3.4. Interactions entre les règles

Je souhaite, dans cette partie, vous expliquer de manière simplifiée, en mettant volontairement de côté certains aspects techniques qui seront abordés bien plus tard, comment Apache lit et applique les règles pour un fichier .htaccess donné. C'est un point essentiel à la rédaction de vos règles : il est important de comprendre comment et en quoi les règles interagissent. Interactions qui, concrètement, peuvent se traduire à voir certaines règles invoquées au détriment d'autres alors que ce n'est pas ce qui était voulu.

Avant tout, même si cela peut paraître bête : les règles de réécriture sont évaluées de haut en bas : la première à apparaître dans le fichier puis la deuxième puis la troisième et ainsi de suite. Par conséquent :

- même s'il ne s'agit que d'une question d'expressions régulières et de logique, vous ne pouvez pas avoir deux motifs strictement identiques ou très proches. J'ai déjà vu des règles comme celles-ci :

```
# Cet exemple est volontairement erroné pour des fins pédagogiques
RewriteRule ^([0-9]+)-([0-9]+)-(.*)\.html$ index.php?menu=$1&rubrique=$2 [L]
RewriteRule ^([0-9]+)-([0-9]+)-(.*)\.html$ index.php?menu=$1&page=$2 [L]
```

Comment Apache, ou même quiconque, pourrait-il distinguer les deux cas ? Ce qu'il va faire, conformément à ce que je viens d'écrire (et si le chemin correspond bien évidemment), c'est toujours appliquer la première. Du coup, la seconde, ne sera **jamais** utilisée.

- du fait de ce sens de lecture, les règles doivent être placées par ordre de spécificité décroissante, celles aux motifs les moins larges en haut de sorte qu'elles aient priorité. Illustration avec un nouvel exemple à ne pas suivre :

```
# Cet exemple est volontairement erroné pour des fins pédagogiques
RewriteRule ^viewtopic-([0-9]+).* viewtopic.php?id=$1 [L]
RewriteRule ^viewtopic-([0-9]+)-([0-9]+).* viewtopic.php?id=$1&p=$2 [L]
```

Qu'est-ce qui ne va pas ici ? La seconde règle est plus précise puisqu'elle attend un chemin composé de "viewtopic-", un nombre, un tiret puis un nombre enfin, optionnellement, n'importe quoi quand la première est satisfaite par un chemin constitué de "viewtopic-", un nombre enfin, optionnellement, n'importe quoi. La sous-partie .* de la première règle est en conflit avec la portion -([0-9]+).* de la seconde du fait que .* inclut -([0-9]+).*. Conséquence : pour les chemins auxquels seraient applicables la seconde règle, c'est la première qui se verrait invoquée ; la seconde ne le serait jamais non plus.

Comment résoudre ce conflit ? J'ai déjà donné la solution en réalité : elles doivent tout simplement être inversées pour respecter cette notion de priorité par rapport à la spécificité respective de leur expression régulière.

```
RewriteRule ^viewtopic-([0-9]+)-([0-9]+).* viewtopic.php?id=$1&p=$2 [L]
RewriteRule ^viewtopic-([0-9]+).* viewtopic.php?id=$1 [L]
```

Un autre exemple où elles doivent être inversées :

```
# Cet exemple est volontairement erroné pour des fins pédagogiques
RewriteRule ([^/]+)/([^/+]) index.php?param1=$1&param2=$2 [L,QSA]
RewriteRule ([^/]+)/([^/+])/([^/+]) index.php?param1=$1&param2=$2&param3=$3 [L,QSA]
```

La première étant satisfaite par la présence du moindre slash.

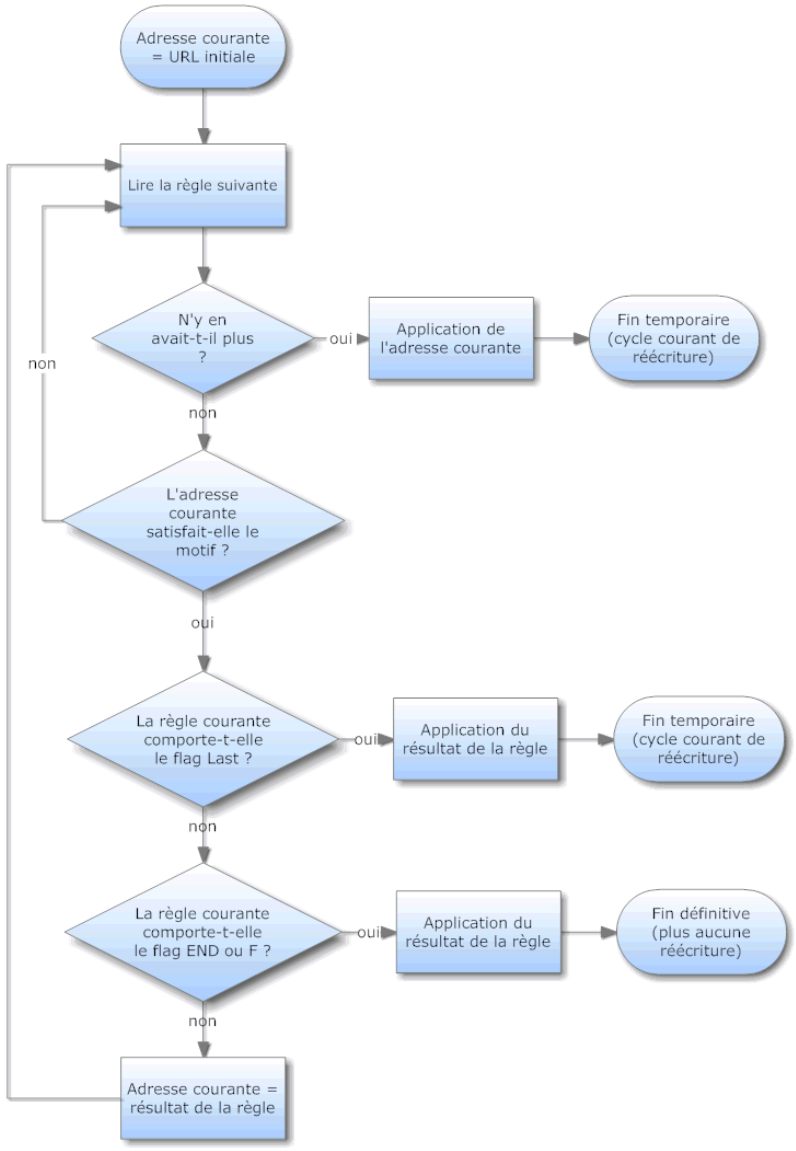
Il arrive parfois que ce soit l'omission d'ancrages (^ et/ou $) qui conduise à un tel conflit. Dans ce dernier exemple, un ancrage, suivant le but recherché, pourrait même s'avérer plus approprié.

Autre point à éventuellement prendre en compte, la présence ou l'absence de drapeaux spécifiques. Pour schématiser, dès la réception d'une requête HTTP, Apache va initialiser une variable en mémoire que j'appelle "adresse courante" à partir de l'URL initialement appelée. Elle sert de référence aux comparaisons avec les règles tout comme elle peut déterminer l'adresse à suivre. La réécriture peut en modifier la valeur. Ainsi quand une règle est satisfaite, certaines options peuvent influer de manière totalement différente sur le déroulement de la réécriture :

- END ou F(orbidden) : Apache stoppe sa lecture des règles, le résultat de la règle (la destination, deuxième paramètre de RewriteRule) entre en vigueur immédiatement sans que la nouvelle adresse obtenue puisse subir par la suite une quelconque réécriture (fin définitive).
- L(ast) : Apache ne poursuit pas la lecture des règles, la règle courante est appliquée, l'adresse obtenue de cette règle est suivie/invoquée mais cette dernière est à son tour éligible à toute réécriture. En d'autres termes, pour Apache, ce résultat de réécriture n'apparaît pas comme tel mais bien comme toute requête HTTP. Nous reviendrons sur ce drapeau particulier par la suite.
- À défaut, sans aucun de ceux-ci : Apache va tout de même continuer à lire les autres règles, l'adresse courante, servant de base à la réécriture, devient le résultat de la règle courante. C'est cette nouvelle adresse qui va servir de base à la comparaison des règles suivantes.

  Si jamais la fin des règles est atteinte (la fin du fichier .htaccess par exemple), qu'il y ait eu ou non une précédente réécriture, l'adresse courante est invoquée.

Ci-dessous une représentation graphique illustrant le traitement des règles :



3.5. Les conditions pour travailler sur les autres parties de l'URL et au-delà

Nous avons vu plus tôt qu'une directive RewriteRule ne s'applique qu'à la partie chemin d'une URL. N'est-ce pas contraignant ? Non parce que le module de réécriture prévoit une directive complémentaire, RewriteCond, pour gérer tout le reste, autres parties d'une URL (nom de domaine, port, query string, etc) comprises. RewriteCond a pour rôle de définir une condition (telle une instruction if dans un langage de programmation quelconque), restreignant ainsi l'effet de la règle, directive RewriteRule, qui la suit et à laquelle elle est couplée.

```
RewriteCond <chaîne à tester> <condition> (<options>)
```

1. *chaîne à tester* : la valeur à tester consiste en un élément interne à Apache décrivant notamment la requête HTTP courante comme le nom du serveur que le client cherche à atteindre, la valeur d'un entête HTTP, la chaîne de requête, etc. Tous ceux-ci sont représentés par une variable nommée prédéterminée de la

forme %{nom} dont vous pourrez trouver une liste dans la partie suivante avec leur description respective.

2. *condition* : la condition que *chaîne à tester* doit satisfaire pour que la règle soit appliquée (si Apache évalue la partie RewriteCond d'une règle, le chemin, par rapport à RewriteRule, correspond déjà). Cette partie se constitue d'un opérateur et éventuellement d'une valeur accolée à ce dernier (sans espace). Par défaut, l'opérateur implicite est la satisfaction d'une expression régulière et la partie valeur est traitée comme une expression régulière.
3. *options* : dans la même veine que RewriteRule, une liste facultative d'options, entre crochets et séparées par une virgule (sans espaces), parmi :

   Tableau 5. Les options de la directive RewriteCond

Abréviation	Nom long	Description	Note
NC	nocase	Rend l'ensemble du motif de la directive RewriteCond courante insensible à la casse	Rappel : elle ne concerne que les caractères de l'ASCII non étendu
OR	ornext	Unit la condition courante à la suivante par un OU logique au lieu d'un ET	

En d'autres termes, une directive RewriteCond n'a pas de sens seule, elle est toujours associée à une RewriteRule, celle qui suit. Cela signifie également qu'une directive RewriteCond n'est valable que pour une seule règle, ne comptez pas les factoriser, il vous faudra les répéter. Démonstration : si vous voulez forcer l'usage du protocole HTTPS pour deux scripts, login.php et register.php, vous ne pouvez pas écrire :

```
RewriteCond %{HTTPS} !=on [NC]
RewriteRule ^login\.php$ https://%{HTTP_HOST}/$0

RewriteRule ^register\.php$ https://%{HTTP_HOST}/$0
```

C'est faux, cela produit une boucle de réécriture sur register.php en https, car RewriteCond vaut bien pour la première RewriteRule mais pas pour la seconde. La correction est de reproduire la condition pour les deux règles :

```
RewriteCond %{HTTPS} !=on [NC]
RewriteRule ^login\.php$ https://%{HTTP_HOST}/$0

RewriteCond %{HTTPS} !=on [NC]
RewriteRule ^register\.php$ https://%{HTTP_HOST}/$0
```

Jusqu'ici j'ai employé le singulier en évoquant le couple condition (RewriteCond)/règle (RewriteRule) or, pour reprendre ma précédente analogie, tel un programme informatique, une même instruction peut dépendre de plusieurs conditions et non d'une seule. Il est parfaitement possible d'assigner plusieurs conditions, à raison d'une par ligne, par autant de directives RewriteCond, à une même règle. Pour illustrer, admettons que je veuille interdire mon site à deux adresses IP, 80.10.250.23 et 81.71.12.78, j'écrirai :

```
# Cet exemple est faux
# L'adresse IP courante du client est représentée par la variable %{REMOTE_ADDR} -
ces variables sont détaillées dans la partie suivante
RewriteCond %{REMOTE_ADDR} =80.10.250.23
RewriteCond %{REMOTE_ADDR} =81.71.12.78
RewriteRule .* - [F]
```

Cependant, il faut noter que, par défaut, l'ensemble des conditions d'une règle donnée sont liées par un et logique. La règle ci-dessus ne peut jamais être satisfaite : le client ne peut avoir à la fois l'adresse IP 80.10.250.23 et, en même temps, 81.71.12.78. Nous avons besoin d'un ou logique à la place (le client a l'adresse IP

80.10.250.23 ou 81.71.12.78) soit introduire l'option [OR], vue ci-dessous, à la fin de la première condition. Notre réécriture, après correction, devient :

```
RewriteCond %{REMOTE_ADDR} =80.10.250.23 [OR]
RewriteCond %{REMOTE_ADDR} =81.71.12.78
RewriteRule .* - [F]

# Une autre façon de l'écrire
# Apache < 2.4
<IfModule !mod_authz_core.c>
    Order Deny,Allow
    Deny from 80.10.250.23 81.71.12.78
</IfModule>
# Apache >= 2.4
<IfModule mod_authz_core.c>
    <RequireAll>
        <RequireNone>
            Require ip 80.10.250.23 81.71.12.78
        </RequireNone>
    </RequireAll>
</IfModule>
```

Vous pourrez trouver ci-dessous l'ensemble des opérateurs s'appliquant à RewriteCond. Ils sont scindés en deux groupes : le premier a pour but de tester le premier paramètre par rapport à une valeur (chaîne, entier, expression régulière) attendue juste derrière l'opérateur quand le second n'attend pas de telle valeur car ils testent la nature d'un fichier dont le nom est fourni en premier paramètre.

Tableau 6. Les différents opérateurs de comparaison

Opérateur	Description	Note
Aucun (absence d'opérateur)	Satisfaction de l'expression régulière	
!	Non satisfaction de l'expression régulière	
=	Égalité (littérale)	
!=	Inégalité (littérale) : différent de	
<	Strictement inférieur selon l'ordre lexicographique	
!<	Supérieur ou égal selon l'ordre lexicographique	Apache 2.4 a introduit la forme équivalente >= plus explicite
>	Strictement supérieur selon l'ordre lexicographique	
!>	Inférieur ou égal selon l'ordre lexicographique	Apache 2.4 a introduit la forme équivalente <= plus explicite

Opérateur	Description	Note
-eq	Égalité numérique	• Les deux chaînes sont converties en entiers (fonction C atoi) puis comparés. • Apache >= 2.4.0 mais non fonctionnel pour Apache < 2.4.3 pour cause d'implémentation erronée (pris pour une expression régulière)
-ne	Inégalité numérique	Mêmes remarques que pour l'opérateur -eq
-ge	Numériquement supérieur ou égal	Mêmes remarques que pour l'opérateur -eq
-gt	Strictement supérieur numériquement	Mêmes remarques que pour l'opérateur -eq
-le	Numériquement inférieur ou égal	Mêmes remarques que pour l'opérateur -eq
-lt	Strictement inférieur numériquement	Mêmes remarques que pour l'opérateur -eq

Avertissement

La notion d'ordre lexicographique est totalement fausse à cause d'une erreur d'implémentation dans la fonction strcmp qui a été réécrite au sein du module de réécriture : elle tient avant tout compte de la longueur des deux chaînes. De ce fait, si deux chaînes n'ont pas la même longueur, c'est la plus longue des deux qui est considérée comme supérieure sans même considérer leurs contenus. Pour illustrer, la comparaison de alexendra, brian et zephir donne, pour Apache : alexendra > zephir > brian (de la plus longue à la plus courte).

Seuls les opérateurs < et > sont impliqués, pas les (in)égalités, et deux chaînes de même longueur ne posent aucun soucis ce qui confère à ce bug un effet très limité et lui vaut d'avoir été aussi longtemps "ignoré". Cependant, il ne sera probablement pas corrigé de si tôt pour des raisons de compatibilité (il est volontairement conservé en versions 2.2 et 2.4 afin de garder le même comportement). En revanche, en versions 2.4, la syntaxe alternative à base d'expression (RewriteCond expr `condition`) ne le reprend pas, cette forme est bien correcte.

En réalité, elle aurait été correcte pour comparer des nombres entiers positifs sans zéro non significatif sous forme de chaînes. Mais ne comptez pas représenter, par exemple, un intervalle d'adresses IP avec ces opérateurs, du moins avec la forme standard de RewriteCond.

Tableau 7. Les opérateurs fichiers, qui peuvent également être niés en les précédant d'un !

Opérateur	Description
-d	Le fichier existe et désigne un répertoire
-f	Le fichier existe et correspond à un fichier régulier
-s	Le fichier existe, est un fichier régulier et possède une taille non nulle
-l	Le fichier existe et est un lien symbolique
-x	Le fichier existe et est exécutable

3.6. Les différentes variables de réécriture

Les "variables" prédéfinies par le module de réécriture prennent la forme %{`nom_de_la_variable`}. Toute variable inexistante se verra interpolée (= dynamiquement remplacée) par la chaîne vide sans l'émission d'une quelconque erreur.

Avertissement

Cette interpolation n'a lieu que pour le premier paramètre de RewriteCond et deuxième paramètre de RewriteRule. Toute variable figurant ailleurs qu'à ces emplacements ne seront pas substituées par leur valeur, elles seraient littérales.

Vous trouverez ci-dessous la liste de ces variables prédéfinies, regroupées par catégorie fonctionnelle :

Tableau 8. Les variables reprenant la configuration d'Apache

Nom	Description
%{SERVER_ADMIN}	Correspond à la valeur de la directive ServerAdmin attribuée au serveur
%{DOCUMENT_ROOT}	La racine du site, telle qu'elle est définie par la directive DocumentRoot
%{SERVER_NAME}	Le nom réel du serveur, la valeur de sa directive ServerName

Tableau 9. Les variables liées à la communication client/serveur (TCP/IP, réseau)

Nom	Description
%{SERVER_ADDR}	L'adresse IP du serveur traitant la demande
%{SERVER_PORT}	Le port TCP du serveur ayant reçu la requête HTTP (80 : port standard du protocole HTTP et 443 pour HTTPS)
%{REMOTE_ADDR}	L'adresse IP du client (ce client peut n'être qu'un intermédiaire - proxy)
%{REMOTE_HOST}	Le nom du client, ceci implique HostnameLookups à valeur On ou Double (attention aux conséquences) sinon elle n'existe pas
%{REMOTE_PORT}	Le port TCP du client employé pour l'émission de la requête

Tableau 10. Les variables propres au protocole HTTP

Nom	Description	Exemple	Note
%{HTTP_HOST}	Le nom du serveur tel que demandé par le client par l'intermédiaire de l'entête HTTP Host	www.monsite.fr	Identique à %{HTTP:Host}
%{REQUEST_URI}	La ressource telle que demandée dans la requête HTTP, vous aurez son chemin complet même dans un contexte dit "de répertoire"	/forum/admin/task.php	Voir également la variable %{THE_REQUEST}

Nom	Description	Exemple	Note
%{THE_REQUEST}	La ligne complète de la requête HTTP (sans les entêtes)	GET /forum/admin/task.php?page=db&start=30 HTTP/1.1	
%{HTTP_COOKIE}	Une chaîne regroupant l'ensemble des cookies que le client renvoie	skin=blue; pref_order=asc (deux cookies : skin de valeur blue et pref_order de valeur asc)	Un raccourci pour %{HTTP:Cookie}
%{HTTP_ACCEPT}	Les formats acceptés et préférences du client	text/xml,application/xml,application/xhtml+xml,text/html; q=0.9,text/plain; q=0.8,image/png,*/*; q=0.5	Même chose que %{HTTP:Accept}
%{HTTP_REFERER}	La provenance de l'utilisateur	-	Strict équivalent de %{HTTP:Referer}
%{QUERY_STRING}	Les paramètres passés dans l'URL, regroupés sous la forme d'une chaîne. Elle représente toute la partie située après le point d'interrogation (exclus), si tant est qu'il y en est une	page=db&start=30	
%{REQUEST_METHOD}	La méthode HTTP employée par la requête (généralement GET ou POST)	GET	
%{HTTP_USER_AGENT}	L'identifiant du client	Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11	Synonyme de %{HTTP:User-Agent}
%{SCRIPT_FILENAME} et %{REQUEST_FILENAME}	Chemin complet (absolu) du document appelé	/usr/local/www/apache22/data/forum/admin/task.php	
%{SERVER_PROTOCOL}	La description du protocole employé	HTTP/1.1	
%{HTTP:`nom d'un entête HTTP`}	La valeur de l'entête HTTP pour le nom donné	-	

Tableau 11. Les variables temps, date/heure du serveur

Nom	Description

Nom	Description
%{TIME_YEAR}	L'année sur 4 chiffres
%{TIME_MON}	Le mois sur 2 chiffres (de 01 à 12)
%{TIME_DAY}	Le jour du moins sur 2 chiffres (avec un zéro initial si besoin)
%{TIME_WDAY}	Le jour de la semaine au format numérique : 0 pour dimanche, 1 pour lundi, ... à 6 pour samedi
%{TIME_HOUR}	L'heure, au format 24h, avec les zéros initiaux (de 00 à 23)
%{TIME_MIN}	Les minutes sur 2 chiffres (de 00 à 59)
%{TIME_SEC}	Les secondes sur 2 chiffres (de 00 à 59)
%{TIME}	Équivalent abrégé de %{TIME_YEAR}%{TIME_MON}%{TIME_DAY}%{TIME_HOUR}%{TIME_MIN}%{TIME_SEC}, la concaténation des année, mois, jour, heures, minutes, secondes des date/heure courantes du serveur

Tableau 12. Les variables héritées du module mod_ssl lorsque le protocole est HTTPS

Nom	Description
%{HTTPS}	Booléen (valeur "on" ou "off") indiquant l'usage ou non du protocole sécurisé (peut être employée indépendamment de la présence ou non du module ssl)
%{SSL:`nom variable SSL`}	La valeur de la variable SSL désignée. Voir la liste dans la documentation d'Apache

Tableau 13. Accès aux variables d'environnement

Nom	Description
%{ENV:`nom variable d'environnement`}	La valeur de la variable d'environnement pointée (ou la chaîne vide si inexistante). Attention : vous ne pourrez pas atteindre les variables d'environnement déclarées par SetEnv car elles sont créées après. Au besoin, pour lever cette limitation, remplacez vos directives Setenv par SetEnvIf(NoCase) en la couplant à un motif qui sera toujours satisfait (^ ou .* notamment)

Tableau 14. Références arrières

Nom	Description
$X avec X tel que [1;9]	Référence arrière pour la X^{ième} parenthèse capturante de la directive RewriteRule courante. Toutes les références arrières (`$0` comprise) sont indisponibles pour une règle RewriteRule dont le motif est nié
$0	Référence arrière spéciale toujours disponible mémorisant la sous-chaîne satisfaisant l'ensemble de l'expression régulière de la directive RewriteRule courante
%X avec X tel que [1;9]	Référence arrière pour la X^{ième} parenthèse capturante de la dernière directive RewriteCond pour la règle courante
%0	Référence arrière spéciale toujours disponible mémorisant la sous-chaîne satisfaisant l'ensemble de l'expression régulière de la dernière directive RewriteCond pour la règle courante

4. Quelques exemples d'applications de la réécriture

4.1. Interdire l'accès direct aux images depuis un site extérieur (direct linking ou hotlinking) ?

En théorie, quand un client HTTP demande une ressource, il envoie dans le même temps un entête HTTP nommé Referer indiquant sa page d'origine. Ainsi, il serait possible de bloquer voire remplacer le chargement de toute image que vous hébergez depuis un autre site.

En pratique, étant donné que c'est le client qui fournit cette information, il est libre de l'omettre ou de la falsifier, que ce soit volontaire ou non.

Malgré le peu de crédit qu'il faut accorder à cet entête, réalisons tout de même cette tâche avec la réécriture pour un but didactique :

- la règle ne doit s'appliquer qu'aux images, noms se terminant par .gif, .png, .jpeg, etc soit répondant au motif `\.(?:gif|jpe?g|png)$`. Il est possible d'ajouter l'option NC pour que ces extensions soient insensibles à la casse ;
- bien que nous pourrions remplacer les images par d'autres, nous nous contentons d'interdire leur consultation en renvoyant une erreur HTTP de code 403. Par conséquent, nous n'avons pas besoin de modifier l'URL : le deuxième paramètre doit être le tiret (-) et nous ajoutons l'option F à RewriteRule ;
- Viennent les conditions (directives RewriteCond) sur la valeur de l'entête HTTP Referer représentée par la variable `%{HTTP_REFERER}` (ou `%{HTTP:Referer}`) :
  - une première pour permettre (donc il faut la nier puisque la règle interdit) la chaîne vide, qui correspond théoriquement à un accès direct (l'utilisateur a tapé l'URL de l'image) ;

- une seconde pour exclure (à nier également), ce qui correspond à notre(nos) propre(s) domaine(s).

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si le référent n'est pas vide (accès direct en
théorie)
RewriteCond %{HTTP_REFERER} !=""
# ET
# si le référent ne correspond pas à une page de notre
propre site
RewriteCond %{HTTP_REFERER}
!^https?://.*\.nom_de_domaine\.fr/ [NC]
# alors en interdire l'accès (403)
RewriteRule \.(?:gif|jpe?g|png)$ - [F,NC]
# Ou pour renvoyer une image de remplacement :
#RewriteRule \.(?:gif|jpe?g|png)$
monImageDeRemplacement.png [NC,T=image/png]

# Alternative sans réécriture :
SetEnvIfNoCase Referer
"^https?://.*\.nom_de_domaine\.fr/" local_ref=1
SetEnvIf Referer ^$ local_ref=1

<FilesMatch "\.(jpe?g|gif|png)$">
    # Apache < 2.4
    <IfModule !mod_authz_core.c>
        Order allow,deny
        Allow from env=local_ref
    </IfModule>
    # Apache >= 2.4
    <IfModule mod_authz_core.c>
        Require env local_ref
    </IfModule>
</FilesMatch>
```

4.2. Bloquer un client ou lui servir un contenu spécifique

De manière théorique, il est possible d'identifier un client sur plusieurs points :

- son adresse IP : du moins, la machine par laquelle il passe pour accéder à Internet. Outre la possibilité de délibérément relayer ses requêtes via un proxy, n'oublions pas que tout le monde ne possède pas une adresse fixe et peut se connecter de plusieurs lieux différents ;
- à chacune de nos requêtes, nous envoyons un entête HTTP nommé User-Agent, dont le but est de fournir quelques informations sur notre environnement. Elles peuvent permettre de déterminer :
  - le navigateur ;
  - le système d'exploitation ;
  - s'il s'agit d'un robot d'indexation ;
  - éventuellement, même indirectement, le type d'appareil (mobile, tablette, etc) ;

  Notons que cet entête HTTP, au même titre que tous les autres, peut être, volontairement ou non, omis ou faussé.

Ainsi, pour travailler en réécriture sur l'adresse IP, nous devons écrire une condition (RewriteCond) qui compare l'adresse courante du client, représentée par la variable `%{REMOTE_ADDR}`, à une valeur à considérer en second argument. Si je désire bloquer l'adresse IP 75.76.77.78, je devrais écrire :

```
RewriteCond %{REMOTE_ADDR} =75.76.77.78
RewriteRule .* - [F]

# Alternative sans réécriture
# Apache < 2.4
<IfModule !mod_authz_core.c>
    Order Deny,Allow
    Deny from 75.76.77.78
</IfModule>
# Apache >= 2.4
<IfModule mod_authz_core.c>
```

```
    Require expr !-R "75.76.77.78"
    # Ou encore :
    <RequireAll>
        <RequireNone>
            Require ip 75.76.77.78
        </RequireNone>
    </RequireAll>
</IfModule>
```

À présent, travaillons sur l'entête HTTP User-Agent, symbolisé par la variable `%{HTTP_USER_AGENT}` ou `%{HTTP:User-Agent}`. Si votre site n'est pas prévu pour des versions d'Internet Explorer antérieures ou égales à 6, vous pourriez parfaitement les renvoyer sur une page les invitant à se mettre à jour ou à changer de navigateur par cette réécriture :

```
RewriteCond %{HTTP_USER_AGENT} MSIE\s*[0-6]\.
RewriteRule .* /dropie.html [L]
```

(on peut facilement trouver des listes de User-Agent par quelques recherches)

4.3. Rediriger un domaine (avec et sans www)

En mutualisé, lorsque vous possédez un nom de domaine, il n'est pas rare que votre prestataire assure automatiquement la résolution de mondomaine.fr et www.mondomaine.fr pour qu'ils aboutissent en fin de compte sur un même répertoire. Dès lors, toutes vos ressources sont normalement accessibles par www.mondomaine.fr mais aussi par domaine.fr. Ce cas de figure peut vous conduire à une situation de duplicate content : une même page pourra alors être référencée sous deux URL distinctes. La solution qui vient immédiatement à l'esprit est alors de rediriger de manière permanente mondomaine.fr sur www.mondomaine.fr. Cependant, puisque la configuration d'Apache (généralement un fichier .htaccess) est partagée par nos deux protagonistes, une directive Redirect n'est pas appropriée. En effet, elle

créerait une boucle infinie (que le client devrait détecter de lui-même), car www.mondaine.fr serait redirigé sur lui-même. L'unique solution, du moins pour Apache < 2.4, est de passer par la réécriture.

De quoi avons-nous besoin ?

- nous devons procéder à une redirection (permanente) : on ne manquera pas d'ajouter l'option R=permanent (ou R=301) à notre directive RewriteRule ;
- nous devons rediriger toute ressource, soit .* pour motif, sur le sous-domaine en www. La destination est donc la concaténation de l'URL http://www.mondomaine.fr/ et de la ressource demandée, `$0` (rappel : `$0` désigne l'ensemble de la sous-chaîne satisfaisant l'ensemble du motif) ;
- il nous manque à présent l'essentiel : la condition qui permet d'exclure www.mondomaine.fr de cette redirection. La variable qui nous intéresse est `%{HTTP_HOST}`, le nom de l'hôte que le client cherche à contacter, à comparer à www.mondomaine.fr.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si le domaine demandé n'est pas "www.mondomaine.fr"
RewriteCond %{HTTP_HOST} !=www.mondomaine.fr
# alors rediriger sur la même ressource que demandée
($0) sur le domaine www.mondomaine.fr
RewriteRule .* http://www.mondomaine.fr/$0
[L,R=permanent]]

# Syntaxe alternative avec Apache >= 2.4 :
#<If "%{HTTP_HOST} != 'www.mondomaine.fr'">
#    RedirectPermanent / http://www.mondomaine.fr/
#</If>
```

4.4. Forcer le protocole https pour une ressource

Nous voulons forcer les utilisateurs à employer le protocole "sécurisé", HTTPS, pour certains aspects sensibles, ici, matérialisés par un script d'authentification : login.php. Nous admettrons que les virtualhosts HTTP et HTTPS partagent la même racine (DocumentRoot), sans quoi cet exercice ne présenterait aucun intérêt. Ceci nous ramène en gros à la même situation que précédemment : le .htaccess où nous écrivons la règle sera lu que l'on utilise le protocole http standard ou https, ce qui, pour le dernier, serait à l'origine d'une boucle de redirection. Il est donc nécessaire de ne rien faire si le protocole https intervient déjà.

Articulation de la règle dont nous avons besoin :

- comme évoqué, la règle ne doit s'appliquer qu'à un client qui passe par le protocole http standard. Ce qui implique une condition (directive RewriteCond) testant que la valeur de la variable `%{HTTPS}` est différente (opérateur !=) de on (en tant que chaîne) ;
- nous procédons ici à une redirection HTTP (permanente), de login.php (motif ^login\.php$) à https://mondomaine.ext/login.php, les options R=permanent (ou R=301) et L(ast) doivent être ajoutées à notre directive RewriteRule.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si le protocole n'est pas sécurisé (protocole http
normal)
RewriteCond %{HTTPS} !=on
# (et que la page est login.php) alors rediriger sur les
mêmes site/page mais en HTTPS
RewriteRule ^login\.php$ https://%{HTTP_HOST}/$0
[L,R=permanent]
```

Note

- J'ai volontairement fait intervenir les variables `%{HTTP_HOST}` et `$0` (qui vaut login.php) pour le paramètre *destination* afin de rendre la règle plus facile à réutiliser.
- Pourquoi ne pas simplement comparer la valeur de la variable `%{SERVER_PORT}` à 80 ou 443 suivant le protocole ciblé ? Parce que cette démarche est erronée : il est parfaitement possible d'employer un port quelconque, à savoir mettre en écoute un serveur http standard sur le port 443 et inversement (de l'https sur le port 80). Au contraire de `%{SERVER_PORT}`, la variable `%{HTTPS}` est fiable et pertinente car à la charge du module SSL.

4.5. Rediriger des ressources qui ont été déplacées ou remplacées

4.5.1. Redirections HTTP simples

J'avais initialement placé mon blog dans un sous-répertoire, /blog/, de mon sous-domaine www.mondomaine.fr. Aujourd'hui, je souhaite le rendre "indépendant", en le déplaçant sous son propre sous-domaine (hôte virtuel et arborescence à part) blog.mondomaine.fr. Afin de conserver mon référencement et ne pas perturber mes lecteurs réguliers, je prends soin de mettre en place une redirection HTTP pour qu'ils puissent être informés de ce changement d'adresse. Ayant physiquement supprimé le sous-répertoire /blog/ de mon sous-domaine www, j'effectuerai cette redirection depuis la racine de son ancien emplacement. Cette redirection ne doit concerner que ce qui commence par /blog/, en n'oubliant pas de supprimer le slash en tête puisque nous passons par un fichier .htaccess, et capturer la partie du chemin qui suit de façon à renvoyer le visiteur sur son équivalent à la nouvelle adresse. Le motif de notre règle est : `^blog/(.*)` et sa destination `http://blog.mondomaine.fr/$1`, `$1` étant dynamiquement remplacé par ce qui suit blog/ dans le chemin. Enfin, je ne manque pas d'ajouter l'option R=permanent pour

obtenir la redirection HTTP permanente désirée ainsi que L qui devrait être systématiquement couplée à R pour s'assurer qu'Apache ne cherche pas à satisfaire d'autres règles.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteRule ^blog/(.*) http://blog.mondomaine.fr/$1
[L,R=permanent]

# Que l'ont peut écrire, sans réécriture :
RedirectPermanent /blog/ http://blog.mondomaine.fr/
```

Comme mentionné en fin de l'exemple, dans des cas triviaux, les redirections HTTP peuvent être assurées par les directives Redirect* plutôt que par la réécriture.

Cependant, personnellement, ne serait-ce que par principe, je ne fais jamais intervenir des directives Redirect* où la réécriture est utilisée. Je réécris systématiquement mes directives Redirect* en règles de réécriture équivalente de façon à éviter et maîtriser au mieux tout conflit mutuel. Il faut savoir que les directives Redirect*, quel que soit leur emplacement par rapport aux règles de réécriture, seront toujours appliquées **avant** la réécriture. En ces circonstances, cet effet pourrait s'avérer difficile à contrôler dans certaines circonstances.

4.5.2. Racine de site déplacée : renvoyer, de manière invisible, sur un sous-répertoire

Bien que ce soit fonctionnellement parlant la pire des solutions, faute d'avoir d'autres choix (surtout en mutualisé), vous avez déplacé une application, disons votre site actuellement en production, qui se situait à la racine vers un sous-répertoire, disons qu'il s'appelle *old*, afin de tester sa nouvelle version en condition réelle qui a pris sa place à la racine.

Tout d'abord, nous avons là un impératif à respecter : nous ne procéderons pas à une redirection HTTP mais bien à une simple réécriture (= cette pseudo-redirection n'apparaît pas côté client) de façon à conserver et

récupérer notre référencement en l'état pour la mise en production, à venir, de cette nouvelle version et, de manière plus générale, ne pas gêner inutilement les clients. Le principe de base consiste à renvoyer toute ressource vers elle-même mais située dans ce sous-répertoire old. Seul bémol, une telle règle va inexorablement conduire à une boucle : si un client demande /contact.html, il va être renvoyé sur /old/contact.html puis sur /old/old/contact.html et ainsi de suite. Il faut donc mettre en place une condition pour ne pas réécrire les chemins (représentés par la variable `%{REQUEST_URI}`) commençant déjà par /old/.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si le chemin ne commence pas par /old/
RewriteCond %{REQUEST_URI} !^/old/
# alors on renvoie sur le sous-répertoire old/
RewriteRule .* old/$0 [L]
```

Il existe plusieurs façons différentes pour arriver à cette fin. Une autre méthode pourrait passer par une règle de non-réécriture :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Ne rien faire pour ce qui commence par old/
RewriteRule ^old/ - [L]

# Pour tout le reste, renvoyer sur le sous-répertoire
old/
RewriteRule .* old/$0 [L]
```

Cette règle de non-réécriture, à placer avant l'autre et à lui adjoindre l'option Last, permet avant tout d'écarter les chemins qui sont corrects, quand l'autre s'assure ensuite de ne réécrire que ce qui ne l'est pas. L'approche employée ne joue pas ici, mais parmi d'autres règles, l'une pourrait s'avérer plus pratique à utiliser que l'autre.

4.6. Rerouter ce qui n'existe pas physiquement vers un contrôleur frontal ou semblable (MVC)

Avec une application, PHP ou autre, développée autour du modèle de conception MVC qui possède un système de routage complet, il devient inutile de gérer des règles de réécriture. C'est l'application elle-même qui va se charger de résoudre les adresses virtuelles, d'autant que c'est totalement différent. Or ceci requiert avant tout de renvoyer l'adresse de toute requête HTTP vers le contrôleur frontal. Cependant, parmi ces requêtes, certaines correspondent à des fichiers statiques physiquement présents, comme les feuilles de styles ou autres images et ce serait un gâchis en termes de ressources de les renvoyer inutilement vers l'application au lieu de les servir directement. C'est pourquoi, nous allons nous contenter de rediriger la requête HTTP vers ce contrôleur frontal uniquement si elle ne correspond pas à un fichier existant sur le disque.

Pour ce faire, à la règle qui renverrait tout au script qui invoque le contrôleur frontal, nous ajoutons deux conditions pour vérifier que un, le chemin, représenté par la variable `%{REQUEST_FILENAME}` ne corresponde pas à un fichier régulier (opérateur -f) et, deux, ne s'avère pas non plus être un répertoire (opérateur -d). C'est potentiellement incomplet suivant le système (liens symboliques non gérés) mais ça suffit généralement.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si la ressource demandée ne correspond pas à un
fichier (régulier)
RewriteCond %{REQUEST_FILENAME} !-f
# ET
# si la ressource demandée ne correspond pas non plus à
un répertoire
RewriteCond %{REQUEST_FILENAME} !-d
```

```
# alors renvoyer la requête sur handler.php
RewriteRule .* handler.php [L]
```

Note

Avec une version >= 2.2.16, vous pouvez avantageusement remplacer cette réécriture par une simple directive FallbackResource qui demande également moins de droits (AllowOverride) que la réécriture.

Avertissement

Contrairement à certains écrits que l'on peut malheureusement trouver, n'utilisez jamais ErrorDocument 404 pour rerouter des pages inexistantes. Non seulement ErrorDocument est le dernier maillon de la chaîne (réécriture > FallbackResource > ErrorDocument) mais mal employée, vous feriez passer des ressources inexistantes pour le contraire, chose que vous finiriez par payer très cher au niveau de votre référencement.

4.7. Hôtes virtuels de masse simulés

Pour une petite structure et une application type CMS partagée, nous voulons dynamiquement attribuer un espace physique (un répertoire sur le disque) à chaque utilisateur et que ce dernier soit accessible par un sous-domaine reprenant son login (`login`.mondomaine.fr). La réécriture va nous permettre d'établir à la volée la correspondance entre le sous-domaine et le chemin sur le disque.

D'un côté nous souhaitons conserver à part le site de base (www.mondomaine.fr sur `/var/www/`) ; de l'autre ces sous-répertoires utilisateurs seront regroupés dans `/var/subdomains/`.

Comment s'y prendre ?

- Au niveau des conditions, nous avons besoin de :

    - écarter (opérateur d'inégalité : !=) l'hôte, représenté par la variable `%{HTTP_HOST}`, www.mondomaine.fr ;
    - vérifier que ce même domaine, nous retrouvons `%{HTTP_HOST}`, se finisse par ".mondomaine.fr" et mémoriser ce qui se trouve avant. Ce qui donne pour expression régulière : `(.+)\.mondomaine\.fr$` ;
    - éventuellement, suivant comment ça doit être géré, vérifier que la partie mémorisée lors du point précédent corresponde (variable `%1`) à un répertoire existant ("opérateur" -d)
- Quant au rôle de la règle, au lieu de servir le fichier foo/bar.html (http://sandrine.mondomaine.fr/foo/bar.html), elle doit renvoyer sur le sous-répertoire sandrine de `/var/subdomains/` (chemin final : `/var/subdomains/sandrine/foo/bar.html`). En d'autres termes, réécrire tout chemin (motif `.*`) satisfaisant les précédentes conditions vers `/var/subdomains/%1/$0` (`%1` désignant toujours le login de l'utilisateur extrait du nom de domaine et `$0` permet de recopier le chemin complet de la requête).

```
# Apache < 2.4, si nécessaire
#NameVirtualHost *:80

<VirtualHost *:80>

    ServerName www.mondomaine.fr
    ServerAlias *.mondomaine.fr
    DocumentRoot /var/www

    <Directory /var/www>
        # Apache < 2.4
        <IfModule !mod_authz_core.c>
            Allow from all
        </IfModule>
        # Apache >= 2.4
        <IfModule mod_authz_core.c>
            Require all granted
```

```
        </IfModule>
    </Directory>

    <DirectoryMatch /var/subdomains/*>
        # Apache < 2.4
        <IfModule !mod_authz_core.c>
            Allow from all
        </IfModule>
        # Apache >= 2.4
        <IfModule mod_authz_core.c>
            Require all granted
        </IfModule>
    </DirectoryMatch>

    RewriteEngine On

    # Si l'hôte demandé n'est pas "www.mondomaine.fr"
    RewriteCond %{HTTP_HOST} !=www.mondomaine.fr
    # ET
    # s'il se termine par ".mondomaine.fr", en
mémorisant ce qu'il y avant pour la suite (en %1)
    RewriteCond %{HTTP_HOST} (.+)\.mondomaine\.fr$
    # ET
    # si ça (/var/subdomains/%1) correspond à un sous-
répertoire existant
    RewriteCond /var/subdomains/%1 -d
    # alors servir le fichier /var/subdomains/%1/$0 (ça
agit comme un DocumentRoot dynamique au final)
    RewriteRule .* /var/subdomains/%1/$0 [L]

</VirtualHost>
```

Astuce

Aux détails près, cette réécriture est le strict équivalent d'une ligne VirtualDocumentRoot :

```
VirtualDocumentRoot /var/subdomains/%-3+
```

Cette dernière ayant l'avantage, pour Apache >= 2.4, de dynamiquement corriger le DocumentRoot associé à la requête de départ.

Avertissement

- Contrairement aux autres exemples où j'ai volontairement privilégié l'approche par fichier .htaccess car plus courante, cette tâche ne peut être réalisée que depuis les fichiers de configuration d'Apache. Je rappelle en effet que rien qu'en ce qui concerne directement la réécriture, seul ce niveau permet de renvoyer directement sur le système de fichiers. Apache vous l'interdit depuis un fichier .htaccess par une erreur 403 (sans cette restriction un utilisateur pourrait, volontairement ou non, rendre public tout ce que les droits système lui permettraient).
- Pour des hôtes virtuels de masse ou assimilé, vous devez préalablement mettre en place une résolution adéquate (wildcard DNS par exemple).

4.8. Effectuer une redirection en fonction d'un paramètre de query string

Voilà, j'ai mis en place une réécriture pour les billets de mon petit blog personnel afin qu'ils aient de belles URL. Or, des gens les ont déjà référencés sur leurs propres sites avec l'ancienne forme hideuse. Tant qu'à faire, j'aimerais bien que ceux qui les suivent en soient informés. Comment forcer l'emploi de la nouvelle forme ?

Pour simplifier et pour l'aspect pédagogique, partons du principe que je ne suis pas un gros blogueur, j'écris au plus un billet par semaine ce qui rend acceptable le fait d'éditer moi-même le .htaccess pour y écrire les redirections nécessaires au fur et à mesure.

Limitons-nous, pour l'exemple, à une URL : on veut rediriger article.php?id=2, l'ancienne forme, sur les-sessions-en-php.html, la nouvelle.

Je rappelle qu'une telle URL doit préalablement être découpée suivant ses différentes composantes dans votre esprit avant de commencer à rédiger la règle

correspondante. En effet, vous ne pouvez pas écrire une règle comme celle-ci :

```
# Ceci est faux
RewriteRule ^article\.php\?id=2$ les-sessions-en-
php.html [L,R=permanent]
```

Voici comment aborder le problème :

- à la directive RewriteRule, Apache ne fournit que le chemin HTTP (pour partie si la règle se situe dans un fichier .htaccess). Ce qui correspond au morceau article.php. Par conséquent, il nous faut `^article\.php$` pour motif et la destination est les-sessions-en-php.html. Puisque le but est de réaliser une vraie redirection HTTP, on ne manquera pas d'ajouter, à la fin, l'option de redirection, R=permanent, et Last pour appliquer de suite la règle et éviter toute erreur ;
- tester toute autre partie, dont la query string (id=2), se réalise uniquement par une directive RewriteCond et la variable associée. On sait que la variable reprenant la chaîne de requête s'appelle `%{QUERY_STRING}`. Par contre, cette dernière est peu commode à gérer car Apache n'en sépare pas les différentes clés/valeurs, vous êtes obligés de travailler sur sa forme brute (clé1=valeur1&clé2=valeur2&clé3=valeur3).

Ce qui nous donnerait :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si la chaîne de requête contient un paramètre id de
valeur 2
RewriteCond %{QUERY_STRING} (?:^|&)id=2(?:&|$)
# (et si la page est article.php) alors renvoyer sur
les-sessions-en-php.html
RewriteRule ^article\.php$ les-sessions-en-php.html
[L,R=permanent]
```

Sauf que cette dernière règle est incomplète : Apache, par défaut, quand la destination ne comporte pas de chaîne de requête, va recopier l'ancienne à la fin de l'URL suivie. En clair, ici, nous serions renvoyé sur les-sessions-en-php.html?id=2 or nous ne voulons pas de cette chaîne de requête inutile. Comment la supprimer ? La réponse est simple, il suffit de créer une chaîne de requête vide en ajoutant un point d'interrogation à la fin du paramètre *destination* (celui-ci n'apparaîtra pas côté client car le module de réécriture supprime ces query string explicitement vides entre temps). Ce qui nous donne, au final :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si la chaîne de requête contient un paramètre id de
valeur 2
RewriteCond %{QUERY_STRING} (?:^|&)id=2(?:&|$)
# (et si la page est article.php) alors renvoyer sur
les-sessions-en-php.html en supprimant la query string
originale
RewriteRule ^article\.php$ les-sessions-en-php.html?
[L,R=permanent]
```

Avec une version 2.4, il est possible d'utiliser le flag QSD qui remplit la même fonction :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteCond %{QUERY_STRING} (?:^|&)id=2(?:&|$)
RewriteRule ^article\.php$ les-sessions-en-php.html
[QSD,L,R=permanent]
```

Je dois cependant avouer que cet exemple est incomplet car si j'ajoute la règle qui se charge de faire la résolution inverse (les-sessions-en-php.html vers article.php?id=2), elles vont mutuellement s'invoquer et ainsi créer une boucle infinie. Pour éviter cela, la seule solution est d'ajouter un faux paramètre de query string pour la réécriture interne (les-sessions-en-php.html vers article.php?id=2) et de n'effectuer la

redirection article.php?id=2 vers les-sessions-en-php.html que quand ce faux paramètre est absent, qui est alors censé être synonyme que le client a utilisé l'ancienne forme. Les règles finales complètes sont les suivantes :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si la chaîne de requête ne contient pas de paramètre
internal_redirect à valeur true (pour éviter une boucle)
RewriteCond %{QUERY_STRING} !
(?:^|&)internal_redirect=true(?:&|$)
# ET
# si la chaîne de requête contient un paramètre id de
valeur 2
RewriteCond %{QUERY_STRING} (?:^|&)id=2(?:&|$)
# (et si la page est article.php) alors renvoyer sur
les-sessions-en-php.html en supprimant la query string
originale
RewriteRule ^article\.php$ les-sessions-en-php.html?
[L,R=permanent]

# les-sessions-en-php.html => article.php?id=2
# On ajoute un faux paramètre (internal_redirect=true)
pour indiquer que article.php?id=2 est le résultat d'une
réécriture
RewriteRule ^les-sessions-en-php\.html$ article.php?
id=2&internal_redirect=true [L]
```

Note

Les directives Redirect* ne sont pas prévues pour gérer la partie query string d'une URL, à moins d'avoir une version 2.4 pour l'encapsuler dans une directive <If>. Seule la réécriture en est capable.

4.9. Interdire l'accès au site avant une certaine date sauf pour une adresse IP

On souhaite interdire l'accès à un site avant sa date de lancement fixée au 1er décembre 2012 à minuit,

excepté pour une adresse IP donnée pour maintenance et tests.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si l'adresse IP est différente de 1.2.3.4
RewriteCond %{REMOTE_ADDR} !=1.2.3.4 [OR]
# OU
# que nous ne sommes pas après le 1er décembre 2012
minuit
RewriteCond %{TIME} <20121201000000
# alors on interdit l'accès
RewriteRule .* - [F]

# Avec Apache >= 2.4, directement, sans réécriture :
#Require expr %{TIME} >= 20121201000000 || -R "1.2.3.4"
```

Commençons par la règle de base :

- l'interdiction par rapport au protocole HTTP se traduit par le retour d'un code HTTP 403. C'est l'option F(orbidden) au niveau de la directive RewriteRule qui remplit cette fonction (ou éventuellement R=403 suivant la version d'Apache) ;
- le site en question doit être totalement inaccessible, ce qui signifie que tout est concerné : implique le motif .* en premier paramètre de RewriteRule ;
- Enfin, notre règle n'a pas à modifier l'URL, ce qui se traduit par un tiret (-) en deuxième paramètre.

Il ne nous reste qu'à ajouter nos deux conditions : une sur l'adresse IP et l'autre sur la date.

- l'adresse IP est représentée par la variable `%{REMOTE_ADDR}`. Elle sera le premier paramètre de notre première RewriteCond. Étant donné que l'on cherche à interdire toute adresse qui n'est pas 1.2.3.4, l'opérateur à utiliser sera l'inégalité (différent de) : !=. La seconde partie de cette même RewriteCond sera donc !=1.2.3.4 ;

- pour la représentation complète de la date (heure comprise), il me paraît plus facile et général de passer par la variable `%{TIME}` dont je rappelle le format : année (4 chiffres), mois (2 chiffres), jour (2 chiffres), heures (2 chiffres), minutes (2 chiffres), secondes (2 chiffres), le tout collé ensemble. Le 1er décembre 2012 minuit s'écrit donc 20121201000000. Ce format permet une comparaison de dates avec un simple ordre lexicographique, l'opérateur < sera suffisant à établir le cas où nous sommes antérieurs à cette date.

Cependant, il ne faut pas oublier que les directives RewriteCond, quand il y en a plusieurs pour une même RewriteRule, sont par défaut unies par un et logique. Or c'est un ou logique qu'il nous faut ici, par conséquent, on ne manquera pas d'ajouter le drapeau ornext (ou OR) à la première des deux.

4.10. Renvoyer le visiteur selon les heures de bureau

Renvoyer l'utilisateur sur une page indiquant que les bureaux sont fermés :

- le samedi après midi
- le dimanche, toute la journée
- les autres jours de la semaine avant 8H00 et après 17H30

Cet exercice n'est pas aussi complexe qu'il pourrait y paraître : il repose sur les variables temps (`%{TIME_*}`), l'option ornext (ou OR) de la directive RewriteCond et les opérateurs de comparaison.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si nous sommes ...
# dimanche (soit 0)
RewriteCond %{TIME_WDAY} =0 [OR]
```

```
# OU
# samedi après 12h00 (soit 6, samedi, 1200)
RewriteCond %{TIME_WDAY}%{TIME_HOUR}%{TIME_MIN} >61200
[OR]
# OU
# avant 8H00
RewriteCond %{TIME_HOUR} <08 [OR]
# OU
# après 17H30
RewriteCond %{TIME_HOUR}%{TIME_MIN} >1730
# alors renvoyer sur bureauxfermes.html
RewriteRule .* bureauxfermes.html [L]
```

4.11. Masquer l'extension de ses scripts PHP

Qu'importe les motivations, apportons une solution à une demande fréquemment rencontrée sur les forums : masquer l'extension de ses scripts. C'est-à-dire, avoir pour URL http://domaine.ext/foo en lieu et place de http://domaine.ext/foo.php.

Comment s'y prendre ?

- avant tout, nous devons ignorer tout chemin qui comporterait déjà une extension de fichiers. Si cette dernière partie est présente, il est inutile de chercher à interpréter l'URL différemment. Un motif simple, possible, pour notre règle (RewriteRule) serait de ne traiter que tout chemin qui ne comporte aucun point, soit `^[^.]+$` ;
- le but de notre règle, quand son motif en est satisfait, est de la suffixer de la chaîne ".php". La destination n'est autre que `$0.php`. Rappelons que `$0` reprend la chaîne qui satisfait l'ensemble du motif, soit, ici, l'intégralité du chemin ;
- enfin, ne réécrivons les URL qui ne comportent aucun point seulement quand il existe un script php du même nom. Ce qui permet d'éviter certaines 404, surtout quand la requête HTTP cible un répertoire, leurs noms comportant rarement un point. Pour nous

en assurer, ajoutons la condition (RewriteCond) : si un fichier php du même nom (`%{REQUEST_FILENAME}.php`) existe (opérateur `-f`).

```
Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si en ajoutant ".php" au chemin, on aboutit sur un
fichier qui existe physiquement
RewriteCond %{REQUEST_FILENAME}.php -f
# ET
# si l'adresse initiale qui ne contient aucun point
(soit pas d'extension de fichier)
# alors servir le script PHP du même nom
RewriteRule ^[^.]+$ $0.php [L]
# NOTE : le motif ^(?:.*/)?[^/.]+$ peut être plus
approprié
```

En bonus, profitons-en pour forcer, par une redirection HTTP, nos visiteurs à employer des adresses où l'extension ".php" ne figure pas. Nous redirigeons (implique le drapeau R pour la redirection couplé à L(ast) pour l'application immédiate de la règle) alors les requêtes directe de tout script php (soit `(.+)\.php$`) vers leur équivalent sans extension (`$1`, notre unique parenthèse capturante mémorisant la partie qui précède le suffixe ".php"). Par contre, en l'état, ces deux règles vont entraîner une boucle sans fin, il est nécessaire d'ajouter une condition pour garantir que l'adresse en .php n'est pas déjà le résultat d'une redirection interne (ce qui est ci-dessous certifié en testant que la variable d'environnement `REDIRECT_STATUS` est [la chaîne] vide). Ce qui nous donne la règle complémentaire suivante :

```
# Si l'adresse courante n'est pas le résultat d'une
précédente redirection
# (condition requise, sans quoi vous créeriez une boucle
infinie entre les 2 règles)
RewriteCond %{ENV:REDIRECT_STATUS} =""
# (et si l'adresse se termine par .php) alors effectuer
```

```
la redirection HTTP permanente vers la même adresse sans
le ".php"
RewriteRule (.+)\.php$ $1 [L,R=permanent]
```

Note

Si le but recherché est réellement de faire croire que ce ne sont pas des scripts PHP, vous pouvez remplacer R=permanent par R=404.

5. Difficultés communes et résolution

5.1. Le "piège" de l'arborescence virtuelle

Ceci ne concerne que les gens :

- qui ont au moins une ressource virtuelle réécrite contenant au moins un slash
- qui n'ont pas prévu l'effet que ça aurait
- qui utilisent des chemins HTTP relatifs pour les ressources liées (liens, CSS, javascript, images ,etc)

Si vous créez une ressource virtuelle faisant intervenir au moins un caractère slash, comme celle-ci :

```
RewriteRule ^article/(\d+)/.+\.html$ article.php?id=$1
[L,QSA]
```

Le problème qui se pose, avec des liens, sources, etc relatifs c'est que comme le client ignore tout de la réécriture, vous allez fausser l'idée qu'il a de l'arborescence du site. Par exemple, si j'ai une css :

```
<link href="stylesheet.css" rel="stylesheet"
type="text/css"/>
```

Si le tout était situé à la racine, que l'URL courante est à présent http://mondomaine.fr/article/2/les-sessions-en-php.html le client va chercher cette CSS à l'URL http://mondomaine.fr/article/2/stylesheet.css au lieu de http://mondomaine.fr/stylesheet.css. Le chemin qu'il a calculé à partir de l'URL ne peut être que faux.

Quelles solutions pour ce cas de figure ?

- Il serait envisageable de réécrire ou rediriger ces ressources fausses par l'introduction de nouvelles règles. Dans notre cas, il serait possible de régler le problème de la feuille de style par :

```
RewriteRule article/\d+/stylesheet\.css$
/stylesheet.css [L,R=permanent]
# Plus généraliste :
#RewriteRule /stylesheet\.css$ /stylesheet.css
[L,R=permanent]
```

Toutefois, je déconseille vivement cette voie, cette multiplication inutile de règles risque d'introduire plus de conflits qu'elle ne saurait en résoudre, cela peut vite devenir complexe et ingérable !

- Indiquer l'adresse de référence par une balise <base href="..."> ajoutée dans l'entête du document. Le client HTTP étant censé se baser sur celle-ci, quand elle est disponible, plutôt que de se remettre à sa propre vision des chemins, il devrait de nouveau être en mesure de résoudre correctement tous ces chemins relatifs indépendamment de la réécriture.

```
<html>
    <head>
        <base href="http://mondomaine.fr" />
        <link href="stylesheet.css" rel="stylesheet"
type="text/css"/>
    </head>
```

La balise <base href="..."> ne concerne que les liens relatifs, pas les URL (http://...) ni les chemins HTTP absolus (qui commencent par un slash).

- Enfin, une autre solution, pourrait être de tout simplement modifier les chemins relatifs en absolus. Notre feuille de style étant à la racine, cela nous donne :

```
<link href="/stylesheet.css" rel="stylesheet"
type="text/css"/>
```

5.2. Conflit entre la négociation de contenu et la réécriture

La négociation de contenu, qu'est-ce que c'est ? De manière simplifiée, la négociation de contenu c'est la capacité du serveur à choisir pour le client, en fonction des préférences de ce dernier, une ressource quand son nom est incomplet (au sens, par rapport au serveur, qu'il manque l'extension réelle). En effet, lors de toute requête HTTP, en tant que client, vous émettez, au travers de différents entêtes, vos préférences par rapport à la langue (entête Accept-Language), au type du fichier (png > gif > jpeg > texte, par exemple, via l'entête Accept), d'un encodage (algorithme comme gzip utilisé pour compresser le corps de la réponse ; entête Accept-Encoding) et un jeu de caractères (entête Accept-Charset).

Prenons un exemple trivial. Admettons que j'ai 2 fichiers côté serveur : page.txt, un fichier texte pur et page.html, le même formaté avec HTML. Que se passe-t-il, si je demande page (sans extension) lorsque la négociation de contenu est activée ? Et bien tout va dépendre de l'entête Accept que vous allez envoyer :

- Comme tout navigateur, si je donne priorité au format html (type mime text/html) en abaissant le facteur de priorité q associé aux autres tel que :

  ```
  Accept: text/html; text/*,q=0.9; */*,q=0.1
  ```

  J'obtiens bien en réalité page.html.
- Inversons à présent, en donnant priorité aux fichiers texte (type mime text/plain) :

  ```
  Accept: text/plain; text/html,q=0.9; */*,q=0.1
  ```

  Nous tombons sur page.txt.

Où cela nous mène-t-il ? Quel est le rapport avec la réécriture ?

Il existe un cas spécifique où la négociation de contenu peut entrer en conflit avec la réécriture. Pour être exact, il ne s'agit pas à proprement parler d'un conflit mais de voir une ou plusieurs règles de réécriture

purement et simplement ignorées. Il faut en effet réunir deux conditions :

- bien que cela puisse paraître évident, il faut que la négociation soit activée (module mod_negociation chargé et qu'un Options (+)MultiViews s'applique au répertoire en question) ;
- que vous ayez une ou des règles qui auraient pour effet d'écourter leur nom et qui auraient la malchance de correspondre à un fichier physique. On pense généralement à une règle comme celle-ci, pour continuer avec mon exemple de "page" :

  ```
  RewriteRule ^page$ index.php?p=$0
  ```

  Comme la négociation de contenu intervient avant la réécriture, en renvoyant page sur page.html, la règle ci-dessus ne sera jamais exécutée.

Jusque là, on serait tenté de se dire que c'est très limité comme effet de bord. Or, ce n'est pas fini : ça peut devenir beaucoup plus vicieux si la négociation de contenu est combinée à la fonctionnalité PATH_INFO, consistant à accepter et extraire la partie de chemin superflu d'une requête.

Je garde mon fichier page.html comme base. En temps normal, si un client demandait page.html/partie/excédentaire, le serveur nous renverrait une erreur 404 car ça ne correspond pas à un fichier physique (page.html n'étant pas un répertoire). Avec la fonctionnalité PATH_INFO, Apache s'arrêterait sur page.html, une ressource qui existe et avant de l'invoquer, peuplerait la variable PATH_INFO avec l'excédent (ici /partie/excédentaire).

Si à présent, nous ajoutons la négociation de contenu à PATH_INFO, cela nous permettrait de rendre l'extension du fichier optionnelle. Pour le même exemple, page/partie/excédentaire serait alors équivalente. Imaginez à présent les effets bien plus vastes au travers d'une règle comme celle-ci :

```
RewriteRule ^page/(\d+)/.+\.html$ index.php?p=$1
```

Elle serait purement et simplement ignorée car toujours du ressort de la négociation de contenu (mais avec la complicité de PATH_INFO). L'URL page/2/nouvelle-version.html n'est pas réécrite mais est résolue comme page.html avec, pour valeur de PATH_INFO, /2/nouvelle-version.html.

Conclusion : désactivez la négociation de contenu, comme je l'ai systématiquement fait dans mes exemples en ajoutant, au pire, en tête de vos fichiers .htaccess :

```
Options -MultiViews
```

La fonctionnalité PATH_INFO est (dés)activable par la directive AcceptPathInfo si tant est que vous ayez le droit de l'utiliser localement. Si vous souhaitez expérimenter cet effet de bord, forcez la négociation de contenu et le PATH_INFO par les lignes suivantes :

```
Options +MultiViews
AcceptPathInfo on
```

5.3. Interprétation des codes d'erreur HTTP renvoyés dans le cadre de la réécriture

Plusieurs types d'erreur liés à la réécriture peuvent se manifester :

- l'erreur 403 Forbidden peut survenir en l'absence d'une des options permettant de suivre les liens symboliques (FollowSymLinks) qui est nécessaire au fonctionnement de la réécriture (restriction à but sécuritaire).
- l'erreur 404 Not Found peut avoir plusieurs causes :
  - Le module de réécriture est bien chargé mais n'est pas activé (absence de RewriteEngine On) ;

    - Le fichier .htaccess est totalement ignoré par le serveur de par sa configuration lorsque la directive AllowOverride, pour le répertoire contenant ce fichier .htaccess (éventuellement par héritage) vaut None alors qu'au moins FileInfo est nécessaire ;
    - Les règles sont erronées dans la mesure où il n'y a pas de correspondance entre la ressource demandée et vos règles (leurs motifs) ;
    - Le document vers lequel la redirection est effectuée est inexistant. Ceci pouvant notamment s'expliquer par une erreur au niveau du chemin.
- L'erreur 500 Internal server error :
    - Le module mod_rewrite n'est pas actif. De ce fait, les directives Rewrite* ne sont pas reconnues et conduisent à ce type d'erreur ;
    - La configuration du serveur n'autorise pas l'usage des directives de réécriture (Rewrite*) par l'absence de la valeur FileInfo au niveau de la directive AllowOverride par rapport au répertoire contenant le fichier htaccess ;
    - Une règle engendre une boucle infinie, Apache met alors fin au processus de réécriture par une erreur 500.

Dans tous les cas, vous trouverez sans doute une explication explicite dans les journaux du serveur si tant est que vous y avez accès.

5.4. En dernier recours : déboguer la réécriture d'URL

Avant la version 2.4.0, le débogage de la réécriture consiste en l'écriture d'une trace, en fin d'un fichier texte désigné, décrivant certaines étapes de chaque processus de réécriture. Ce débogage ne peut être activé que si l'on a la main sur le serveur puisqu'il implique de modifier directement le fichier de configuration d'Apache où il sera nécessaire d'ajouter deux directives :

- RewriteLog [fichier] : le journal où écrire les différentes traces qui concernent la réécriture
- RewriteLogLevel [nombre de 0 à 9] : le degré de verbosité. 0, la valeur minimale, désactive toute journalisation des actions de réécriture. Un niveau élevé peut aller jusqu'à ralentir le serveur.

Exemple :

```
RewriteLog logs/rewrite.log
RewriteLogLevel 2
```

En revanche, le système de journalisation d'Apache a été complètement revu à la version 2.4.0 : il est désormais possible d'assigner un niveau de verbosité propre à chaque module, ce qui rend obsolète les deux directives ci-dessus (elles n'existent plus, chercher à les utiliser provoquerait de fait une erreur 500). Seul bémol, non des moindres, les traces produites par le module de réécriture seront écrites dans le journal d'erreur au lieu d'un fichier séparé. Mis à part ce point, le fonctionnement est équivalent : il y a 8 niveaux de verbosité nommés pour le mode de débogage, allant de `trace1`, le plus faible, à `trace8`, le plus élevé et il doit être à présent spécifié sur la ligne de la directive LogLevel. Exemple pour un débogage relativement moyen (`trace3`) :

```
# warn est le niveau par défaut
LogLevel warn mod_rewrite.c:trace3
```

Attention : le nombre de traces qui vont s'inscrire est directement proportionnel au niveau de verbosité et au nombre de règles !

6. Aller plus loin

6.1. Gérer le possible duplicate content inhérent à la réécriture

Le duplicate content (contenu dupliqué) est le fait d'avoir un même contenu pour plusieurs URL différentes. Pourquoi en parler ? En quoi est-ce gênant ? Les moteurs de recherche repèrent très bien la chose et si j'aborde cette question, c'est que quand cela arrive, vous pouvez payer très cher une telle erreur en étant déclassé plus ou moins significativement.

Cette situation est intimement liée à la réécriture pour plusieurs raisons :

- les formes initiales d'URL, type article.php?id=2, faute d'avoir été redirigées sont toujours valides et référencées
- en général, au niveau de la réécriture, on écrit des règles comprenant des expressions régulières donc qui peuvent accepter bien d'autres chaînes que celle que nous prévoyons : il suffit d'une erreur de la part de quelqu'un qui recopie l'URL comme vous, qui modifieriez un titre sans penser aux répercussions. Concrètement, j'ai fait en sorte d'obtenir le lien article-2-les-sessions-en-php.html pour article.php?id=2, mais que se passerait-il si un visiteur faisait un lien sur son blog avec article-2-sessions-en-php.html ?

Il est vital de prendre en compte cette question, qui n'a normalement un coût que très faible en temps et lignes de code. Ceci implique tout d'abord d'avoir le moyen d'identifier ou retrouver l'URL réécrite officielle faisant foi :

- soit vous stockez cette URL (en base, par exemple, dans une colonne dédiée parmi les données qu'elles représentent)
- soit, si elle est calculée à partir du reste (comme article-`identifiant`-

`titre_sous_une_forme_URL_friendly`.html), vous recalculez cette URL réécrite à chaque fois que vous en avez besoin

La première possède l'avantage d'avoir un coût moindre en principe, par contre, l'éventuel inconvénient c'est de bien gérer la mise à jour de cette donnée si elle représente les autres données quand ces dernières sont modifiées. Ici, afin que le code soit moins long et plus explicite, je n'utilise que la seconde.

Ensuite, il y a deux approches :

- vous effectuez une redirection HTTP permanente vous-même, en PHP, les URL qui conduiraient au duplicate content sur la bonne :

```
<?php
if (isset($_GET['id']) && (FALSE !== $id =
filter_var($_GET['id'], FILTER_VALIDATE_INT))) {
    $bdd = new PDO(/*...*/);
    $bdd->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $stmt = $bdd->prepare('SELECT * FROM billets WHERE
id = ?');
    $stmt->bindValue(1, $id, PDO::PARAM_INT);
    $stmt->execute();

    if (FALSE === ($billet = $stmt-
>fetch(PDO::FETCH_ASSOC))) {
        http_not_found();
    }
    $expectedPath = sprintf('/billet-%d-%s.html', $id,
slugify($billet['titre']));
    if ($_SERVER['REQUEST_URI'] != $expectedPath) {
        http_redirect_permanent($expectedPath);
    }
    // affichage normal du billet
} else {
    http_not_found();
}
```

- vous laissez les robots d'indexation se débrouiller mais il vous faut indiquer, dans votre page, via une balise <link rel="canonical"/> l'url réelle de la ressource que vous voulez qu'il référence quoi qu'il arrive :

```
<link rel="canonical" href="url complète réelle à
référencer"/>
```

Exemple :

```
<?php
if (isset($_GET['id']) && (FALSE !== $id =
filter_var($_GET['id'], FILTER_VALIDATE_INT))) {
    $bdd = new PDO(/*...*/);
    $bdd->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    $stmt = $bdd->prepare('SELECT * FROM billets WHERE
id = ?');
    $stmt->bindValue(1, $id, PDO::PARAM_INT);
    $stmt->execute();

    if (FALSE === ($billet = $stmt-
>fetch(PDO::FETCH_ASSOC))) {
        http_not_found();
    }
    $expectedPath = sprintf('/billet-%d-%s.html', $id,
slugify($billet['titre']));
    // ...
    // dans la partie <head></head>, vous ajoutez :
    echo '<link rel="canonical" href="' .
$_SERVER['HTTP_HOST'] . $expectedPath . '"/>';
    // ...
    // puis affichage normal du billet et de la page
} else {
    http_not_found();
}
```

6.2. URL et caractères "spéciaux"

Ce point délicat mérite un aparté dédié : il faut bien prendre conscience qu'Apache ne considère rien

d'autre que l'ASCII non-étendu. De ce fait, il ignore toute locale et tout jeu (ISO-8859-1 comme UTF-8 ou autre), donc vous ne pourrez pas gérer les caractères accentués notamment. Ce qui est totalement justifié puisque tout bonnement impossible car c'est le jeu de la requête HTTP, telle qu'elle est envoyée par le client, qu'Apache doit alors traiter. Or le client est libre d'envoyer ce qu'il veut ; aucun algorithme n'est capable de valider et/ou retrouver le jeu de la requête d'origine.

Si vous tenez vraiment à réécrire des URL précises qui contiennent des caractères "spéciaux", ceci ne vous laisse pas d'autre choix que d'écrire la représentation de tels caractères pour un jeu donné telle que la machine les représente, octet par octet, par la notation hexadécimale \xAA où AA est la valeur hexadécimale d'un octet.

Par exemple, pour réécrire noeud comme nœud, de manière insensible à la casse, en supposant que les requêtes des clients sont en UTF-8, nous devrions alors écrire une règle telle que celle ci-dessous :

```
RewriteRule ^[nN](?:[oO][eE]|\xC5[\x92\x93])[uU][dD]$
noeud.html
```

Conclusion : il est préférable, sauf cas où cela est réellement justifié, d'opter pour un motif générique (type .* combiné à un préfixe ou suffixe) plutôt que de chercher à inclure tous les caractères, ce qui est difficile voir impossible puisque notamment dépendant du jeu de la requête telle que le client l'envoie.

Note

- Les systèmes Windows possèdent une API à part entière qui interface le système de fichiers en Unicode (UTF-16), Apache suppose, donc attend sur ceux-ci, des requêtes HTTP en UTF-8 de la part des clients. Pour être précis, c'est la bibliothèque sous-jacente APR (Apache Portable Runtime) qui s'en charge et effectue les conversions UTF-16 (système de fichiers) <=> UTF-8 (HTTP).

- L'option NC (ou nocase), au niveau de RewriteRule comme RewriteCond, ne considère pas non plus les caractères hors de la plage ASCII non-étendu.

6.3. RewriteBase : quand Apache est incapable de résoudre physiquement les chemins HTTP

En temps normal, RewriteBase est inutile. Vous utilisez des chemins (HTTP) relatifs pour les ressources réelles vers lesquelles sont reroutées les requêtes HTTP et Apache sait les gérer. Toutefois, dans les situations où une partie, au moins, de l'arborescence HTTP est virtuelle (physiquement inexistante), Apache sera incapable de résoudre correctement la requête de destination. Voici une liste non exhaustive de cas où RewriteBase devient nécessaire pour indiquer à Apache le chemin HTTP qu'il doit utiliser :

- les répertoires personnels des utilisateurs système (module mod_userdir, directive UserDir)
- les alias (module mod_alias, directives Alias et AliasMatch)
- les hôtes virtuels de masse (module : mod_vhost_alias ; directives VirtualDocumentRoot, VirtualDocumentRootIP, VirtualScriptAlias et VirtualScriptAliasIP)

Admettons que je fasse intervenir userdir : je créée /home/julp/public_html/.htaccess pour renvoyer foo (http://localhost/~julp/foo) sur bar.php (http://localhost/~julp/bar.php) :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteRule ^foo$ bar.php
```

Apache ne sait pas ici gérer correctement le chemin de bar.php : il va chercher celui-ci par rapport à la racine

du serveur (DocumentRoot). Avec /var/www en DocumentRoot, il va le réécrire en /var/www/home/julp/public_html/bar.php (au lieu de simplement /home/julp/public_html/bar.php).

Pour que cela fonctionne correctement, on a besoin ici d'expliciter le chemin HTTP sur lequel il doit se baser, /~julp/ ici :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on
RewriteBase /~julp/

RewriteRule ^foo$ bar.php
```

Et à présent, tout fonctionne correctement.

À noter que RewriteBase :

- ne s'applique qu'aux chemins relatifs des ressources vers lesquelles vous réécrivez :
  - une URL complète comme, http://www.mondoamaine.fr/bar.php n'est bien évidemment pas concernée
  - un chemin HTTP absolu, donc qui commence par un slash, comme /bar.php, sera conservé/repris intact
- concerne/impacte **tous** les chemins relatifs des ressources vers lesquelles vous réécrivez

Si je reprends l'exemple précédent, on pourrait se passer de tout RewriteBase en modifiant le .htaccess, et plus particulièrement le chemin de la règle, tel que :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteRule ^foo$ /~julp/bar.php
```

Toutefois, l'usage de RewriteBase est recommandé pour la simple et bonne raison que si vous changez de configuration ou de serveur, vous n'avez qu'une ligne à

modifier alors qu'en réalisant une telle adaptation au niveau de chaque règle, vous en auriez autant à maintenir.

6.4. Comprendre réellement le flag L(ast)

Il n'est pas rare de voir le rôle de l'option Last comprise de travers, confusion que l'on peut certainement attribuer au nom de ce drapeau. En effet, celui-ci n'empêche en rien une réécriture de boucler indéfiniment. Comme je l'ai déjà expliqué dans la partie Section 3.4, « Interactions entre les règles », il a pour seul but de mettre fin au processus **courant** de réécriture, c'est à dire appliquer immédiatement la règle au lieu de poursuivre la lecture des règles suivantes à la recherche d'autres correspondances. Mais une fois la règle appliquée, flag Last ou non, la nouvelle adresse subit à son tour toute potentielle réécriture, c'est un nouveau processus de réécriture totalement indépendant des précédents qui a lieu. Par conséquent, dans un contexte de répertoire (bloc <Directory> ou fichier .htaccess), si on retombe sur ce même répertoire, on en retrouve à nouveau ses règles, d'où une boucle.

En réalité, dans ce contexte dit de répertoire, c'est le module de réécriture, lui-même, qui implémente une sécurité de telle sorte que si le chemin courant est **strictement** égal à celui de destination de la règle (la valeur du deuxième paramètre de RewriteRule) alors Apache stoppe la réécriture. Faites l'essai par vous-mêmes, avec et sans l'option Last, vous ne constaterez aucune différence :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# en n'oubliant pas de préalablement créer le script
handler.php
RewriteRule .* handler.php
# puis retenter avec [L]
```

Histoire de prouver ce que j'avance et pour les curieux, voici le code en question (tiré d'une version 2.4 mais présent depuis toujours) :

```
/*
 * Fixup hook
 * [RewriteRules in directory context]
 */
static int hook_fixup(request_rec *r)
{
    /* ... */

    /* Check for deadlooping:
     * At this point we KNOW that at least one rewriting
     * rule was applied, but when the resulting URL is
     * the same as the initial URL, we are not allowed
to
     * use the following internal redirection stuff
because
     * this would lead to a deadloop.
     */
    if (ofilename != NULL && strcmp(r->filename,
ofilename) == 0) {
        rewritelog((r, 1, dconf->directory, "initial URL
equal rewritten"
                    " URL: %s [IGNORING REWRITE]", r-
>filename));
        return OK;
    }

    /* ... */
```

Dans la mesure où nous retrouverions scrupuleusement les mêmes règles, au final, que le drapeau Last soit ou non présent ne change quasiment rien. L'unique différence serait liée à l'ordre des règles : les règles précédant celle appliquée ne pouvant l'être à leur tour qu'au prochain processus de réécriture. Celles situées en-dessous pouvant être appliquées de suite, sans nécessiter un nouveau processus de réécriture.

L'option Last prend essentiellement tout son sens quand un nouveau répertoire est impliqué. Pour vous

en convaincre, effectuons une petite démonstration. Reproduisez tout d'abord la hiérarchie suivante :

```
/
    page.php
    sousrepertoire/
        .htaccess
        c.php
```

Le contenu de sousrepertoire/.htaccess est le suivant :

```
Options +FollowSymLinks -MultiViews
RewriteEngine On
RewriteBase /sousrepertoire/

RewriteRule ^a /page.php
RewriteRule p c.php
```

Faites un premier test avec l'URL http://localhost/sousrepertoire/abc. Puis ajoutez l'option Last à la première règle et recommencer.

Pourquoi le résultat est-il différent ? Explications :

- Avec Last, http://localhost/sousrepertoire/abc est réécrit de la sorte :
  - L'adresse courante, abc est transformée, en mémoire (ce n'est pas encore effectif), en /page.php du fait de la correspondance avec ^a ;
  - Last conduit à l'arrêt immédiat de la lecture des règles ;
  - Apache applique le nouveau chemin /page.php ;
  - http://localhost/page.php est la page finale car on suppose qu'il n'y a pas d'autres règles applicables (ni de fichier .htaccess à la racine ni de règles dans le httpd.conf).
- Quand, sans Last, http://localhost/sousrepertoire/abc, le cheminement est tout autre :

- abc est transformé dans un premier temps en /page.php (de par la correspondance de "abc" avec ^a) ;
- absence de flag Last, Apache poursuit la lecture des règles (/page.php n'est pas appliquée mais est le chemin en cours) ;
- /page.php est transformé en c.php (correspondance de "page.php" avec le motif p) ;
- il ne reste plus de règles à lire, Apache applique la dernière règle, l'url finale devient http://localhost/sousrepertoire/c.php.

6.5. Exemple de résolution d'une boucle infinie de réécriture

Je vais prendre l'approche naïve pour une application développée autour du motif MVC, c'est-à-dire avec un point d'entrée unique qui serait le script d'index. Quelqu'un pour qui la réécriture est nouveau sera probablement tenté de dire de tête : "on redirige tout sur index.php", soit :

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteRule .* /index.php
```

La personne va tester et constater que ça boucle indéfiniment (en réalité, non, Apache met fin à la requête par une erreur 500 à la dixième redirection interne). Le fait est que "tout" comprend le script index.php puisqu'il n'est fait aucune distinction sur ce qui existe physiquement sur le disque dur ou non.

Comment ne pas avoir cette boucle ? Ici, il existe plusieurs solutions :

- L'exclusion par une règle de non réécriture : il s'agit d'introduire une nouvelle règle pour intercepter l'exception (l'index). Elle doit être placée avant les autres et lui accoler l'option Last. Du fait qu'Apache lit

les règles de haut en bas dans un fichier .htaccess et qu'il ne poursuit pas la lecture si la règle trouvée est marquée Last, nous n'avons plus de boucle.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteRule ^(?:index\.php)?$ - [L]
RewriteRule .* /index.php
```

- L'exclusion par l'ajout d'une condition consiste à ajouter une condition (directive RewriteCond) à la règle problématique. Ce qui revient à écrire "tout réécrire vers l'index sauf l'index lui-même".

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

RewriteCond %{REQUEST_URI} ^/(?:index\.php)?$
RewriteRule .* /index.php
```

- Réécrire que ce qui n'existe pas physiquement : en couplant RewriteCond, les opérateurs -f et -d et la variable `REQUEST_FILENAME`, il est possible de réaliser une telle chose.

```
#Options +FollowSymLinks -MultiViews
RewriteEngine on

# Si la ressource demandée ne correspond pas à un
fichier
RewriteCond %{REQUEST_FILENAME} !-f
# Et si la ressource demandée ne correspond pas à un
répertoire
RewriteCond %{REQUEST_FILENAME} !-d
# La renvoyer sur index.php
RewriteRule .* /index.php
```

Pour l'exemple que j'ai choisi, cette dernière approche est à mon sens la meilleure dans la mesure où les fichiers statiques (css, js et autres images) ne sont pas inutilement renvoyés à PHP pour être servis.

6.6. Les différents niveaux de réécriture

En temps normal, Apache ne cherche les règles qu'à deux emplacements distincts dans l'ordre suivant :

1. du premier fichier .htaccess ou bloc <Directory> trouvé en partant du répertoire correspondant à la requête HTTP en remontant jusqu'à la racine du site ;
2. **et**, ensuite, celles de l'hôte virtuel répondant à la requête HTTP.

Si vous souhaitez qu'Apache considère également les règles du répertoire parent (pour un fichier .htaccess ou un bloc <Directory>) ou, pour un hôte virtuel, que ce dernier hérite de celles du serveur principal, vous devez ajouter une directive RewriteOptions ayant pour valeur `Inherit` à tous les endroits où vous souhaitez que le contexte parent soit consulté :

```
RewriteOptions Inherit
```

Les règles de niveaux supérieurs étant traitées **après** celles de niveaux inférieurs. Concrètement, il est possible d'aboutir à cette chaîne :

```
répertoire (.htaccess ou <Directory>) courant >
répertoire(s) parent(s) > hôte virtuel > serveur
principal
```

Cependant, Apache 2.4 introduit la valeur `InheritBefore` pour cette même directive RewriteOptions, inversant cet ordre : les règles de niveaux supérieurs sont alors évaluées avant celles de niveaux inférieurs.

Je rappelle que l'option END stoppe sur-le-champ toute réécriture. Par conséquent, les niveaux supérieurs (ou inférieurs si RewriteOptions est à `InheritBefore`) ne seront pas consultés. Quant au drapeau Last, c'est plus complexe pour les raisons que j'ai déjà pu évoquer : dans un contexte de répertoire, une fois la règle appliquée, nous retrouvons au moins en partie les

mêmes règles, à minima, celles au niveau de l'hôte virtuel. Le processus de réécriture ne peut véritablement être arrêté par Last que pour les règles au niveau de l'hôte virtuel ou du serveur principal.

7. Conclusion

Je suis conscient de ne pas avoir tout traité notamment certaines options de RewriteRule (S, N, C, CO, E, etc) ou encore RewriteMap. Le présent document est déjà bien assez long rien qu'en tentant d'aborder l'usage qu'il est communément fait de la réécriture avec Apache. J'espère que mon approche constituera une bonne base pour les lecteurs qui s'intéressent fraîchement au sujet ou l'approfondir mais s'avérera aussi un bon catalogue des différentes problématiques que l'on peut rencontrer.

Bien que n'ayant pas pu tout détailler, j'espère avoir pu démontrer que la réécriture est un véritable outil qui s'apparente à un couteau-suisse tant elle permet d'accomplir de tâches bien différentes.

Le procédé de réécriture d'URL n'est en rien propre à Apache, tout serveur HTTP intègre sa propre implémentation. Outre la question de la syntaxe, dans le fond, le principe est toujours le même. Par exemple, quelques parallèles avec Nginx :

- les directives RewriteCond s'écrivent sous forme de bloc if ;
- rewrite se comporte comme RewriteRule placée dans le fichier de configuration d'Apache au niveau d'un <VirtualHost> puisque c'est l'ensemble du chemin HTTP qui est testé ;
- la query string est toujours recopiée pour Nginx à moins d'utiliser la même solution qu'Apache : ajouter un point d'interrogation à la fin de la nouvelle destination à suivre ;
- la gestion des variables représentant la requête HTTP comme l'état du serveur, est totalement différente et indépendante du module de réécriture. Un gros avantage par rapport à Apache est que les valeurs composées, comme la query string, sont réellement parsées par Nginx et forment autant de variables, ce qui est plus commode.

Si vous deviez mémoriser quelques éléments à propos de la réécriture avec Apache, voici les points les plus importants :

- RewriteRule ne teste que la partie chemin d'une URL, pour tout le reste il faut s'orienter sur RewriteCond et la variable de réécriture adéquate ;
- par défaut, si la règle suivie ne définit pas de query string, alors celle d'origine, s'il y en a une, est recopiée. À l'inverse, si la règle appliquée définit une chaîne de requête, alors cette dernière remplace celle initiale ;
- les directives RewriteCond sont, par défaut, liées par un et logique ;
- les directives RewriteCond ne s'appliquent qu'à la règle qui suit. Elles ne se factorisent pas, il faut les répéter si nécessaire ;
- les chemins, au niveau d'une règle RewriteRule située dans un fichier .htaccess ne commencent jamais par un slash. C'est l'inverse pour les fichiers de configuration d'Apache à l'exception des blocs <Directory> ;
- le flag L(ast) n'est pas END : il n'empêche pas une boucle de réécriture contrairement à ce que beaucoup croient et colportent à tort ;
- il est recommandé de placer ses règles par ordre de spécificité décroissante, les motifs les moins "larges" en haut afin de limiter les conflits.

Liens :

- la documentation du module de réécriture : version 2.2 et 2.4 ;
- la documentation des expressions pour la syntaxe alternative de RewriteCond en versions 2.4.