?????? ?????? ??????? RPM

error: Installed (but unpackaged) file(s) found

Ошибка означает, что в новой версии пакета появились новые файлы, отсутствующие ранее, и rpmbuild не знает, что с ними делать (и нужны ли ли вообще).

Если вы полагаете, что эти файлы нужны (а это так в 99.9% случаев), необходимо добавить их в секцию %files spec-файла. Если у вас есть несколько подпакетов, то надо сначала определить - к какому из них эти файлы относятся. Универсального рецепта здесь нет, однако есть несколько правил:

- файлы в директориях `/usr/include`, `/ust/lib(64)/pkgconfig`, добавляются в devel-пакеты
- файлы библиотек в директориях `/lib(64)` и `/usr/lib(64)`, оканчивающиеся на суффикс .so (например, `libfoo.so`), также добавляются в devel-пакеты
- файлы библиотек в директориях `/lib(64)` и `/usr/lib(64)`, оканчивающиеся на цифру (например, `libfoo.so.1`), добавляются в пакеты с соответсвующими библиотеками. Согласно правилам РОСЫ, каждая библиотека должна упаковываться в отдельный подпакет, соотевтсвующий ее имени и значению SONAME (как правило, это цифра после суффикса .so - так что файлы `libfoo.so.1` и `libfoo.so.1.2` должны находиться в пакете **lib(64)foo1**). Часто ошибка "Installed but unpackaged files found) в пакетах с библиотеками вызвана изменением значения SONAME - например, вместо `libfoo.so.1` в новой версии пакета появился файл `libfoo.so.2`. В этом случае вы также дополнительно увидите ошибку "Cannot find file or directory", сообщающую, что отсутсвует файл `libfoo.so.1`. Для исправления сборки в такой ситуации достаточно изменить значение макроса major в начале spec-файла:

```
%define major 2
```

Помните, что при добавлении файлов в секции ***%files*** принято заменять некоторые стандартные пути макросами (например вместо `/usr/lib` и `/usr/lib64` необходимо использовать макрос ***%{_libdir***}, вместо `/usr/share/man` - ***%{_mandir***} и так далее). Более полный список можно посмотреть с [скрипте из Updates Builder].

File not found

Ошибка означает, что в новой версии пакета отсутсвуют некоторые файлы, присутствувавшие ранее. Причин для этого может быть несколько:

- в новой версии действительно нет таких файлов - в этом случае надо просто убрать их описание из секции ***%files***
- в новой версии файлы переименованы либо перемещены в другое место. В этом случае вы также увидите ошибку "Installed (but unpackaged) file(s) found", как в случае с библиотеками (см. выше). В случае с библиотеками для исправления сразу обеих ошибок необходимо изменить значение переменной major в начале spec-файла. В общем же случае необходимо исправить секцию ***%files***, чтобы она соответсвовала новым реалиям.
- отсутсвующие файлы собираются или не собираются в зависимости от параметров окружения и сборки - опций компиляции или установленных в сборочной среде пакетов (описанных в BuildRequires). Возможно, в новой версии пакета надо использовать какие-то новые опции сборки либо добавить дополнительные сборочные зависимости для получения этих файлов. Выявить такие ситуации можно на основе анализа журналов сборки и вывода команд наподобие configure или cmake, которые сообщают, какие опцилнальные возможности будут включены при сборке, а какие - нет. Например, **squid** может собираться с поддержкой аутентификации SASL и без нее. В первом случае в пакете будет присутсвовать файл basic_sasl_auth, во втором его не будет. Для ключения/отключения SASL необходимо добавить/удалить значение SASL из параметра --enable-auth-basic команды `configure`, а также добавить сборочную зависимость (BuildRequires) от **sasl-devel**.

Помните, что при добавлении файлов в секции ***%files*** принято заменять некоторые стандартные пути макросами, а также что при описании файлов в секции ***%files*** могут использоваться символы '?' и '*' (означающие один произвольный символ и любое количество произвольных символов соответсвенно).

cp: cannot stat '<some_file>': No such file or directory

Ошибка означает, что rpmbuild не смог найти файл <some_file>, помеченный как ***%doc*** в секции %files вашего пакета. Возможно, этот файл был переименован (в этом случе вы также увидите ошибку "Installed (but unpackaged) file(s) found") - в этом случае надо изменить имя файла на новое (например, `README` могут переименовать в `README.md`). Если неупакованного файла с близким именем не появилось, то значит старый файл в новой версии отсутсвует и его определение в ***%doc*** необходимо просто удалить.

Reversed (or previously applied) patch detected

Ошибка означает, что применявшийся к старой версии пакета патч по крайней мере частично уже применен в новой версии. Обратите внимание, что команда `patch` делает такой вывод на основе попытки применить только первую часть патча! Если патч сложный и затрагивает несколько файлов или несколько мест одного файла, то необходимо вручную проверить, какие из этих изменений уже есть в новой версии, а каких нет. Для этого можно попробовать применить патч к новой версии вручную, отвечая "n" на вопрос "Assume '-R'" и "y" на предложени продолжить применять остальные части патча.

Если все изменения из патча действительно уже присутсвуют в новой версии, то патч можно спокойно удалить (физически из Git-репозитория, а также из spec-файла). Если же нет, то необходимо определить - нужны ли в новой версии оставшиеся изменения и если да, то переделать патч соответствующим образом. Такой анализ уже требует некоторых познаний в том, что именно делает патч.

/var/tmp/rpm-tmp.xxxxx: line yy: cd: <some_folder_name>: No such file or directory

Ошибка означает, что rpmbuild не может определить, в какую директорию ему перейти после распаковки архива с исходным кодом. Имя директории указывается в опции ***-n*** макроса ***%setup*** в секции ***%prep***. Если эта опция отсутсвует, то подразумевается использование "-n %{name}-%{version}". Для исправления ошибки необходимо посмотреть, как расположены файлы в новой версии архива с исходным кодом - обычно такой архив содержит директорию вида "<имя_программы>-<версия>", но время от времени разработчики могут что-то изменять (например, переименовывать директорию просто в <имя_программы>. Для исправления ошибки необходимо соответсвующим образом исправить значение опции "-n" макроса ***%setup***.

empty-debug-info ? debuginfo-without-sources

Ошибки означают, что rpmbuild не смог должным образом сформировать подпакет с отладочной информацией. Две основные причины такого поведения:

- в пакете нет исполнимых файлов или библиотек. Если в пакете при этом вообще нет архитектурно-зависимых файлов (т.е. пакеты для разных архитектур имею абсолютно одинаковое содержимое), то необходимо объявить пакет как независимый от архитектуры, добавив в заголовок spec-файла декларацию "BuildArch: noarch". Если же архитектурно-зависимые файлы все-таки есть (например, какие-то данные могут быть специфичны для каждой арзитектуры; другой типичный пример - проприетарное приложение, которое уже поставляется в виде бинарных файлов бех отладочной информации), то необходимо отключить генерацию debug-пакетов, сбросив определение макроса ***debug_package*** в начале spec-файла:

```
%define debug_package %{nil}
```

- скрипты сборки приложения в пакете устроены таким образом, что сразу удаляют отладочную информацию из собранных файлов (например, явно вызывают команду strip либо просто не передают опцию "-g" компилятору). В таких случаях необходимо посмотреть, как заставить скрипты сборки оставить отладочную информацию. Иногда этого можно добиться с помощью переменных среды, а иногда может потребоваться патч, изменяющий сборочные скрипты. Если никакие способы не помогают, то можно отключить генерацию debug-пакетов, как описано выше, но делать так не рекомендуется - эти пакеты очень полезны для анализа ошибок в случае падений приложения.

Остановимся подробнее на втором пункте. Большинство приложений позволяют настраивать передаваемые компилятору опции при старте сборки - например, при вызове **`configure`** или **`cmake`**. Для генерации отладочной информации для подобных приложений вам не нужно прилагать особых усилий - достаточно вместо прямого вызова **`configure`** или **`cmake`** использовать соответствующие макросы **`rpm`** (в нашем примере - ***%configure2_5x*** или ***%cmake*** соответственно). Если использование макросов не помогает (либо в проекте не используются стандартные средства типа GNU Autotools или CMake), необходимо изучить схему сборки и посмотреть - нельзя ли на нее повлиять как-то еще - например, специфическими опциями либо переменными окружения. Типичными переменными, которые могут повлиять на сборку, являются ***CFLAGS***, ***CXXFLAGS*** и ***CPPFLAGS***. Мы рекомендуем выставить эти переменные в макрос ***%optflags***; при этом может оказаться необходимым перед использованием ***%%optflags*** вызвать макрос ***%%setup_compile_flag*** (например так сделано в пакете [slock].

Обратите внимание на использование кавычек при выставлении переменных среды - они необходимы, поскольку макрос разворачивается в большой набор опций, разделенных пробелами.

Бывают ситуации, когда повлиять на сборку извне нельзя никак - в скриптах сборки или файлах типа Makefile инструкции сборки "забиты" намертво. В таких случаях необходимо подготовить патч, добавляющий во все вызовы компилятора опцию ***-g***. Помимо опции ***-g*** у компилятора, повлиять на отладочную информацию может компоновщик - например, его

опции ***-s*** и ***-S***, удаляющие подобные данные из результирующих бинарных файлов. Если такие опции явно передаются компоновщику в скриптах сборки либо файлах Makefile, то необходимо подготовить патч, убирающий их - например, так сделано в пакете [kicad].

Error: Can't locate Some/Perl/Module.pm in @INC

Ошибка означает, что для сборки необходим Perl-модуль `Some/Perl/Module.pm`, который не был обнаружен в сборочном окружении.

Для исправления ошибки необходимо добавить сборочную зависимость от такого модуля в spec-файл:

```
BuildRequires: perl(Some::Perl::Module)
```

Предварительно необходимо убедиться, что такой модуль вообще существует - попробовать его установить с помощью dnf (в rosa2019.1 и новее) либо urpmi (в rosa2016.1):

```
# dnf install 'perl(Some::Perl::Module)'
```

```
# urpmi 'perl(Some::Perl::Module)'
```

Если urpmi скажет, что не нашел нужного пакета, то необходимо сначала добавить пакет с модулем в репозитории РОСЫ.

Обратите внимание, что необходимый модуль может находиться в репозитории Contrib, в то время как вы собираете пакет в репозитории Main, Non-free или Restricted. В таком случае Contrib при сборке не используется и требуемый модуль необходимо перенести в репозиторий Main.

Package(s) suggested but not available

Данная ошибка может встречаться при сборке расширений языка R. Она означает, что у собираемого пакета есть необязательная зависимость от каких-то других модулей, которые в сборочной среде отсутствуют. В идеале для каждой такой зависимости необходимо прописать BuildRequires и Requires - например, если ошибка относится к модулю "foo", то нужны зависимости от R-foo:

```
BuildRequires: R-foo
Requires: R-foo
```

Предварительно необходимо убедиться, что такой пакет вообще есть в репозиториях - вывод команды "urpmq R-foo" должен быть непуст. Если пакета нет, то желательно его сначала собрать и добавить в репозитории. Однако при сборке может оказаться, что он зависит от того пакета, который вы собирали до этого и для которого вы собственно и хотите добавить зависимость R-foo. В этом случае первый пакет можно собрать и без R-foo, закомментировав при этом команду "%{_bindir}/R CMD check %{packname}" в секции %check. После этого можно будет уже собрать R-foo и вернуться к исходному пакету, добавив в него зависимость от R-foo и включив тесты в секции %check

ERROR: dependency(ies) not available

Данная ошибка также может встречаться при сборке расширений языка R, но в отличие от предыдущей, здесь речь идет об обязательных зависимостях сборки. Без них пакет не может быть собран, поэтому вам необходимо добавить в spec-файл соответсвующие BuildRequiers и Requires, а при необходимости предварительно собрать нужные пакеты в репозитории РОСЫ.

hunk FAILED -- saving rejects

Данная ошибка означает, что один из патчей (какой именно - указано в выводе rpmbuild) не может быть применен без изменений к новой версии программы. Обычно это означает, что патч необходимо переделать (предварительно определив - нужен ли он еще), что требует определенной квалификации. Однако иногда такая ошибка возникает из-за "строгости" rpmbuild при применении патчей и проблемный патч может быть исправлен в автоматическом режиме с помощью утилиты rediff_patch.

Для этого достаточно склонировать себе Git-репозиторий, перейти в склонированную папку, поместить туда архив с исходным кодом новой версии и запустит rediff_patch, передав ей нужный патч к качестве аргумента:

```
$ abf get myrepo/myproject
$ cd myproject
$ wget http://project-upstream.org/new-version.tgz
$ rediff_patch <some_patch_to_rediff>
```

Если все сложится удачно, то радом со старым патчем "some_patch_to_rediff" появится новый - "some_patch_to_rediff.new" Если же что-то пойдет не так, то в текущей директории останутся папка "rediff_patch" с подпапками вида "myproject.orig" и "myproject" - содержащие соответсвенно оригинальный исходный код и исходный код, получившийся после попытки применить патч. Во второй папке вы найдете файлы с расширениями *rej - это куски патчей, которые применить не удалось.

package 'foo' not found

Аналогична следующей ошибке

"No package '.*' found"

Такая ошибка возникает, если пакет проверяет необходимые сборочные зависимости с помощью `pkgconfig` и не обнаруживает одну из них.

Для исправления ошибки достаточно добавить нужную зависимость сборки в spec-файл:

```
BuildRequires: pkgconfig(foo)
```

Предварительно необходимо убедиться, что такая зависимость может быть разрешена - попробовать ее установить с помощью `urpmi` :

```
# urpmi 'pkgconfig(foo)'
```

Если urpmi скажет, что не нашел нужного пакета, то необходимо сначала добавить пакет с соответсвующей библиотекой (как правило, она и называется **libfoo**) в репозитории РОСЫ.

Обратите внимание, что необходимый пакет может находиться в репозитории Contrib, в то время как вы собираете пакет в репозитории Main, Non-free или Restricted. В таком случае Contrib при сборке не используется и требуемый пакет необходимо перенести в репозиторий Main.

/usr/bin/ld: cannot find -lfoo

Данная ошибка возникает при линковке уже собранных объектных файлов и означает, что линкер не смог найти библиотеку "foo". Для исправления ошибки достаточно добавить зависимость от библиотеки в spec-файл. Чтобы определить, как именно эту зависимость прописать, необходимо сначала поискать devel-пакет для библиотеки **libfoo** (или **lib64foo** в 64битной системе) с помощью urpmq:

```
# urpmq -a libfoo
  lib64foo1
  lib64foo-devel
```

Интересующий нас пакет - тот, что оканчивается на -devel. Посмотрим, что он предоставляет:

```
# urpmq --provides libfoo-devel
  lib64foo-devel: devel(libfoo(64bit))
  lib64foo-devel: lib64sasl-devel[== 2.1.25]
  lib64foo-devel: libsasl-devel[== 2.1.25]
  lib64foo-devel: libfoo-devel[== 2.1.25]
  lib64foo-devel: sasl-devel[== 2.1.25-7]
  lib64foo-devel: lib64foo-devel[== 2.1.25-7:2014.1]
  lib64foo-devel: pkgconfig(foo)[== 2.1.25-7]
```

Из данного набора необходимо выбрать что-то одно. В POCE отдается предпочтение зависимостям вида **pkgconfig(...)**, так что в нашем примере в spec-файл необходимо добавить следующую строку:

```
BuildRequires: pkgconfig(foo)
```

Если pkgconfig(foo) не окажется в выводе urpmq, то надо добавить зависимость от foo-devel, а если и ее нет - то от libfoo-devel.

Build.PL: No such file or directory

Такая ошибка означает, что предыдущая версия пакета собиралась с помощью скрипт Build.PL, отсутсвующего в новой версии. Часто встречающаяся ситуация - замена Build.PL на Makefile.PL. Если в архиве с исходным кодом новой версии есть скрипт Makefile.PL, то в spec-файле необходимо произвести следующие замены:

- "/Build.PL installdirs=vendor" -> "Makefile.PL INSTALLDIRS=vendor"
- ./Build install destdir=%{buildroot} -> %makeinstall_std
- perl Build.PL install destdir=%{buildroot} -> %makeinstall_std
- ./Build test -> %make test
- ./Build -> %make

Для примера можно посмотреть на пакет ...

Если же скрипта Makefile.PL в новой версии также нет, то необходимо смотреть - а что, собственно, есть - файлы CMake, configure или готовый Makefile и модифицировать spec-файл в соответствии с используемой системой сборки.

Makefile.PL: No such file or directory

Возможна и обратная ситуация - вместо Makefile.PL разработчики перешли на Build.PL или что-то еще. Если в архиве с новым исходным кодом есть файл Build.PL, то необходимо

провести в spec-файле замены, обратные приведенным в предыдущем пункте:

- "Makefile.PL INSTALLDIRS=vendor" -> "./Build.PL installdirs=vendor"
- %makeinstall_std -> ./Build install destdir=%{buildroot}
- (если предыдущая замена не сработала) %makeinstall_std -> perl Build.PL install destdir=%{buildroot}
- %make test -> ./Build test
- %make -> ./Build

Если же скрипта Build.PL в новой версии также нет, то необходимо смотреть - а что, собственно, есть - файлы CMake, configure или готовый Makefile и модифицировать spec-файл в соответствии с используемой системой сборки.

fg: no job control

Ошибка означает, что внутри spec-файла используется макрос RPM, не поддерживаемый в POCE. Поддерживается или нет каждый конкретный макрос, можно узнать с помощью команды `rpm --eval` :

```
$ rpm --eval %macro_name
```

Если **rpm** ничего не знает о таком макросе, то результатом выполнения этой команды будет "%macro_name". В противном случае **rpm** развернет определение макроса.

Package check "/usr/bin/rpmlint ..." failed

Для проверки соблюдения политик сборки и оформления spec-файлов, в Росе используется утилита Rpmlint. Многие ошибки этой утилиты некритичны, однако многие имеют "вес" (badness) и если суммарный вес ошибок для пакета больше или равен 50, то сборка завершается с ошибкой. Если это ваш случай - то первым делом необходимо поискать в логе ошибки, для которых светится "Badness: 50" и исправлять их. Перечень всех ошибок можно найти здесь: Rpmlint Errors

?????? ????????????? ??????????

- Ошибки сборки пакетов RPM

Revision #3
Created 2024-07-24 05:01:30 UTC by Антон Сергеевич Абраменко

Updated 2024-07-24 05:20:03 UTC by Антон Сергеевич Абраменко