目录

1. 基于源码编译 deb 包

安装基础构建工具 build-essential 和 devscripts

```
sudo apt install build-essential devscripts
```

1.1. 基于 debian 仓库

http://ftp.debian.org/debian/pool/main/f/fonts-wqy-microhei/ 为例

使用 dget 下载源码以及 dsc 文件

```
dget -ux http://deb.debian.org/debian/pool/main/f/fonts-wqy-microhei/fonts-wqy-microhei_0.2.0-beta-3.dsc
cd fonts-wqy-microhei-0.2.0-beta/
```

使用 dpkg-buildpackage 在本机进行编译

```
dpkg-buildpackage
```

缺少 xdelta 依赖，在本机安装，安装后重新编译

```
sudo apt install xdelta
dpkg-buildpackage
```

1.2. 基于 Ubuntu 仓库

https://packages.ubuntu.com/impish/elementary-xfce-icon-theme 为例

使用 dget 下载源码以及 dsc 文件

```
dget -ux http://archive.ubuntu.com/ubuntu/pool/universe/e/elementary-xfce/elementary-xfce_0.15.2-1.dsc
```

使用 dpkg-buildpackage 在本机进行编译

```
dpkg-buildpackage
```

缺少 icon-naming-utils 依赖，在本机安装，安装后重新编译

```
sudo apt install icon-naming-utils
dpkg-buildpackage
```

1.3. 基于 GitHub 源码

https://github.com/tmux/tmux 为例

使用 git 下载源码

```
git clone https://github.com/tmux/tmux.git
```

下载完成后查看 release 版本

```
git tag
```

将源码目录修改成“包名-版本号”的格式，包名需要小写；然后生成 debian 目录

```
dh_make --createorig
```

可修改 debian 目录下的 rules 文件以及 control 文件，然后进行包构建

```
dpkg-buildpackage
```

缺少依赖时查找库

```
apt search *
```

安装生成的 deb 包

```
sudo dpkg -i *.deb
```

1.4. 源码拆包

http://ftp.debian.org/debian/pool/main/e/elementary-xfce/ 为例

使用 dget 下载源码以及 dsc 文件

```
dget -ux http://archive.ubuntu.com/ubuntu/pool/universe/e/elementary-xfce/elementary-xfce_0.15.2-1.dsc
```

根据 debian/control 文件，在 debian 路径下新建对应的.install 文件

自动在包名生成架构和版本号
debian/control 文件内，将 package 的 Architecture 改成 any；Depends 加上,${shlibs:Depends};

1.5. quilt 工具添加补丁

安装 quilt 工具

```
sudo apt install quilt
```

创建一个新 patch

```
quilt new change-buile-opts.patch
```

把 patch 文件加入到 CMakeLists.txt

```
quilt add CMakeLists.txt
```

编辑 CMakeList.txt 文件

```
vim CMakeLists.txt
```

更新 patch

```
quilt refresh
```

1.6. pbuilder 构建工具的安装与使用

安装 pbuilder 构建工具

```
sudo apt install pbuilder
```

在 home 目录下添加.pbuilderrc 文件

```
vim .pbuilderrc
```

内容如下

```
MIRRORSITE=http://10.15.0.120:8081/repository/debian-proxy/
USENETWORK=yes
DISTRIBUTION=testing
ALLOWUNTRUCTED=yes
```

```
DEBOOTSTRAPOPTS=('--variant=buildd' '--no-check-gpg')
```

使用 pbuild create 创建纯净的编译构建环境，可以通过参数指定所要模拟的 debian 环境版本，最终会打包为 base.tgz。

```
sudo pbuilder create
```

使用 dget 下载 dsc 文件以及源码

```
dget http://deb.debian.org/debian/pool/main/d/debootstrap/debootstrap_1.0.114.dsc
```

用 pbuilder build 编译目标源码包，参数为 src 包的 dsc 文件

```
sudo pbuilder build debootstrap_1.0.114.dsc
```

编译生成的 deb 包位于/var/cache/pbuilder/result/目录下

```
ls /var/cache/pbuilder/result/
```

源码内不含有 debian 目录时，需要先生成 debian 目录

```
dh_make --createorig
```

当修改源码内容时，需要重新生成 dsc 文件

```
dpkg_source -b .
```

2. 构建软件包仓库

2.1. 构建本地仓库

安装 reprepro

```
sudo apt install reprepro
```

创建 repository 目录，并在目录内创建 conf 目录,在 conf 路径下创建 distributions 文件，内容如下

```
Origin: cetc15_stable
Suite: stable
Codename: buster                    # 这是 debian10 的代号，可以随便叫，记住就行，之后经常用到
Version: 1.0
Architectures: arm64 source         # 拉取构架和源码（不要源码去掉 source 就行）
Components: main contrib non-free
UDebComponents: main contrib non-free
Description: this is a test repo    # 描述信息
```

将 deb 包导入 reprepro 仓库，在 repo 目录下执行

```
reprepro includedeb buster *.deb
```

2.2. 安装 nginx 并对 reprepro 仓库映射

安装 nginx

```
sudo apt install nginx-full
```

设置 location 映射

```
vim /etc/nginx/sites-enabled/default
```

修改内容如下

```
location /repo{
    alias /home/user/repo;
    autoindex on;
}
```

修改 nginx 访问用户,第一行将 www-data 改为 user 或者 root

```
vim /etc/nginx/nginx.conf
```

重新读取 nginx

```
sudo nginx -s reload
```

2.3. pbuilder 构建工具与自建仓库的连接

自定义一个.pbuider 目录

```
mkdir ~/.pbuider/hooks
```

在 hooks 目录下新建一个 D05localrepo 文件，内容如下

```
#!/bin/bash

echo "deb [trusted=yes] http://localhost/repo buster main" > /etc/apt/sources.list.d/cetc15.list

cat > /etc/apt/preferences.d/90cetc15<<EOF
Package: *
Pin: release o=cetc15_stable
Pin-Priority: 900
EOF
apt-get -y update
```

注：

- repo 地址后的 buster 与 reprepro 仓库的~/repo/conf/distributions 内的 codename 一致
- o=cetc15_stable 与 reprepro 仓库的~/repo/conf/distributions 内的 origin 一致

在~/.pbuilderrc 文件内添加 hook 的路径

```
vim ~/.pbuilderrc
```

内容如下

```
MIRRORSITE=http://10.15.0.120:8081/repository/debian-proxy/
USENETWORK=yes
DISTRIBUTION=testing
ALLOWUNTRUCTED=yes
DEBOOTSTRAPOPTS=('--variant=buildd' '--no-check-gpg')
HOOKDIR="$HOME/.pbuilder/hooks/"
```

3. 构建基础镜像

3.1. debootstrap 工具的安装与使用

安装 debootstrap 工具

```
sudo apt install debootstrap
```

使用 debootstrap 构建基础镜像

```
sudo debootstrap --variant=minbase buster rootfs http://10.15.0.120:8081/repository/debian-proxy
```

使用 chroot 进入相应的镜像，进入后可执行命令操作

```
sudo chroot rootfs
```

将镜像打包导入 docker 中

```
sudo tar -C rootfs -c . | sudo docker import -
```

使用 docker tag 对导入的镜像命名

```
sudo docker tag  上一步生成的 sha256 值  baseimage
```

使用 docker images 查看导入的镜像

```
sudo docker images
```

使用 docker run 进入相应的镜像

```
sudo docker run -i -t base /bin/bash
```

3.2. 连接自建软件仓库以及镜像的简单裁剪

执行 debootstrap 构建命令时，连接自建仓库

```
sudo debootstrap --variant=minbase buster rootfs http://10.15.0.120:8085/basedeb
```

执行脚本，对镜像进行简单的裁剪，并导入 docker 内

```
cleanup() {
    IFS=$'\n'; set -o noglob
    slimExcludes=( $(grep -vE '^#|^$' "slimify-excludes" | sort -u) )
    slimIncludes=( $(grep -vE '^#|^$' "slimify-includes" | sort -u) )
    set +o noglob; unset IFS
    findMatchIncludes=()
    for slimInclude in "${slimIncludes[@]}"; do
        [ "${#findMatchIncludes[@]}" -eq 0 ] || findMatchIncludes+=( '-o' )
        findMatchIncludes+=( -path "$slimInclude" )
    done
    findMatchIncludes=( '(' "${findMatchIncludes[@]}" ')' )

    for slimExclude in "${slimExcludes[@]}"; do
        chroot "$targetdir" find "$(dirname "$slimExclude")" \
            -depth -mindepth 1 \
            -not \( -type d -o -type l \) \
            -not "${findMatchIncludes[@]}" \
```

```
            -exec rm -f '{}' ';'

            while [ "$(chroot "$targetdir" \
                                    find "$(dirname "$slimExclude")" \
                                            -depth -mindepth 1 \( -empty -o -xtype l \) \
                                            -exec rm -rf '{}' ';' -printf '.' \
                                    | wc -c
                    )" -gt 0 ]; do true; done
    done


    chroot $targetdir apt clean
    chroot $targetdir find /var/log -type f -exec rm -rf '{}' ';'
    chroot $targetdir iconvconfig
}

import() {
    tar --exclude-from tar-exclude -C $targetdir -c . | docker import - $targetname
    docker save base -o /home/user/build/images/$targetname.tar
    gzip -9 /home/user/build/images/$targetname.tar
}
```