

# Introduction of RPM

Hua Jie  
feedback@asianux.com



# RPM

**rpm is a powerful Package Manager, which can be used to build, install, query, verify, update, and erase individual software packages. A package consists of an archive of files and meta-data used to install and erase the archive files.**

**---- man page**

# Install

## Steps of installation:

- **Check dependency of packages**
- **Check conflicts of packages**
- **Execute preinstall script procedure**
- **Deal with configuration files**
- **Uncompress packages and store into corresponding position**
- **Execute script procedure after installing**
- **Update RPM database**
- **Execute triggering script procedure**

# Commonly used parameters during installing

- **-i** Install
- **-h** Display installation schedule by #
- **--test** Test installation
- **--replacepkgs** Replace packages
- **--allfiles** Install all files
- **--excludedocs** Do not install explanation documents
- **--includedocs** Install explanation documents
- **--noscripts** Do not execute scripts
- **--nodeps** Do not detect dependency
- **--force** Force to install
- **--relocate** Relocate
- **--excludepath** Do not install files of customized directory
- **--prefix** Establish relocating prefix



# Query

It can:

- Query whether packages have been installed
- Query version of packages
- Query which package a file belongs to
- Query which files are included in package
- Query descriptive information of package
- Customize output

# Commonly used parameters of query

- **-q** Query
- **-p <package filename>** Query package file
- **-a** Query all the packages
- **-f <file list>** Query which package a file belongs to
- **-g <group list>** Query which files are included in group
- **--whatrequires <package name list>** Query which package needs appointed function
- **--whatprovides <package name list>** Query which package provides appointed function
- **--requires/-R** Query package names which a package needs
- **--provides** Query package name which a package provides
- **-i** Display information of package
- **-l** Display the list of files which are contained in package
- **-d** Display the list of document files which are contained in package
- **-c** Display the list of configuration files which are contained in package
- **-s** Display the state of files which are contained in package

## Commonly used parameters of query 2

- **--conflicts** Display conflicted functions of package
- **--scripts** Display the built-in script of package
- **--triggers** Display the built-in triggering script of package
- **--changelog** Display maintaining record of package
- **-qf/queryformat** Customize format of output

# Customize format of output

- **Customize output is a powerful function of RPM, when user wants RPM to output information of package according to its own format, it can use `-qf` (or `--queryformat`) to customize.**

## Customize format of output 2

- **Common text will be output originally**
- **Text which contains ESC will be output after interpreting latter characters according to pre-defined ESC sequence**
- **Function label, RPM has built in many function labels, for example, NAME means name of software, VERSION means version number, RELEASE means release number and so on**
- **Use rpm --querytags command to find all the built-in function labels of RPM**

# Verify

- **Check data integrity of RPM package**
- **Ensure normal running of system**

# Content of verifying

- **Dependency of packages**
- **Attribute of each file**
- **File attributes contains owner, group, rights, MD5 check sum, size, primary device number, secondary device number, symbol link and last modified time.**

# Verifying of file attribute

File type	size	Right	MD5SUM	Primary device number	Secondary device number	Symbol link	Owner	Group	Last modified time
Directory file	-	V	-	-	-	-	V	V	-
Symbol link	-	V	-	-	-	V	V	V	-
FIFO queue	-	V	-	-	-	-	V	V	-
Device file	-	V	-	V	V	-	V	V	-
Common file	V	V	V	-	-	-	V	V	V

# Results of verifying

- If it finds that file has been lost, RPM will output “missing filename”.
- If there has some error, RPM will identify what has changed using the following format:

**SM5DLUGT c filename**

**S: Size of file;**

**M: Mode;**

**5: MD5 check sum;**

**D: Primary and secondary device number;**

**L: Symbol link;**

**U: Owner;**

**G: Group;**

**T: Last modified time.**



## Results of verifying 2

- If certain attribute of file is normal, it will display dot (.); otherwise, it will display its delegated character.
- “c” character in this format will only be output when verified file is configuration file.
- # rpm -V httpd
- S.5....T c /etc/httpd/conf/ httpd.conf

# Verifying commands

- **rpm -V [option1option2 ...] [package ID or pack file]**

Appointed options	
<b>-a (or -all)</b> <b>-f (or --file) file list</b> <b>-g (or --group) class list</b> <b>-p pack file list</b> <b>--noscripts</b> <b>--nodeps</b> <b>--nofiles</b> <b>--nomd5</b>	<b>Verify all installed packages</b> <b>Verify package that contains appointed file</b> <b>Verify package that is appointed class</b> <b>Verify appointed pack file</b> <b>Do not execute verifying script process</b> <b>Do not check dependency</b> <b>Ignore errors of losing file</b> <b>Ignore errors of MD5SUM</b>
General options	
<b>-v</b> <b>-vv</b> <b>--root directory</b> <b>--rcfile file</b> <b>--dbpatch directory</b>	<b>Display accessional information</b> <b>Display debugging information</b> <b>Appoint root directory</b> <b>Appoint resource configuring file of RPM</b> <b>Appoint database library of RPM</b>

# Upgrade

- Upgrade a package from old version to new version
- Appropriately deal with configuration file (CONFIG FILE)

# Treatment of configuration file

- **Through comparing three different kinds of MD5 check sum to decide how to deal with it. The three different kinds of MD5 check sum are:**
  - 1. Original check sum. It is MD5 check sum of configuration file when installing old version package.**
  - 2. Current check sum. It is MD5 check sum of old version configuration file when upgrading.**
  - 3. New check sum. It is MD5 check sum of configuration file in new version package.**

# Treatment of configuration file

- 1. Original check sum=X, Current check sum=X, New check sum=X:  
RPM will use new configuration file to cover old one.
- 2. Original check sum=X, Current check sum=X, New check sum=Y:  
RPM will use new file to cover old one, and old file will not be saved.
- 3. Original check sum=X, Current check sum=Y, New check sum=X:  
RPM will reserve current file.
- 4. Original check sum=X, Current check sum=Y, New check sum=Y:  
RPM will use new file to cover current file.
- 5. Original check sum=X, Current check sum=Y, New check sum=Z:  
RPM will save current file as well as changing another name (suffix .rpmsave to original filename), install new file at the same time and give a warning, for example:

warning: /etc/.funkey saved as /etc/.funkey.rpmsave

- 6. When there is no check sum:  
RPM will save current file as well as changing another name (suffix .rpmorig to original filename, not .rpmsave), install new file at the same time and give a warning, for example:

warning: /etc/.inputdef saved as /etc/.inputdef.rpmorig



# Format of upgrading command

- **rpm -U [option1 option2...] package1 package2**  
It can also use **--upgrade** to replace **-U**

# Upgrading option list

Appointed options	
<b>--oldpackage</b>	Allow upgrading to old version
<b>--hash or -h</b>	Display installing schedule by #
<b>--percent</b>	Display installing schedule by %
<b>--test</b>	Installing test
<b>--replacepks</b>	Replace package
<b>--replacefiles</b>	Replace file
<b>--allfiles</b>	Install all files
<b>--force</b>	Force to execute
<b>--excludedocs</b>	Do not install document files
<b>--includedocs</b>	Install document files
<b>--noscripts</b>	Do not execute script procedure
<b>--nodeps</b>	Do not check dependency
<b>--notrigger</b>	Do not execute triggering procedure
<b>--ignorearch</b>	Ignore architecture
<b>--ignoreos</b>	Ignore operating system
<b>--ignoresize</b>	Do not check size of space
<b>--relocate old directory=new directory</b>	Relocate
<b>-vv</b>	Display debugging information
<b>--root directory</b>	Appoint root directory
<b>--rcfile file</b>	Appoint resource configuration file of RPM
<b>--dbpath directory</b>	Appoint database directory of RPM

# Uninstall

## Steps of uninstall:

- **Check dependency**
- **Execute triggering script procedure before uninstalling**
- **Execute script procedure before uninstalling**
- **Check configuration file**
- **Uninstall**
- **Execute script procedure after uninstalling**
- **Update RPM database**
- **Execute triggering script procedure after uninstalling**

# Format of uninstall command

- **rpm -e [option1option2...] [packageID1 packageID2...]**

# Package ID

- To installed packages, RPM uses the following format to mark it:

**Name[-Subname]-Version-Release;**

**Name:** name of software;

**Subname:** optional, name of sub-package

**Version:** version of software. Note: it can not contain subtraction (-);

**Release:** release number of software.

# Uninstall option list

<b>Appointed options</b>	
<b>--test</b> <b>--nodeps</b> <b>--noscripts</b> <b>--notriggers</b> <b>--allmatches</b> <b>--justdb</b>	<b>Uninstall test</b> <b>Do not check dependency</b> <b>Do not execute script procedure</b> <b>Do not execute triggering procedure</b> <b>Uninstall all the matching packages</b> <b>Just modify database</b>
<b>General options</b>	
<b>-v</b> <b>-vv</b> <b>--root</b> <b>--rcfile</b> <b>--dbpath</b>	<b>Display accessional information</b> <b>Display debugging information</b> <b>Appoint root directory</b> <b>Appoint resource configuration file of RPM</b> <b>Appoint database directory of RPM</b>

# Make RPM package

- **SPEC file**
- **Source code**

# Introduction of SPEC file

- **Commentary line**
- **File head**
- **Function section**

## Introduction of SPEC file ---- Commentary line

- **It starts with #**
- **It is used to include a comment or explanation**
- **It help users to understand the contents**
- **It has no effect on creating of package**
- **It can be located in any position of spec file**

## Brief introduction of SPEC file ---- File head

- It describes basic information of package
- It contains several domains, some are necessary and some are optional
- A domain occupies one line
- Format:  
Field name: Field value  
Note: field name is not case sensitive, and the value can not be null.

## Introduction of SPEC file ---- Function section

- It describes important data and operating injunction of package
- It includes section name and section content
- Section name starts with %, and it occupies one line
- The range of section content starts from the next line of the name of this function section to the pre-line of the name of the next function section or to the end of spec file.
- The position of each function section is unrestricted, it can be put in any position below file head.

# File head ---- necessary domain

- **1. Name:**
  - This field defines name of software
- **2. Version:**
  - This field defines version. Note: it can not contain subtraction in version number
- **3. Release:**
  - This field defines release number. If software has changed only a little, just add release number, do not change version. Note: it can not contain subtraction in release number.
- **4. Summary:**
  - This field defines introduction of package.
- **5. Group:**
  - This field defines class of software.
- **6. License:**
  - This field defines license or copyright rule that is applicable for software. It also can be defined in Copyright.
- License: GPL, BSD, MIT, Public Domain, Distributable, Commercial, Share and so on.

# File head -- optional domain -- basic information

- **1. Vendor:**
- **This field defines vendor of software**
  
- **2. Distribution:**
- **This field defines which release version the software belongs to, this is the classification of package maker itself. Generally, a release version is composed by several packages. If I want to make a release version named “Panda’95”, then the description file of every package must contain the following line:  
Distribution: Panda’95**
  
- **3. Packager:**
- **This field defines packager, that is the person or company who set up this package. The format is:  
Name of packager <e-mail or correlative web page>**
  
- **4. Serials:**
- **This field defines serial number of software, it can also use filed name Epoch. Serial number is integer, it is appointed by packager. It should be increased according to the release version, and must be unique. Serial number can be used to describe the dependency of packages. The following example appoints serial number to be 4:  
Serial: 4 or Epoch: 4**
  
- **5. URL:**
- **This field defines the web address of correlative information about packed software. For example:  
URL: <http://www.asianux.com>**



## File head -- optional domain -- dependency

- **Dependency is used by RPM to describe relationship between packages. The dependent package is called function by RPM. It can be existed package, and also can be virtual package. Virtual package has no version number.**

# File head -- optional domain – dependency 2

- **1. Provides:**
- This field defines functions that package provides. The format is as following:  
Provides: function 1 [, function 2]...
- Generally, the function that package provides is share library which exist as virtual package. When several packages provide the same function, it usually uses virtual package to express its service.
- **2. Requires:**
- This field defines the needed function. The format is as following:  
Requires: function 1 [comparison operators1 [serial number 1:] version number 1 [- release number 1]] [, function 2 [comparison operator 2 [serial number 2:] version number 2 [- release number 2]]]...
- \* [] means it is optional;
- \* Comparison operator can be <, >, =, <= or >=;
- \* When serial number is not selected, RPM is 0 by default;
- \* Comma between functions is optional, it can use space to separate.
- **3. Conflicts:**
- This field defines which function can not coexist with this package. Its format is as following:  
Conflicts: function 1 [comparison operators1 [serial number 1:] version number 1 [- release number 1]] [, function 2 [comparison operator 2 [serial number 2:] version number 2 [- release number 2]]]...



## File head -- optional domain – dependency 3

- **Automatic implement of dependency**
- **/usr/lib/rpm/find-requires**  
**Used to find share library that package requires, these libraries will be added into ‘Requires’ of package by using virtual form.**
- **/usr/lib/rpm/find-provides**  
**Used to find share library that package provides, these libraries will be added into ‘Provides’ of package by using virtual form.**

# File head -- optional domain – dependency 4

- **4. Autoreq:**
- **This field is used to dictate RPM whether to find required share library automatically. RPM does not execute find-requires procedure only when the domain value is no or 0.**
  
- **5. Autoprov:**
- **This field is used to dictate RPM whether to find provided share library automatically. RPM does not execute find-provides procedure only when the domain value is no or 0.**
  
- **6. Autoreqprov:**
- **This field is used to dictate RPM whether to find provided and required share library automatically. RPM does not execute find-provides procedure and find-requires procedure only when the domain value is no or 0. This field sets field value of Autoreq and Autoprov to be appointed value at the same time.**
  
- **In spec file, the result will change according to the order of the above three domains. Usually, set the last one as standard.**

## File head -- optional domain – system correlated

- **When RPM making package, it can appoint applicable CPU architecture or OS, and it can also appoint inapplicable CPU architecture or OS, in this way, when RPM found that current CPU architecture or OS are not incompatible with packages, it will cease making packages. Current default CPU architecture of RPM is defined by macro `%_arch`, usually it is `i386`. The default current OS of RPM is defined by macro `%_os`, it usually is `linux`.**

## File head -- optional domain – system correlated 2

- **1. Excludearch:**
- **This field defines architectures that are inapplicable for packages.**
- **Its format is:**
- **Excludearch: architecture 1 [architecture 2]...**
  
- **If current architecture is among this field value, RPM will exit when making packages.**
  
- **2. Exclusivearch:**
- **This field defines architectures that are applicable for package. Its format is similar with Excludearch:**
- **Exclusivearch: architecture 1 [architecture 2]...**

## File head -- optional domain – system correlated 3

- **3. Excludeos:**
- **This field defines OS that is inapplicable for packages.**
- **Its format is:**
- **Excludeos: OS 1 [OS 2]...**
  
- **4. Exclusiveos:**
- **This field defines OS that is applicable for packages.**
- **Its format is:**
- **Exclusiveos: OS 1 [OS 2]...**

## File head -- optional domain – directory correlated

- **1. Prefix:**
- **This field defines directory prefix that is re-locatable, it can be written in spec file for several time. Its format is:**
- **Prefix: directory prefix 1 [directory prefix 2]...**
  
- **2. Buildroot:**
- **This field defines the mutual root directory of files which are contained by package. This root directory can only be used when RPM making packages. That is to say, when RPM making packages, it will set this directory to be root (assign chroot function), pick up needed files, create packages.**

# File head -- optional domain – source code correlated

- **1. Source:**
- **This field defines the source code file which is to be contained when packing RPM.**
- **Source [number]: source code file**
  
- **Note: The value of this field can adopt URL form. This way can provide other users with location information of this source code. When RPM making source package, it picks up the last filename, the content before URL will be ignored by RPM.**
  
- **2. NoSource:**
- **In the above example, if you don't want to include files which are defined by Source1 and Source2 when packing, what should you do?**
- **Method 1: delete the whole line;**
- **Method 2: make this line to be commentary (add # at the head of this line);**
- **Method 3: appoint Nosource field, this field can be repeated. The format is:**
- **NoSource: number of source code field**



## File head -- optional domain – source code correlated 2

- **3. Patch:**
- **This domain defines the patches that is needed by RPM when making source packages. To the name of this file, it is suggested to use format: “software name –version.function .patch”**
- **Describing format of this domain:**
- **Patch [number]: source code patch file**
- **Note: the value of this domain also can adopt URL form as Source domain, RPM only pick up its software name.**
- **4. NoPatch:**
- **The function of this domain is similar with NoSource. To the patch file which is corresponding to the number it defined, RPM will not pack it. This domain can be repeated in description file.**

# Function section ---- necessary section

- 1. %description
- This section is description section, its content is the detailed introduction of packages. The text form of introduction is unrestricted.
- The description format of this section name is:
- %description [sub-package option]
- Format of sub-package is: [-n] sub-package name
- Three forms of section name of description section:
- (1) when the format of section name is “%description”,
- The content of the function section is about father package, father package can also be called primary package, it uses software name to name it, its format is: software name-version-release number.architecture.rpm.
- (2) when the format of section name is “%description sub-package name”,
- The content of the function section is about sub-package, when there is no “-n” in sub-package option, sub-package is name by form of software name sub-package name, its format is: software name – sub-package name – version – release number.architecture.rpm.
- (3) when the format of section name is “%description –n sub-package name”,
- The content of the function section is also about sub-package, when there is “-n” in sub-package option, sub-package is named by the form of package name directly. It does not contain software name, the format is: sub-package name – version – release number.architecture.rpm.

# Function section ---- necessary section 2

- **2. %files:**
- **This section is file section, it defines the which file the package needs to contain. This section is usually at the end of spec file in order to add filename and easy to be edited.**
- **The format of this section name is:**
- **%files [sub-package option] [-f filename]**
  
- **When there isn't any option, the content of this section defines the file list that father package wants to pack;**
- **When there is option, the content of this section defines the file list that sub-package wants to pack;**
- **When it chooses -f option, RPM will read the list of files to be packed in appointed file as well as read list of packed file in file section. In appointed file, one file occupies on line.**
  
- **The content format of file section is:**
- **[modifier 1 [modifier 2]...] filename**
- **Among this, modifier is option, one file can contain several modifiers, filename must begin with / (absolute path form).**

# Function section ---- necessary section 3

- **(1) File correlated**
- **\* %doc:**
- **This modifier defines file type to be document file;**
- **\* %config:**
- **This modifier defines file type to be configuration file;**
- **\* %config(missingok):**
- **This modifier defines file type to be configuration file, and this file can be lost. Even it is lost, RPM does not consider it is an error when uninstalling packages.**
- **\* %config(noreplace):**
- **This modifier defines file type to be configuration file, and if there has a file with the same name in system when installing, then the file in this package will be installed by changing another name, and it will add a postfix to its filename: .rpmnew. (If it does not use this modifier, RPM will replace the name of file in system when finding file with the same name, and add postfix .rpmorig, the file in package will still use the original name.) When uninstalling package, the file with the same name in system will be saved by RPM by changing to another name, and add postfix .rpmsave.**

# Function section ---- necessary section 4

- **\* %ghost:**
- **The content of files which are bedecked by this modifier can not be included in package. Generally, this kind of file is log file, its attributes (filename, owner, group and so on) are very important, but its content is not important. After using this modifier, RPM will add its file attribute to package.**
- **\* %attr:**
- **This modifier sets attribute information of file. Its format is:**
- **%attr(popedom, owner, group)**
- **Note: it usually use number to express popedom (octal), owner and group can be numbers or character string. If you want to use default value of popedom, owner and group, then you can use subtraction sign to express it.**
- **\* %verify:**
- **This modifier sets attributes which need to be checked out. These attributes include owner, group, mode (popedom), md5 (MD5 check sum), size, maj (major device number), min (sub-device number), symlink (symbol link), mtime (last modified time).**
- **Format of using this modifier:**
- **%verify([not] owner group mode md5 size maj min symlink mtime)**
- **Note: 'not' is optional. When it chooses 'not', this means that it needs to check out attributes except the selected attributes.**

# Function section ---- necessary section 5

- (2) Directory correlated
- \* %docdir:
  - This modifier defines prefix of explanation document, in this way, the type of all the following files which use appointed filename as prefix will be set to be explanation document when packed by RPM.
  - Through this modifier, users can conveniently set files like document files, because they usually under certain fixed directory, and have common prefix.
- \* %dir:
  - When RPM making packages, if the file to be packed is directory, then RPM will include all the files under this directory in package. (Note: when the file to be packed is symbol link and this symbol link point to a directory, RPM will not consider it as a directory, but deal with it as a common file.) If you only want to include this directory name in package, maker only needs to use this modifier on this directory name.

# Function section ---- necessary section 6

- **(3) Modifier to set all the files**
- **\* %defattr:**
- **It is usually output to the first line of file content.**
- **The format to use it is:**
- **%defattr(popedom, owner, group)**
  
- **Thereinto: Popedom, owner and group all can use subtraction (-). The attributes which use subtraction will be decided by system.**

# Function section ---- optional section

- Its content is script procedure
- Description format is:  
Section name [sub-package option]
- Sections which are used to build package
- Sections which are used to manage
- Alternating section
- Other sections

## Function section -- optional section -- build package

- **When RPM builds a package through source procedure, it needs to execute four operations ---- pretreatment, compile, install and clean. They respectively correspond to %prep, % build, %install and %clean.**

## Function section -- optional section -- build package 2

- **%prep:**
- **This is pretreatment section, its content is pretreatment script procedure. This procedure implement the following task:**
- **\* Establish directory which is used to compile software;**
- **\* Uncompress source procedure;**
- **\* Upgrade source procedure through patches;**
- **\* Execute some other operation, that can make it possible to compile source procedure at any time.**

## Function section -- optional section -- build package 3

- **1.1 %setup**
- **This macro uses commands gzip and tar in system to uncompress source package. Its format is:**
- **%setup [-n name] [-c] [-D] [-T] [-b N] [-a N]**

## Function section -- optional section -- build package 4

- (1) When there isn't any option,
- This macro is used to uncompress default source package (appointed by `Source` or `Source0` field of head file). Note: the files in source package should use “software name-version number” as their upper directory, in this way, macro `%setup` can work normally. If it does not use “software name-version number” as their upper directory, then, when macro `%setup` works, a command “`cd software name-version number`” will slip a cog and exit because there is no such directory in system (unless adding command of establishing this directory on macro).

## Function section -- optional section -- build package 5

- **(2) -n name:**
- **If files in source package don't adopt "software name-version number" as their upper directory, macro %setup can not work normally, then it can use -n option to quote this directory.**
  
- **(3) -c:**
- **The function of this option is to establish upper directory ("software name-version number") and switch to this directory.**
- **It's suitable for the following situation: some source package is packed under the directory where the source procedure locates, so the files in it don't have upper directory. In this case, if you want to uncompress correctly, it must establish upper directory.**

## Function section -- optional section -- build package 6

- **(4) -D:**
- **Its function is that, do not delete the upper directory (software name-version number) of software before uncompressing source package.**
  
- **(5) -T:**
- **The function of this option is that, do not uncompress default source package (defined by Source or Source0 domain of head file).**

## Function section -- optional section -- build package 7

- (6) **-b N:**
- This option indicates RPM to uncompress the nth source package before switching to upper directory (defined by SourceN domain of head file). This is suitable for source package which contains upper directory. Note: if it does not use **-T** option at the same time when using this option, then the two source packages that RPM uncompress, one is default package (defined by Source or Source0 domain), another one is package which is appointed by **-b** option (defined by SourceN domain).
- (7) **-a N:**
- This option indicates RPM to uncompress the nth source package after switching to upper directory (defined by SourceN domain of head file). This is not suitable for source package which contains upper directory. Generally, in order to establish upper directory and switch to it, add **-c** option when using this option. Note: if it does not use **-T** option at the same time when using this option, then the two source packages that RPM uncompress, one is default package (defined by Source or Source0 domain), another one is package which is appointed by **-a** option (defined by SourceN domain). In this way, when N equals 0, source package will be uncompressed twice. So, if you only want to uncompress appointed package, please use **-T** option to avoid uncompressing default source package.

## Function section -- optional section -- build package 8

- **1.2 %patch**
- **This macro uses command patch in system to add patches to appointed source package, so that to upgrade program. Its format is:**
- **%patch [-P N] [-p N] [-b name] [-E]**

## Function section -- optional section -- build package 9

- (1) When there isn't any option:
- This macro uses default patch file (the 0th patch file), it is file defined by Patch or Patch0 domain of head file.
  
- (2) -P N:
- Use this option to indict RPM to use the Nth patch file (which is defined by PatchN domain of head file).
  
- (3) -p N:
- This option and its parameters are transferred to command patch by macro %patch directly.

## Function section -- optional section -- build package 10

- (4) **-b name:**
- **When several patch commands operate on same file, patch will save the original file by using another name (change its suffix to .orig). If you want to use other suffix, use -b to set, then original file will change name to “original filename + suffix”.**
- **When executing this option, it in fact transfer a option and parameter to command patch, that is --suffix name.**
  
- (5) **-E:**
- **This option is transferred to command patch directly, its function is: if the content of file is null (byte number is 0) after adding patch, then delete this file.**

## Function section -- optional section -- build package 11

- **2 %build:**
- **This is compile section, its content is compile script procedure. This procedure implements compiling and link of source program. The simplest example is that there only has one make command in program. This is suitable for most situations, because most software have their own makefile, it can implement compiling and link through make command. If there is no makefile, it needs maker of packages to write a serials of compile and link commands on compile section.**

## Function section -- optional section -- build package 12

- **(3) %install**
- **This is install section, its content is installation script procedure. This procedure saves executable program which has been compiled and linked or other files to appointed directory. These program or files are used when RPM makes packages. A simplest example is that program uses only one make install command to implement installation. If there isn't, package maker has to write dictation itself.**

## Function section -- optional section -- build package 13

- **1.4 %clean:**
- **This is clean section, its content is clean script procedure. This procedure executes after RPM has made package. It usually deletes temporary files or directories when compiling and linking, accomplishes ending work.**

## Function section -- optional section -- management

- **This section is used to manage package itself (install, uninstall and check out), include five function sections: %pre, %post, %preun, %postun and %verifyscript.**

## Function section -- optional section – management 2

- 1. % pre:
- The content of this section is pre-installation script program. It executes before install packages. Generally, it inspects OS environment, establishes corresponding directory, clean up excrescent files and so on, and prepares for the installation of packages. This section is seldom used.
- The format of this section name is:  
`%pre [sub-package option]`

## Function section -- optional section – management 3

- **2 %post:**
- **The content of this section is script procedure after installation. It execute after finishing to install package, it is usually used to set up symbol link, modify system configuration file, run ldconfig program and so on, so that can make software works normally.**
- **Its format is: %post [sub-package option]**

## Function section -- optional section – management 4

- **3 %preun**
- **The content of this section is script procedure before uninstalling. It executes before uninstall, prepares for uninstall. For example, when certain program in the package which is to be uninstalled is running, this script procedure must kill it, otherwise it can not be uninstalled normally.**
- **Its format is: %preun [sub-package option]**

## Function section -- optional section – management 5

- **4 %postun:**
- **The content of this section is script procedure after uninstalling. It executes after uninstall, finish the ending work of uninstall. For example, change system configuration file to the original one, rerun command ldconfig, delete uninstalled share library from cache file ld.so.cache.**
- **Its format is: %postun [sub-package option]**

## Function section -- optional section – management 6

- **5 %verifyscript:**
- **The content of this section is verify script procedure. When RPM verify packages, except executing standard verify, if package maker has set this verify script procedure, it will execute.**
- **Its format is: %verifyscript [sub-package option]**

## Function section -- optional section -- alternate

- This kind of section includes:  
`%triggerin`, `%triggerun`, `%triggerpostun`, their contents are all script procedure that are used by RPM on alternate control among packages. These script procedure are triggered when system fill appointed conditions.

## Function section -- optional section – alternate 2

- (1) **%triggerin**: it contains script procedure when installing. To the package it locates and the appointed package, when only one of them is installed, installing another one will trigger this procedure to execute.
- (2) **%triggerun**: it contains script procedure when installing. To the package it locates and the appointed package, when both of them are installed, uninstalling anyone of them will trigger this procedure to execute.
- (3) **%triggerpostun**: it contains script procedure after installing. It can only be triggered when the appointed package is uninstalled.
- Their format is:
- **Alternate section name [sub-package option] [-p interpreter] – triggering condition 1 [, triggering condition 2] ...**

## Function section -- optional section – alternate 3

- **-p option:**
- **This option is used to appoint an interpreter to interpret the procedure which execute alternate section. By default, RPM uses /bin/sh to execute script (this kind of script is written by using SHELL, it can also be call shell program). But some of the RPM packages use /usr/bin/perl to execute procedure (this kind of script is written by using perl), this needs to use -p option to appoint interpreter to be /usr/bin/perl**

## Function section -- optional section – alternate 4

- **Triggering condition:**
- **The format of triggering condition is:**
- **Function name [comparison operator version number]**
  
- **Among it: comparison operator and version number is optional, when there is only one function name, it means to trigger procedure when this function exists. Comparison operator can use >, =, <, >= and <=.**
- **Such as triggering condition: bash and fileutils>3.0, these are all legitimate.**
- **Alternate function must at least have one triggering condition. When there have several triggering conditions, use comma to separate them, RPM will execute trigger script if one or more than one conditions fulfill.**

## Function section -- optional section – alternate 5

- **Why it must use alternate section? The following example will well explain this.**
- **We assume that package mymailer needs symbol link file `/etc/mymailer/mailer` to appoint currently used mail sending proxy program. If sendmail package has been installed, then this symbol link program will appoint `/usr/bin/vmail` program. If both of two packages have been installed (actually, sendmail and vmail conflict with each other), so we need not to consider about which file the symbol link file appoints to. Of course, if neither has been installed, then symbol link file `/etc/mymailer/mailer` has no reason to exist.**

## Function section -- optional section – alternate 6

- To the above request, it implements by compile triggering script procedure for mymailer package. These scripts will change content of /etc/mymailer/mailer when the following event happens:
  - 1) sendmail has already been installed;
  - 2) vmail has already been installed;
  - 3) sendmail is being uninstalled;
  - 4) vmail is being uninstalled.
- The script of the former two events can be written like this:
  - %triggerin -- sendmail
  - In -sf /usr/bin/sendmail /etc/mymailer/mailer
  - %triggerin -- vmail
  - In -sf /usr/bin/vmail /etc/mymailer/mailer
- These are two scripts that is triggered by sendmail or vmail when installing. They will be executed in the following situations:
  - 1) Install or upgrade sendmail package when mymailer package has already been installed;
  - 2) Install or upgrade vmail package when mymailer package has already been installed;
  - 3) Install or upgrade mymailer package when sendmail package has already been installed;
  - 4) Install or upgrade mymailer package when vmail package has already been installed.

## Function section -- optional section – alternate 7

- The script of the latter two events can be written like this:
- `%triggerun -- sendmail`
- `if [ -f /usr/bin/vmail ]`
- `then`
- `In -sf /usr/bin/vmail /etc/mymailer/mailer`
- `else`
- `rm -f /etc/mymailer/mailer`
- `fi`
- `%triggerun -- vmail`
- `if [ -f /usr/bin/sendmail ]`
- `then`
- `In -sf /usr/bin/sendmail /etc/mymailer/mailer`
- `else`
- `rm -f /etc/mymailer/mailer`
- `fi`
  
- These two scripts will be executed in the following situations:
- 1) Uninstall mymailer package when sendmail package has already been installed;
- 2) Uninstall mymailer package when vmail package has already been installed;
- 3) Uninstall sendmail package when mymailer package has already been installed;
- 4) Uninstall vmail package when mymailer package has already been installed.

## Function section -- optional section – alternate 8

- In order to make sure that symbol link file `/etc/mymailer/mailer` is deleted after uninstalling mymailer package. It can add command of deleting this file in `%postun` function section of mymailer package's description file:
- `%postun`
- `rm -f /etc/mymailer/mailer`
  
- Note: the content of `%postun` section is script procedure after uninstalling, it will execute after uninstalling mymailer package.
- As it describes above, when one package has close relationship with another package, we can implement the management of some files though alternating function. This not only extend the management function of RPM packages, but also helpful for the normal running of packages.

## Function section -- optional section – others

- **%changelog: the content of this section is software maintenance record, it records time, person, its RMAIL and maintained item of every maintenance of software.**
- **The format of %changelog is:**
- **\* week month day year person EMAIL**
- **- maintained content 1**
- **- maintained content 2**
- **- ...(other contents)**
- **Note: every maintenance record starts with “\*”, week and month must use abbreviation. Content can be separate to several lines, and its better to start with “–” in every line.**