

Подводные камни и распространенные ошибки

Как новые, так и старые пользователи могут столкнуться с ловушкой. Ниже мы описываем проблемы, с которыми мы часто сталкиваемся, а также объясняем, как их решить. В IRC-канале #nginx в Libera Chat мы часто видим эти проблемы.

В этом руководстве говорится

Наиболее частая проблема, с которой мы сталкиваемся, возникает, когда кто-то пытается просто скопировать и вставить фрагмент конфигурации из какого-либо другого руководства. Не все руководства неверны, но их пугающее количество.

Эти документы были созданы и проверены членами сообщества, которые работают напрямую со всеми типами пользователей NGINX. Этот конкретный документ существует только из-за большого количества распространенных и повторяющихся проблем, которые видят члены сообщества.

Моей проблемы нет в списке

Вы не видите здесь ничего, относящегося к вашей конкретной проблеме. Возможно, мы указали вам сюда не из-за конкретной проблемы, с которой вы столкнулись. Не просматривайте эту страницу и не думайте, что вас отправили сюда без причины. Вас отправили сюда, потому что здесь указано что-то, что вы сделали неправильно.

Когда дело доходит до поддержки многих пользователей по многим вопросам, члены сообщества не хотят поддерживать неисправные конфигурации. Исправьте свою конфигурацию, прежде чем обращаться за помощью. Исправьте свою конфигурацию, прочитав это. Не просматривайте ее.

Chmod 777

NEVER use 777. It might be one nifty number, but even in testing it's a sign of having no clue what you're doing. Look at the permissions in the whole path and think through what's going on.

To easily display all the permissions on a path, you can use:

```
namei -om /path/to/check
```

Root inside Location Block

BAD:

```
сервер {
    имя_сервера www.example.com;
    location / {
        root /var/www/nginx-default/;
        # [...]
    }
    location /foo {
```

 Поиск вики


Главная страница
Wiki

Приступая к работе

- Пример динамическогоSSI
- Полный пример настройки
- Еще один полный пример
- Простая балансировка нагрузки
- Обратный прокси с кэшированием
- SSL-загрузчик
- Ротация журналов
- Примеры блокировки сервера
- Пример PHP FastCGI
- PHP-FastCGI в Windows
- Отправка TurboGears Python с помощью FCGI
- Simple Ruby FCGI
- Django FastCGI
- FCGI Wrap
- FastCGI Example
- Java servers like Jetty, GlassFish and Tomcat
- Mono FCGI
- X-Accel
- NGINX Solution for Apache ProxyPassReverse
- Like Apache: .htaccess
- Разделение журналов ошибок для каждого виртуального хоста
- Пример прокси IMAP
- Использование Perl-скрипта в качестве серверной части

```

    root /var/www/nginx-default/;
    # [...]
}
location /bar {
    root /some/other/place;
    # [...]
}

```

Это работает. Размещение `root` внутри `location` блока будет работать, и это совершенно допустимо. Что не так, так это когда вы начинаете добавлять `location` блоки. Если вы добавите `root` к каждому блоку местоположения, то у `location` блока, который не соответствует, не будет `root`. Поэтому важно, чтобы директива `root` выполнялась до ваших `location` блоков, которые затем могут переопределить эту директиву, если им потребуется. Давайте рассмотрим хорошую конфигурацию.

хорошо:

```

сервер {
    имя_сервера www.example.com;
    root /var/www/nginx-по умолчанию/;
    location / {
        # [...]
    }
    location /foo {
        # [...]
    }
    location /bar {
        root /some/other/place;
        # [...]
    }
}

```

Директивы с несколькими индексами

плохо:

```

протокол HTTP {
    индекс index.php index.htm index.html;
    сервер {
        имя_сервера www.example.com;
        местоположение /
            индекс index.php index.htm index.html;
            # [...]
    }
    сервер {
        имя_сервера example.com;
        местоположение /
            индекс index.php index.htm index.html;
            # [...]
    }
    месте в/Foo {
        индекс index.php;
        # [...]
    }
}

```

Зачем повторять так много строк, когда это не нужно? Просто используйте `index` директиву один раз. Это должно произойти только в вашем `http{}` блоке, и оно будет унаследовано ниже.

хорошо:

```

протокол HTTP {
    индекс index.php index.htm index.html;
}

```

- аутентификации IMAP
- Использование PHP-скрипта на сервере Apache в качестве серверной части аутентификации IMAP
- Некорневой веб-путь
- Управление заголовками запросов
- Гибкость HTTP-заголовков
- Получаем значение заголовка
- Как я могу установить заголовок?
- Петрсикора
- Примеры
- Сценарии инициализации NGINX
- Ошибки проверки аппаратного балансировщика нагрузки
- XSendfile
- Предварительная загрузка Memcache
- Встроенный Perl Минимизирует JS
- Встроенный прокси-сервер на Perl Sitemaps
- Использование Forwarded заголовка
- CMS упростили
- Codeigniter
- Drupal
- Dokuwiki
- Elgg
- ExpressionEngine
- Geth (go-ethereum)
- iRedMail
- Почтальон
- MediaWiki
- MoinMoin
- MyBB
- Omeka-s
- Omeka
- osCommerce
- osTicket
- Список PHPList
- Matomo
- PmWiki
- Пилоны
- PygoCMS
- Qwebrick
- Redmine
- SilverStripe
- SPIP

```

сервер {
    имя_сервера www.example.com;
    местоположение / {
        # [...]
    }
}
сервер {
    имя_сервера example.com;
    местоположение / {
        # [...]
    }
    месте в/Foo {
        # [...]
    }
}
}

```

Использование if

Есть небольшая страница об использовании `if` инструкций. Она называется IfIsEvil, и вам действительно стоит ее просмотреть. Давайте рассмотрим несколько плохих вариантов использования `if`.

Смотреть также:

[If - это зло](#)

Имя сервера (Если)

плохо:

```

^/(.*) переписать
; $1 $ raw_domainустановить
{ ^ www \.(.+)* ~$ host ( если
; *.example.com example.сомимя_сервера
{ server$ $raw_domain/$1 постоянный;
}
# [...]
}

```

На самом деле здесь три проблемы. Первая - это `if`. Это то, что нас сейчас волнует. Почему это плохо? Вы читали "Если - это зло"? Когда NGINX получает запрос - независимо от того, какой поддомен запрашивается, будь то `www.example.com` или просто обычный `example.com` - эта `if` директива **всегда** оценивается. Поскольку вы запрашиваете у NGINX проверку заголовка `Host` для **каждого запроса**, это крайне неэффективно. Вам следует избегать этого. Вместо этого используйте две `server` директивы, подобные приведенному ниже примеру.

хорошо:

```

сервер {
    имя_сервера www.example.com;
    верните 301 $scheme://example.com$request_uri;
}
server {
    имя_сервера example.com;
    # [...]
}

```

Помимо упрощения чтения файла конфигурации. Такой подход снижает требования к обработке NGINX. Мы избавились от ложных `if`. Мы также используем `$scheme` который не жестко кодирует используемую вами схему URI, будь то `http` или `https`.

- [Symfony](#)
- Еще один работающий `symfony`
- [WordPress](#)
- [XenForo](#)
- [Yii](#)
- [Zend Framework](#)
- [Zenphoto](#)
- [Zope с помощью FastCGI](#)
- [b2evolution](#)
- [UVdesk](#)
- [Программное обеспечение для магазинов](#)
- [Bagisto](#)
- [Krayin](#)
- [QloApps](#)
- [If - это зло... при использовании в контексте местоположения](#)
- [Приступая к работе](#)
- [Установить](#)
- [Параметры установки и компиляции](#)
- [Командная строка](#)
- [Подводные камни и распространенные ошибки](#)
 - В этом руководстве говорится
 - Моя проблема нет в списке
 - `chmod 777`
 - Root внутри блока `Location`
 - Директивы с несколькими индексами
 - Использование `if`
 - Имя сервера (Если)
 - Проверьте, существует ли файл
 - Шаблон интерфейса веб-приложений
 - Передача неконтролируемых запросов в PHP
 - Путь FastCGI в имени файла скрипта
 - Облагающие налогом перезаписи
 - Отсутствует перезапись `http://`
 - Прокси Все

Проверьте, существует ли файл

Использование `if` для обеспечения существования файла ужасно. Это подло. Если у вас установлена какая-либо последняя версия NGINX, вам следует взглянуть на `try_files`, которая только что значительно упростила жизнь.

плохо:

```
сервер {
    root /var/www/example.com;
    местоположение / {
        if (!-f $request_filename) {
            break;
        }
    }
}
```

хорошо:

```
сервер {
    root /var/www/example.com;
    местоположение / {
        try_files $uri $uri/ /index.html;
    }
}
```

Что мы изменили, так это то, что мы пытаемся увидеть, `$uri` существует ли, не требуя `if`. Использование `try_files` означает, что вы можете протестировать последовательность. Если `$uri` не существует, попробуйте `$uri/`, если этого не существует, попробуйте найти запасной вариант.

В этом случае, если `$uri` файл существует, отправьте его. Если нет, проверьте, существует ли этот каталог. Если нет, то перейдите к отправке, `index.html` в существовании которой вы убедитесь. Это загружено, но так просто! Это еще один пример, когда вы можете полностью исключить `If`.

Шаблон интерфейса веб-приложений

Проекты “Front Controller Pattern” популярны и используются во многих самых популярных программных пакетах PHP; Но многие примеры более сложны, чем это необходимо. Для Drupal, Joomla и т.д. Просто используйте это:

```
try_files $uri $uri/ /index.php?q=$uri$args;
```

Примечание - имена параметров различаются в зависимости от используемого пакета. Например:

- “`q`” – это параметр, используемый Drupal, Joomla, WordPress
- “страница” используется CMS Made Simple

Некоторым программам даже не нужна строка запроса, и они могут считывать данные из `REQUEST_URI`. Например, WordPress поддерживает это:

```
try_files $uri $uri/ /index.php;
```

Если вас не волнует проверка существования каталогов, вы можете пропустить ее, удалив `$uri/`.

Конечно, ваш опыт может отличаться, и вам может потребоваться что-то более сложное в зависимости от ваших потребностей, но для базовых сайтов это подойдет идеально. Вы всегда должны начинать с простого и строить исходя из этого.

- Используйте `$request_filename` для `SCRIPT_FILENAME`
 - Изменения конфигурации Не отражены
 - VirtualBox
 - Отсутствующие (исчезающие) HTTP-заголовки
 - Не используются стандартные корневые папки документов
 - Использование корневого каталога документов по умолчанию
 - Использование имени хоста для разрешения адресов
 - Использование SSLv3 с HTTPS
 - Использование `try_files $uri` директивы с `alias`
 - Некорректный `return` контекст
 - Отладка
 - Оптимизация
 - Установка на Solaris 10u5
 - Тестирование NGINX
 - Запуск NGINX как SMF-сервиса
 - Установка на Solaris 11
 - Скрипт запуска
 - Установка и настройка NGINX / Mongrel в OpenBSD с поддержкой Rails
 - Установка и настройка NGINX
 - Инструкции по полному стеку
 - Готовые конфигурации
 - Другие примеры
 - Инструменты
 - Расширенные разделы
 - Учебные ресурсы
- Сообщество
- NGINX 3 Сторонние модули
Расширение NGINX

Передача неконтролируемых запросов в PHP

Многие примеры конфигураций NGINX для PHP в Интернете рекомендуют передавать интерпретатору PHP каждый URI, заканчивающийся на `.php`. Обратите внимание, что это представляет серьезную проблему безопасности для большинства настроек PHP, поскольку может допускать выполнение произвольного кода третьими сторонами.

Раздел проблем обычно выглядит следующим образом:

```
\.php *~ location$ {
    fastcgi_pass серверная часть;
    # [...]
}
```

Здесь каждый запрос, заканчивающийся на `.php`, будет передаваться на серверную часть FastCGI. Проблема заключается в том, что конфигурация PHP по умолчанию пытается угадать, какой файл вы хотите выполнить, если полный путь не ведет к реальному файлу в файловой системе.

Например, если сделан запрос для `/forum/avatar/1232.jpg/file.php` которого не существует, но если `/forum/avatar/1232.jpg` существует, интерпретатор PHP обработает `/forum/avatar/1232.jpg` вместо этого. Если это содержит встроенный PHP-код, этот код будет выполнен соответствующим образом.

Этого можно избежать следующими способами:

- Устанавливается `cgi.fix_pathinfo=0` в `php.ini`. Это заставляет интерпретатор PHP пробовать только указанный буквальный путь и останавливать обработку, если файл не найден.
- Убедитесь, что NGINX передает для выполнения только определенные PHP-файлы:

```
(file_a|file_b|file_c)\.php *~ location$ {
    fastcgi_pass серверная часть;
    # [...]
}
```

- Специально отключите выполнение PHP-файлов в любом каталоге, содержащем загруженные пользователем файлы:

```
location /uploaddir {
    location ~ \.php$ {return 403;}
    # [...]
}
```

- Используйте директиву `try_files` для фильтрации проблемных условий:

```
\.php *~ location$ {
    try_files $uri =404;
    fastcgi_pass серверная часть;
    # [...]
}
```

- Используйте вложенное расположение, чтобы отфильтровать проблемное условие:

```
\.php *~ location$ {
    location ~ \..*/.*\.php$ {return 404;}
    бэкенд fastcgi_pass;
    # [...]
}
```

ОСТАВАЙТЕСЬ В КУРСЕ
СОБЫТИЙ 

Редактировать эту страницу

 Показать на GitHub

 Редактировать на GitHub

So many guides out there like to rely on absolute paths to get to your information. This is commonly seen in PHP blocks. When you install NGINX from a repository, you'll usually wind up being able to toss `include fastcgi_params;` in your config. This is a file located in your NGINX root directory which is usually around `/etc/nginx/`.

GOOD:

```
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
```

BAD:

```
fastcgi_param SCRIPT_FILENAME /var/www/yoursite.com/$fastcgi_script_name;
```

Где `$document_root` установлен? Он устанавливается директивой `root`, которая должна быть в вашем серверном блоке. Вашей директивы `root` там нет? Смотрите первый подводный камень.

Облагающие налогом перезаписи

Не расстраивайтесь, здесь легко запутаться в регулярных выражениях. На самом деле, это так легко сделать, что мы должны приложить усилия, чтобы поддерживать их в чистоте. Проще говоря, не добавляйте `cruff`.

плохо:

```
^/(.*) переписать$ http://example.com /$ 1 постоянно;
```

хорошо:

```
переписать ^ http://example.com$request_uri? постоянно;
```

лучше:

```
возврат 301 http://example.com$request_uri;
```

Посмотрите выше. Затем вернитесь сюда. Затем вверх и вернитесь сюда. ОК. Первая Посмотрите выше. Затем вернитесь сюда. Затем вверх и вернитесь сюда. ОК. Первая перезапись фиксирует полный URI за вычетом первой косой черты. Используя встроенную переменную `$request_uri`, мы можем эффективно избежать какого-либо захвата или сопоставления вообще.

Отсутствует перезапись `http://`

Проще говоря, перезаписи относительны, если только вы не сообщите NGINX, что это не так. Сделать перезапись абсолютной очень просто. Добавьте схему.

плохо:

```
переписать ^ example.com постоянно;
```

хорошо:

```
переписать ^ http://example.com постоянно;
```

Из приведенного выше вы увидите, что все, что мы сделали, это добавили `http://` к переписыванию. Это просто, доступно и эффективно.

Прокси Все

плохо:

```
server {
    имя_сервера _;
    root /var/www/site;
    location / {
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_na
me;
        fastcgi_pass unix:/tmp/phpcgi.socket;
    }
}
```

Отвратительно. В этом случае вы передаете ВСЕ на PHP. Почему? Apache может это сделать, но вам это не нужно. Директива `try_files` существует по удивительной причине: она проверяет файлы в определенном порядке. NGINX может сначала попытаться обработать статическое содержимое, а если это не удастся, движется дальше. Это означает, что PHP вообще не задействован. НАМНОГО быстрее. Особенно, если вы отправляете изображение размером 1 МБ через PHP несколько тысяч раз, а не отправляете его напрямую. Давайте посмотрим, как это сделать.

хорошо:

```
server {
    имя_сервера _;
    root /var/www/site;
    location / {
        try_files $uri $uri/ @proxy;
    }
    location @proxy {
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_n
ame;
        fastcgi_pass unix:/tmp/phpcgi.socket;
    }
}
```

Также ХОРОШО:

```
сервер {
    имя_сервера _;
    root /var/www/site;
    местоположение / {
        try_files $uri $uri/ /index.php;
    }
    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_na
me;
        fastcgi_pass unix:/tmp/phpcgi.socket;
    }
}
```

Легко, не так ли? Проверьте, существует ли запрошенный URI и может ли он обрабатываться NGINX. Если нет, проверьте, можно ли обслуживать этот каталог. Если нет, то передайте его своему прокси. Только когда NGINX не может обслуживать запрошенный URI напрямую, возникают накладные расходы вашего прокси-сервера.

Подумайте, сколько ваших запросов относится к статическому контенту (изображениям, css, javascript и т.д.). Вероятно, вы только что сэкономили много накладных расходов.

Используйте `$request_filename` для

SCRIPT_FILENAME

Используйте `$request_filename` вместо `$document_root$fastcgi_script_name`.

Если вы используете `alias` директиву с `$document_root$fastcgi_script_name`, `$document_root$fastcgi_script_name` вернет неправильный путь.

плохо:

```
местоположение /api/ {
    индекс index.php index.html index.htm;
    псевдоним /app/www/;
    местоположение ~* "\.php$" {
        try_files $uri =404;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script
        _name;
    }
}
```

Запрос `/api/testing.php`:

- `$document_root$fastcgi_script_name == /app/www//api/testing.php`
- `$request_filename == /app/www/testing.php`

Запрос `/api/`:

- `$document_root$fastcgi_script_name == /app/www//api/index.php`
- `$request_filename == /app/www/index.php`

И если вы используете `$request_filename`, вы должны установить индекс с помощью `index` директивы, `fastcgi_index` не сработает.

хорошо:

```
местоположение /api/ {
    индекс index.php index.html index.htm;
    псевдоним /app/www/;
    местоположение ~* "\.php$" {
        try_files $uri =404;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME $request_filename;
    }
}
```

Изменения конфигурации Не отражены

Кэш браузера. Ваша конфигурация может быть идеальной, но вы будете месяц сидеть и биться головой о цементную стену. Проблема в кэше вашего браузера. Когда вы что-то загружаете, ваш браузер сохраняет это. В нем также хранится способ обработки этого файла. Если вы играете с `types{}` блоком, вы столкнетесь с этим.

Исправление:

- В Firefox нажмите `Ctrl + Shift + Delete`, проверьте кэш, нажмите Очистить сейчас. В любом другом браузере просто спросите свою любимую поисковую систему. Делайте это после каждого изменения (если только вы не знаете, что в этом нет необходимости), и вы избавите себя от множества головных болей.
- Используйте `curl`.

Если это не работает, и вы запускаете NGINX на виртуальной машине в VirtualBox, возможно, проблема в `sendfile()`. Просто закомментируйте директиву `sendfile` или установите для нее значение “выкл.”. Директива, скорее всего, находится в вашем файле `nginx.conf`:

отправка файла отключена;

Отсутствующие (исчезающие) HTTP-заголовки

Если вы явно не зададите `underscores_in_headers on`, NGINX автоматически удалит HTTP-заголовки с подчеркиванием (которые абсолютно допустимы в соответствии со стандартом HTTP). Это сделано для предотвращения двусмысленностей при сопоставлении заголовков с переменными CGI, поскольку во время этого процесса как тире, так и подчеркивания сопоставляются с подчеркиваниями.

Not Using Standard Document Root Locations

Some directories in any file system should never be used for hosting data from. These include `/` and `root`. You should never use these as your document root.

Doing this leaves you open to a request outside of your expected area returning private data.

NEVER DO THIS!!! (yes, we have seen this)

```
server {
    root /;

    location / {
        try_files /web/$uri $uri @php;
    }

    location @php {
        # [...]
    }
}
```

Когда делается запрос для `/foo`, запрос передается в `php`, потому что файл не найден. Это может казаться нормальным, пока запрос не будет выполнен для `/etc/passwd`. Да, вы только что предоставили нам список всех пользователей на этом сервере. В некоторых случаях на сервере NGINX даже настроен запуск `workers` от имени `root`. Да, теперь у нас есть ваш список пользователей, а также хэши паролей и способы их хэширования. Теперь у нас есть ваш почтовый ящик.

[Иерархия файловой системы](#) определяет, где должны находиться данные. Вам обязательно следует это прочитать. Короткая версия заключается в том, что вы хотите, чтобы ваш веб-контент существовал в любом из `/var/www/`, `/srv`, `/usr/share/www`.

Использование корневого каталога документов по умолчанию

Пакеты NGINX, существующие в Ubuntu, Debian или других операционных системах, как простые в установке пакеты, часто предоставляют файл конфигурации по умолчанию в качестве примера методов настройки и часто включают корневой каталог документа для хранения базового HTML-файла.

Большинство этих систем упаковки не проверяют, изменены ли файлы или существуют ли они в корневом каталоге документов по умолчанию, что может привести к потере кода при обновлении пакетов. Опытные системные администраторы знают, что данные в корневом

каталоге документов по умолчанию не должны оставаться нетронутыми во время обновлений.

обновлении пакетов. Опытные системные администраторы знают, что данные в корневом каталоге документов по умолчанию не должны оставаться нетронутыми во время обновлений.

Вам не следует использовать корневой каталог документов по умолчанию для любых файлов, критически важных для сайта. Нет никаких оснований ожидать, что корневой каталог документов по умолчанию останется нетронутым системой, и существует чрезвычайно высокая вероятность того, что критически важные для вашего сайта данные могут быть потеряны при обновлении пакетов NGINX для вашей операционной системы.

Использование имени хоста для разрешения адресов

плохо:

```
upstream {
    сервер http://someserver;
}

сервер {
    прослушивать имя_моего_хостинга:80;
    # [...]
}
```

Вам никогда не следует использовать имя хоста в директиве `listen`. Хотя это может сработать, оно сопряжено с большим количеством проблем. Одна из таких проблем заключается в том, что имя хоста может не разрешаться во время загрузки или во время перезапуска службы. Это может привести к тому, что NGINX не сможет выполнить привязку к нужному TCP-сокету, что вообще предотвратит запуск NGINX.

Более безопасная практика - знать IP-адрес, к которому необходимо привязать, и использовать этот адрес вместо имени хоста. Это избавляет NGINX от необходимости искать адрес и устраняет зависимости от внешних и внутренних преобразователей.

Эта же проблема относится и к расположениям, расположенным выше по потоку. Хотя не всегда возможно избежать использования имени хоста в блоке, расположенном выше по потоку, это плохая практика, и для предотвращения проблем потребуется тщательное рассмотрение.

хорошо:

```
входящий поток {
    сервер http://10.48.41.12;
}

сервер {
    прослушивание 127.0.0.16:80;
    # [...]
}
```

Использование SSLv3 с HTTPS

Из-за уязвимости POODLE в SSLv3 рекомендуется не использовать SSLv3 на ваших сайтах с поддержкой SSL. Вы можете очень легко отключить SSLv3 с помощью этой строки и вместо этого предоставить только протоколы TLS:

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
```

Using the `try_files $uri directive with alias`

Симптомы этого трудно диагностировать: обычно кажется, что вы все сделали правильно, и все же вы получаете загадочные ошибки 404. Почему? что ж, включение ведения журнала ошибок на уровне отладки показывает, что `try_files` добавляется `$uri` к пути, уже установленному с помощью `alias`. Это связано с ошибкой в NGINX, но не волнуйтесь – решение простое! Пока ваша `try_files` строка имеет вид `try_files $uri $uri/ =404;`, вы можете просто удалить `try_files` строку без существенного негативного эффекта. Вот пример, в котором вы не можете использовать `try_files`.

плохо:

```
местоположение ~^/~(?<пользователь>[^/]*)/(?<страница>.*)$ {  
    псевдоним /home/$user/public_html/$страница;  
    try_files $uri $uri/ =404;  
}
```

хорошо:

```
местоположение ~^/~(?<пользователь>[^/]*)/(?<страница>.*)$ {  
    псевдоним /home/$user/public_html/$страница;  
}
```

Единственное предостережение заключается в том, что это обходное решение не позволяет вам использовать `try_files` для предотвращения атак PATH_INFO . Альтернативные способы смягчения этих атак см. в разделе [Передача неконтролируемых запросов в PHP](#) выше.

Также обратите внимание, что `snippets/fastcgi-php.conf` файл, поставляемый некоторыми дистрибутивами Linux, может потребоваться отредактировать для удаления `try_files` директивы, если она включена в `location` блок с `alias`.

Некорректный `return` контекст

Директива `return` применяется только внутри самого верхнего контекста, в котором она определена. В этом примере:

```
сервер {  
    местоположение /a/ {  
        try_files test.html =404;  
    }  
  
    возвращает 301 http://example.org;  
}
```

Запрос на `/a/test.html` вернет значение 301. Чтобы это сработало должным образом, оберните второй блок внутри `location /`:

запрос на /a/test.html вернет значение 301. Тогда это сработало должным образом, оберните второй блок внутри `location /`:

```
сервер {  
    расположение /a/ {  
        try_files test.html =404;  
    }  
  
    местоположение / {  
        возврат 301 http://example.org;  
    }  
}
```

Продукты

- [NGINX Plus](#)
- [Контроллер NGINX](#)
- [Защита приложений](#)
- [NGINX](#)
- [Модуль NGINX](#)
- [NGINX усиливает](#)
- [NGINX на Github](#)
- [NGINX с открытым исходным кодом](#)
- [Модуль NGINX](#)
- [NGINX усиливает](#)
- [Входной контроллер](#)
- [NGINX Kubernetes](#)
- [Эталонная архитектура микросервисов NGINX](#)

Решения

- [АЦП / Балансировка нагрузки](#)
- [Микросервисы](#)
- [Облако](#)
- [Безопасность](#)
- [Производительность веб- и мобильных устройств](#)
- [Управление API](#)

Ресурсы

- [Документация](#)
- [Электронные книги](#)
- [Вебинары](#)
- [Технические характеристики](#)
- [Истории успеха](#)
- [Блог](#)
- [Вопросы и ответы](#)
- [Учиться](#)
- [Глоссарий](#)
- [Поддержка](#)
- [Профессиональные услуги](#)
- [Обучение](#)

Партнеры

- [Веб-сервисы Amazon](#)
- [Облачная платформа Google](#)
- [IBM](#)
- [Microsoft Azure](#)
- [Red Hat](#)
- [Найти партнера](#)
- [Сертифицированная модульная программа](#)

Свяжитесь с нами**ОСТАВАЙТЕСЬ В КУРСЕ СОБЫТИЙ**