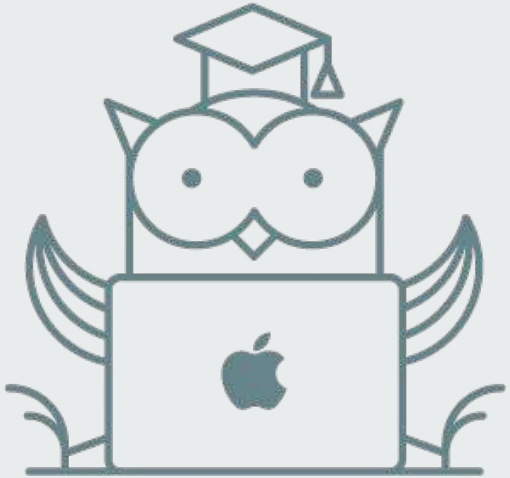
Курс «Администратор Linux»

Управление пакетами

Занятие # 7

Дмитрий Молчанов
Григорий Ожегов



Есть 3 стула

make && make install

Дисклеймер:

Авторы абсолютно против использования этого метода в повседневной жизни. Бездумное использование «make install» опасно для операционной системы и вашей счастливой профессиональной жизни. Дальнейшее повествование имеет отрицательный эмоциональный окрас.

make && make install

Самый простой метод, который описывается командой:

```
./configure && make && make install
```

Суть метода состоит просто в сборке и инсталляции необходимого ПО куда-то в операционную систему.

make && make install

- 👍 Возможность использовать последнюю или, наоборот, устаревшую версию ПО, которой нет в репозиториях (но надо не лениться и написать spec'у)
- 👍 Возможность собрать ПО с теми модулями/опциями, которые необходимы именно в вашем случае и для ваших целей (но надо не лениться и написать spec'у)
- 👍 Отсутствие необходимости разбираться с созданием пакета и делать свои сложные «обертки» для сборки (но надо не лениться и написать spec'у, которая «make install» выполнит внутри себя)

- 👎 Требует наличия сборочного окружения на сервере
  - 👎 В случае компрометации сервера - облегчает задачу по сборке эксплоита злоумышленнику
  - 👎 это окружение не является необходимым для выполнения приложения, оно нужно только для подготовки окружения.
- 👎 Занимает много времени.
- 👎 Ручное разрешение зависимостей.
- 👎 Сложность тиражирования.
- 👎 Возможны затруднения с unit'ами и init-скриптами
- 👎 Больше затрат на поддержание системы в целом за счет необходимости самостоятельно следить за обновлениями безопасности

На macOS также не стоит использовать «make && make install» для установки ПО при наличии пакетного менеджера `brew` 🍺. С помощью «`brew edit`» можно отредактировать формулу, например добавить флаги сборки, и переустановить пакет.

make && make install

Самая ключевая проблема этого способа установки ПО заключается в стремлении к хаосу.
Подробнейший разбор и холивар есть в очень хорошей статье: https://otus.pw/ulZn/

В ОС устанавливается уже собранное ПО, поддерживаемое командой поддержки репозитория/дистрибутива.

RPM-пакет можно рассматривать как умный gzip-архив

Важно понимать, что при установке не происходит и не должно происходить сборки/компиляции ПО

RPM

- 👍 Экстремально низкие трудозатраты на установку ПО
- 👍 ПО поддерживается maintainer’ами репозитория/дистрибутива
- 👍 Зависимости и их автоматическое разрешение
- 👍 Большинство пакетов разбито на binary/dev-подпакеты, нет мусора в ОС
- 👍 Для установки ПО не нужно окружение для сборки, меньше софта в ОС
- 👍 Возможность автоматически выполнять скрипты при установке/удалении

- 👎 Замороженные версии софта, не всегда последние

Пакеты представляют из себя архив определенного формата, в нем содержится:

- Метаинформация:
  - Имя
  - Версия
  - Релиз
  - Архитектура
  - Зависимости
  - Ресурсы
  - ...
- Файлы
- Скриптлеты
  - pre-install
  - post-install
  - pre-remove
  - post-remove
  - ...

RPM: управление

- `rpm -q {name}` — проверить установлен ли пакет {name}
- `rpm -qi[p] {name}/{pkg}` — показать мета-информацию о пакете
- `rpm -qp --queryformat %{VERSION}-%{RELEASE} {pkg}` — формат вывода информации
- `rpm -e {name}` — удалить пакет
- `rpm -i {pkg}` — установить пакет
- `rpm -ql[p] {name}/{pkg}` — вывести список файлов пакета
- `rpm -q[p] --scripts` — показать скриптлеты
- `rpm -q[p]R` — показать от каких пакетов зависит этот
- `rpm -q[p] --provides {name}` — показать ресурсы предоставляемые пакетом
- `rpm -qf {file}` — показать какому пакету принадлежит {file}.
- `rpm2cpio {pkg} | cpio -idmv` — распаковать содержимое пакета

RPM: сборка

```
[root@ozhegov rpmbuild]# tree -d -L 1 ~/rpmbuild
/root/rpmbuild
├── BUILD # директория в которой происходит сборка
├── BUILDROOT # директория-корень файловой системы, на которую накатывается пакет
├── RPMS # директория с собранными пакетами
├── SOURCES # директория с исходными файлами
├── SPECS # директория с spec-файлами
└── SRPMS # директория с SRPM-пакетами
```

При установке SRPM-пакета (`rpm -i`) создается такая структура, которая позволяет сразу начать собирать

RPM: spec-файл

[root@v ozhegov]# vim otus.spec # vim при создании нового файла с расширением .spec покажет шаблон

```
Name: otus
Version: 2017.11.0
Release: 948%{?dist}
Summary:otus
Group: Apps/sys
License:
URL: https://otus.ru
Source0: otus # то лежит в папке SOURCES. Внутри spec-файла можно получить путь через макрос %{SOURCE0}
Autoreq: 0 # не входит в шаблон, но очень важный флаг. Отключает автоматическое определение зависимостей пакета
BuildRequires: python36u, python36u-devel, rpm-build, redhat-rpm-config, mysql-devel # зависимости при сборке, без этих пакетов будет ошибка
Requires: MySQL-python, python36u, wkhtmltopdf # зависимости установки. Yum попробует их скачать и поставить; rpm проверит, без них выдаст ошибку

%description # Описание пакета

%prep
%setup -q # Подготовка к сборке

%build # сборка: конфигурация, компиляция, результаты сборки сохраняются в BUILD
%configure
make %{?_smp_mflags}

%install # установка. Все, что собрано на этапе %build копируется в BUILDROOT, который представляет из себя корень файловой системы ОС, на которую
будет устанавливаться пакет. То, что будет лежать в %{buildroot}/opt/otus при установке окажется в /opt/otus
make install DESTDIR=%{buildroot}

%files # файлы, одна из ключевых директив. Необходимо перечислить все файлы, которые входят в пакет
%defattr(755,root,root)  # можно управлять пользователем, которому будет принадлежать файл и правами
/opt/otus/
%config(noreplace) /etc/otus # файлы, которые находятся под %config(noreplace) не будут перетираться RPM'кой (например конфиги)
```

RPM: spec-файл (explained)

- http://ftp.rpm.org/max-rpm

RPM: сборка

`rpmbuild -bb otus.spec` – сборка RPM

`rpmbuild -ba otus.spec` – сборка RPM+SRPM

yum

Yum является программой управления и автоматизации работы с пакетным менеджером rpm (Redhat Packet Manager).

Сам по себе rpm способен только проинсталлировать пакет в систему, проверив установлены ли пакеты-зависимости.

Yum позволяет реализовать скачивание пакета, автоматическое разрешение зависимостей , обновления и т.д.

yum: приготовление

/etc/yum.conf — конфиг в ini-формате

[root@otus ~]# cat /etc/yum.conf

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=5
bugtracker_url=http://bugs.centos.org/set_project.php?project_id=23&ref=http://bugs.centos.org/bug_report_page.php?category=yum
distroverpkg=centos-release
```

/etc/yum.repos.d/*.conf — файлы в ini-формате описывающие репозитории

[root@ns1 ~]# cat /etc/yum.conf.d/go.repo

```
[c7-go-repo]
name=go-repo - CentOS
baseurl=https://mirror.go-repo.io/centos/7/$basearch/
gpgcheck=0
enabled=1
```

yum: управление

- `search` — поиск пакета
- `install` — установка пакета(ов)
- `update` — обновление (до версии)
- `downgrade` — откат до до версии
- `check-update` — проверка обновлений
- `remove` — удаление пакета
- `info` — информация о пакете
- `whatprovides` — найти из какого пакета файл
- `shell` — CLI

Сделаем еще шажок:

В ОС устанавливается уже собранное ПО, поддерживаемое командой поддержки репозитория/дистрибутива. RPM-пакет можно рассматривать как умный gzip-архив

В ОС устанавливается уже небольшая ОС. Docker-образ можно рассматривать как виртуальную машину

docker: приготовление, Dockerfile

```
FROM centos:7
MAINTAINER Grigory Ozhegov <ozhegov@ktsstudio.ru>

RUN yum clean all
RUN yum update -y --verbose --noplugins
RUN yum groupinstall -y --noplugins "Development tools"

RUN yum install -y --noplugins git openssh openssh-clients python36u python36u-libs python-pip nodejs-6.12.0 yarn
which libpng libpng12

RUN pip install --upgrade pip
RUN pip install virtualenv

COPY . /opt/otus
RUN virtualenv -p python3.6 /opt/otus/env && source /opt/otus/env/bin/activate && pip install -r requirements.txt
RUN yarn build:prod

ENTRYPOINT /opt/otus/env/bin/python /opt/otus/env/bin/gunicorn -c /etc/otus/gunicorn.conf otus.wsgi:application
```

docker: приготовление, сборка

Таким образом по аналогии с BUILDROOT формируется слепок ФС, который разворачивается в виде виртуальной машины:

```
docker build -t registry.otus.ru/otus/otus -f Dockerfile  .
docker push registry.otus.ru/otus/otus
docker run registry.otus.ru/otus/otus:latest
```

Спасибо за внимание

Дмитрий Молчанов
Григорий Ожегов