KYLINSOFT
麒麟软件

银河麒麟桌面操作系统 V10
Qt 应用开发者打包指南

麒麟软件有限公司

生态发展中心

2025 年 12 月 17 日

版本说明

版本号	版本说明	作者	日期	变更内容
V1.0	首次发布	吴兆惠	2023-02-20	首次创建
V1.1	内容变更	吴兆惠	2023-05-30	指定拉取源码的版本; 修改链接目录；
V1.2	内容变更	吴兆惠、冯培培	2023-08-28	修订部分内容
V1.3	内容变更	吴兆惠	2024-01-11	修订部分内容
V1.4	内容变更	吴兆惠	2025-10-29	结构与标题优化、细节准确性修订、操作规范性强化、语言与实用性提升
V2.0	模板变更	吴兆惠	2025-12-17	模板变更

目录

1 概述

本文针对 银河麒麟桌面操作系统 V10 环境，详细介绍将 Qt 应用源码打包为符合《银河麒麟桌面操作系统 V10-DEB 包打包规范》deb 包的两种实现方法，为 Qt 应用开发者提供标准化打包参考，助力高效完成软件上架准备。

本文提及的 “银河麒麟桌面操作系统 V10” 包含以下版本，后续表述将做简化以提升阅读效率：

- V10 系统：特指老版本 V10（内核版本 4.4）；
- V10 (SP1) 系统：包含 V10 (SP1)（内核 5.4）与 V10 (SP1-HWE)（内核 5.10）两个子版本。

Qt 应用打包需结合实际系统环境（在线 / 离线）选择适配方法，两种方法的适用场景、优劣势及推荐度如下，开发者可按需选择：

打包方法	适用场景	核心优势	主要不足	推荐度
离线打包法	无外网连接的离线系统	无需拉取外网依赖，安装后可直接启动	依赖内置导致包体积较大	按需选择
Debian 打包法	有稳定外网连接的在线系统	符合 Debian 标准，包体积小、易维护	需从外网源拉取依赖，离线环境不友好	优先推荐

所谓的离线打包方法就是应用编译完成后，通过 linuxdeploy 或 linuxdeployqt 工具，将应用运行所需的所有依赖（含系统库、Qt 组件等）全量内置到包中，先生成 unpack 中间包（非标准 deb 格式，需二次处理），再手动封装为标准 deb 包。

具体步骤（含关键操作补充）：

1、安装 Qt 应用开发基础环境（需包含 Qt SDK、gcc、make 等工具，建议按应用依赖清单安装完整）；

2、下载应用源码包（或通过 Git 拉取源码，如 git clone 源码地址）；

3、进入源码目录，执行配置命令生成 Makefile（如 Qt 应用常用 qmake 或 cmake，需根据源码构建方式选择）；

4、运行 make 命令完成应用编译（大项目可加 -jN 多线程编译，N 为 CPU 核心数，提升效率）；

5、执行 make install DESTDIR=指定目录（建议自定义安装目录，如 ./install），将编译产物统一输出到指定路径，便于后续打包；

6、借助 linuxdeploy 或 linuxdeployqt 拉取全量依赖；

7、手动封装 deb 包：按 deb 包标准结构（DEBIAN 目录 + 应用文件目录）整理 unpack 包内容，通过 dpkg-deb -b 命令生成标准 deb 包；

8、验包：通过 dpkg -i 包名.deb 测试安装，验证应用是否能正常启动，同时检查依赖是否缺失（可通过 ldd 应用可执行文件 排查）。

Debian 打包方法遵循 Debian 官方打包规范，先通过 debmake 工具生成标准化的 debian 目录（含依赖配置、编译规则等核心文件），再用 debuild 工具自动完成“编译 - 打包 - 校验”全流程，支持通过 launchpad 平台（需提前配置）一次性编译多架构适配包。

具体步骤：

安装 Debian 打包工具与 Qt 开发环境：需安装 debmake、debuild、dh-make 等打包工具，以及 Qt 开发依赖（如 libqt5widgets5-dev 等，需按应用依赖清单安装）；

下载应用源码包（或通过 Git 拉取源码），并将源码目录按 Debian 规范重命名：格式为“软件名 - 版本号”（如 ksnip-1.10.1），避免特殊字符；

进入重命名后的源码目录，执行 debmake -y 生成 debian 目录及默认配置模板（control、rules、changelog 等文件）；

修改 debian 目录下的模板文件（核心步骤，需按软件商店规范调整）：

control：填写包名、版本、依赖（如 Depends: libqt5core5a (>= 5.9.0), libgtk-3-0）、维护者信息等；

rules：调整编译规则（如 Qt 应用需确保 qmake 路径正确，避免依赖冲突）；

changelog：按 Debian 格式填写版本更新记录（首次打包可通过 dch --create -v 版本号 -D unstable 生成）；

执行 debuild -us -uc 命令（-us -uc 表示不签名，适合测试打包；正式上架需配置 GPG 签名），自动完成编译与 deb 包生成，产物输出到源码目录上级路径；

验包：通过 lintian 包名.deb 检查 deb 包是否符合 Debian 规范（修复报错项），再通过 dpkg -i 测试安装，验证应用运行与依赖拉取是否正常。

为直观展示两种打包方法的操作细节，下文将以 开源 Qt 应用 ksnip（截图工具） 为示例，在 银河麒麟 V10 (SP1) 系统（x86_64 架构）环境中，分步演示离线打包法与 Debian 打包法的完整操作流程。

ksnip 源码获取地址：https://github.com/ksnip/ksnip（建议选择稳定版本源码，如通过 git checkout 标签名 切换到指定版本，避免开发分支兼容性问题）。

2 离线打包

2.1 安装开发环境

Qt 应用开发与编译需要线部署 Qt 开发环境，银河麒麟操作系统中可通过系统源安装基础依赖，麒麟系统默认提供的 Qt 基础开发环境包为 qt5-default，执行以下命令安装：

```
sudo apt install -y git  qt5-default
```

V10 系统中安装后默认 Qt 版本为 5.6.1，V10(SP1)系统中安装后默认 Qt 版本为 5.12.8。若项目需使用上述版本外的 Qt（如 5.15.x），需从 Qt 官网下载对应架构的离线安装包，或获取 Qt 源码后自行编译。开发与编译过程中若提示 “缺少某模块”（如 QtWidgets、QtNetwork），需根据报错信息安装对应依赖包（如 libqt5widgets5-dev、libqt5network5-dev），确保编译无依赖缺失。

2.2 拉取源码

为避免因 “缺少编译依赖模块” 导致编译报错，建议通过 git clone --recursive 命令拉取源码（该命令会同时拉取应用主源码及关联的子模块 / 依赖库），以开源 Qt 应用 ksnip 为例：

```
# 拉取 ksnip 源码（指定版本 v1.10.0，--recursive 拉取子模块）
$ git clone --recursive https://github.com/ksnip/ksnip.git -b v1.10.0
```

执行上述命令后，终端会输出类似以下的拉取日志（表示源码与子模块均拉取完成）：

```
正克隆到 'ksnip'...
remote: Enumerating objects: 20676, done.
remote: Counting objects: 100% (238/238), done.
remote: Compressing objects: 100% (111/111), done.
remote: Total 20676 (delta 146), reused 210 (delta 126), pack-reused 20438
接收对象中: 100% (20676/20676), 6.52 MiB | 523.00 KiB/s, 完成.
处理 delta 中: 100% (16103/16103), 完成.
注意：正在切换到 '07d6e164734e3dab7f90eadd0f20c15b1653f94f'。
```

2.3 生成 Makefile

生成 Makefile 需先根据源码的构建配置文件，选择对应的构建工具，核心逻辑与操作步骤如下：

2.3.1 构建工具选择原则

源码根目录中常见的构建配置文件及对应工具：

若存在 CMakeLists.txt：使用 cmake 工具生成 Makefile（本文以该场景为例）；

若存在 .pro 文件（Qt 专属配置）：使用 qmake 工具生成 Makefile（命令格式：qmake 源码路径.pro -o Makefile）；

若需可视化操作：可通过 Qt Creator（直接打开项目文件）或 VS Code（安装 CMake/QT 插件）生成 Makefile 并编译。

2.3.2 生成 Makefile 操作步骤（以 CMake 为例）

关键说明

- 推荐 “Out-of-Source 编译”：在源码目录下新建 build 子目录，进入该目录执行 cmake 命令。此方式可避免编译生成的临时文件（如 .o 目标文件、日志）污染源码目录，便于后续清理。
- 必设 CMake 变量：默认情况下，CMake 的 CMAKE_INSTALL_PREFIX 变量（指定安装路径）值非 /usr；若需将应用安装到系统默认路径（符合 deb 包规范），需显式添加参数 -DCMAKE_INSTALL_PREFIX=/usr。

具体命令（以 ksnip 源码为例）

```
$ cd ksnip
# 查看目录下是否存在 CMakeLists.txt
$ ls
CHANGELOG.md      CODINGSTYLE.md  icons          snap      translations
cmake             CONTACT.md      LICENSE.txt    src
CMakeLists.txt    desktop         README.md      tests
$ mkdir build && cd build
$ cmake .. -DCMAKE_INSTALL_PREFIX=/usr
```

2.3.3 常见报错与解决方案

2.3.3.1 报错 1：“cmake：未找到命令”

- 报错日志：

```
$ cmake .. -DCMAKE_INSTALL_PREFIX=/usr
Could not find command-not-found database. Run 'sudo apt update' to populate it.
cmake：未找到命令
```

- 原因：系统未安装 cmake 工具
- 解决方案：通过系统源安装 cmake

```
sudo apt install cmake
```

2.3.3.2 报错 2：“No CMAKE_CXX_COMPILER could be found”

- 报错日志核心信息：

```
-- The CXX compiler identification is unknown
CMake Error at CMakeLists.txt:2 (project):
  No CMAKE_CXX_COMPILER could be found.
  Tell CMake where to find the compiler by setting either the environment
  variable "CXX" or the CMake cache entry CMAKE_CXX_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.
```

- 原因：系统缺少 C++ 编译器（如 g++）及基础编译工具集。

- 解决方案：安装 build-essential 包（含 g++、make 等核心编译工具）

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

2.3.3.3 报错 3：“Could not find a package configuration file provided by 'ECM'”

- 报错日志核心信息：

```
CMake Error at CMakeLists.txt:37 (find_package):
  Could not find a package configuration file provided by "ECM" with any of
  the following names:
    ECMConfig.cmake
    ecm-config.cmake
```

- 原因：ECM（Extra CMake Modules，额外 CMake 模块集）是 ksnip 源码依赖的构建模块，系统未安装
- 解决方案：安装 extra-cmake-modules 包

```
sudo apt install extra-cmake-modules
```

2.3.3.4 报错 4：Qt5Svg 模块缺失

- 报错日志核心信息

```
-- Could NOT find XCB (missing: XFIXES) (found version "1.14")
CMake Error at /usr/lib/x86_64-linux-gnu/cmake/Qt5/Qt5Config.cmake:28 (find_package):
  Could not find a package configuration file provided by "Qt5Svg" with any
  of the following names:
    Qt5SvgConfig.cmake
    qt5svg-config.cmake
```

- 原因：ksnip 依赖 Qt 的 Qt5Svg 模块（用于 SVG 矢量图形的加载与渲染），系统未安装该模块的开发依赖包，导致 CMake 无法找到对应的配置文件。
- 解决方案：通过系统源安装 Qt5Svg 模块的开发包

```
sudo apt install libqt5svg5-dev
```

2.3.3.5 报错 5：Qt5X11Extras 模块缺失

- 报错日志核心信息：

```
-- Could NOT find XCB (missing: XFIXES) (found version "1.14")
CMake Error at /usr/lib/x86_64-linux-gnu/cmake/Qt5/Qt5Config.cmake:28 (find_package):
  Could not find a package configuration file provided by "Qt5X11Extras" with
  any of the following names:

    Qt5X11ExtrasConfig.cmake
    qt5x11extras-config.cmake
```

- 原因：ksnip 依赖 Qt 的 Qt5X11Extras 模块（用于 X11 桌面环境下的额外功能支持），系统仅安装了 Qt 基础包，未安装该模块的开发依赖。
- 解决方案：通过系统源安装 Qt5X11Extras 的开发包

```
sudo apt install libqt5x11extras5-dev
```

2.3.3.6 报错 6：kImageAnnotator 库缺失

- 报错日志核心信息

```
CMake Error at CMakeLists.txt:64 (find_package):
  By not providing "FindkImageAnnotator.cmake" in CMAKE_MODULE_PATH this
  project has asked CMake to find a package configuration file provided by
  "kImageAnnotator", but CMake did not find one.
  Could not find a package configuration file provided by "kImageAnnotator"
  (requested version 0.6.0) with any of the following names:
```

- 报错原因：kImageAnnotator 是 ksnip 的核心依赖库（用于图像标注功能），系统默认源中无该库的安装包，需手动获取适配架构的安装文件。

- 解决方案:

根据当前系统架构（amd64/arm64/mips64el/loongarch64），从以下链接下载 kImageAnnotator 相关 deb 包：

  - amd64 架构：https://wwpp.lanzoum.com/b03dzdmob 密码:2jp0
  - arm64 架构：https://wwpp.lanzoum.com/b03dzdmre 密码:5u13
  - mips64el 架构：https://wwpp.lanzoum.com/b03dzdmtg 密码:4d07
  - loongarch64 架构：https://wwpp.lanzoum.com/b03dzdmuh 密码:52qi

在 ksnip 源码目录外（或源码目录内）新建 deps 目录，用于存放下载的 deb 包，避免与源码文件混淆：

```
# 示例：在源码目录同级新建 deps 目录
$ mkdir -p ../deps
# 将下载的 deb 包移动到 deps 目录
$ mv 下载的 kImageAnnotator 相关 deb 包 ../deps/
$ ls deps/
lib0_0.6.0-1~bpo11+1kylin1_loongarch64.deb
libkimageannotator-common_0.6.0-1~bpo11+1kylin1_all.deb
libkimageannotator-dev_0.6.0-1~bpo11+1kylin1_loongarch64.deb
# 批量安装目录下所有 kImageAnnotator 相关 deb 包
$ sudo apt install ./deps/*.deb
```

2.3.3.7 报错 7：kImageAnnotator 依赖 kcolorpicker 缺失

- 报错日志核心信息

```
下列软件包有未满足的依赖关系：
 libkimageannotator-dev : 依赖: libkcolorpicker-dev 但无法安装它
 libkimageannotator0 : 依赖: libkcolorpicker0 (>= 0.2.0) 但无法安装它
E: 无法修正错误，因为您要求某些软件包保持现状，就是它们破坏了软件包间的依赖关系。
```

- 报错原因：kImageAnnotator 本身依赖 kcolorpicker 库（用于颜色选择功能），未安装该依赖会导致 kImageAnnotator 的 deb 包无法正常安装。

- 解决方案：

根据系统架构，从以下链接下载 kcolorpicker 相关 deb 包：

- amd64 架构：https://wwpp.lanzoum.com/b03dzdo0j 密码:48lk
- arm64 架构：https://wwpp.lanzoum.com/b03dzdo2b 密码:2c0v
- mips64el 架构：https://wwpp.lanzoum.com/b03dzdo4d 密码:dfa2
- loongarch64 架构：https://wwpp.lanzoum.com/b03dzdo7g 密码:ha91

将下载的 kcolorpicker deb 包移动到之前创建的 deps 目录：

```
# 将 kcolorpicker deb 包移动到 deps 目录
$ mv 下载的 kcolorpicker 相关 deb 包 ../deps/
```

再次执行批量安装命令，系统会自动处理 kcolorpicker 与 kImageAnnotator 的依赖关系：

```
$ sudo apt install ./deps/*.deb
```

安装完成后，回到 ksnip 的 build 目录，重新执行 CMake 命令即可

2.3.3.8 报错 8：Qt5LinguistTools 模块缺失（翻译功能依赖）

- 报错日志核心信息：

```
CMake Warning at translations/CMakeLists.txt:1 (find_package):
  By not providing "FindQt5LinguistTools.cmake" in CMAKE_MODULE_PATH this
  project has asked CMake to find a package configuration file provided by
  "Qt5LinguistTools", but CMake did not find one.
  Could not find a package configuration file provided by "Qt5LinguistTools"
  with any of the following names:
    Qt5LinguistToolsConfig.cmake
    qt5linguisttools-config.cmake
```

- 报错原因：ksnip 的 translations 目录（用于多语言翻译）依赖 Qt 的 Qt5LinguistTools 模块，该模块提供 qt5_add_translation 等翻译相关命令，系统未安装对应的开发包。
- 解决方案：通过系统源安装 Qt5LinguistTools 所属的开发包

```
sudo apt install -y qttools5-dev
```

2.3.3.9 报错 9：XCB::XFIXES 目标缺失（X11 图形依赖）

- 报错日志核心信息

```
CMake Error at src/CMakeLists.txt:253 (add_executable):
  Target "ksnip" links to target "XCB::XFIXES" but the target was not found.
```

- 报错原因：ksnip 依赖 XCB 协议的 XFIXES 子模块（用于窗口修复与图形渲染），系统仅安装了 XCB 基础库，未安装 XFIXES 子模块的开发依赖，导致 CMake 无法找到对应的链接目标。
- 解决方案：安装 XCB XFIXES 子模块的开发包：

```
sudo apt install -y libxcb-xfixes0-dev
```

2.3.4 最终：生成 Makefile 成功

所有依赖处理完成后，需清理 build 目录下的旧配置文件（避免缓存干扰），重新执行 CMake 命令生成 Makefile。

```
# 进入之前创建的 build 目录，删除目录内所有文件（确保无残留配置）
```

```
$ rm -rf *
# 重新执行 cmake
$ cmake .. -DCMAKE_INSTALL_PREFIX=/usr
# 执行后终端输出类似以下日志，表明 Makefile 生成成功
-- The CXX compiler identification is GNU 9.3.0
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found X11: /usr/include
-- Looking for XOpenDisplay in /usr/lib/x86_64-linux-gnu/libX11.so;/usr/lib/x86_64-linux-gnu/libXext.so
-- Looking for XOpenDisplay in /usr/lib/x86_64-linux-gnu/libX11.so;/usr/lib/x86_64-linux-gnu/libXext.so - found
-- Looking for gethostbyname
-- Looking for gethostbyname - found
-- Looking for connect
-- Looking for connect - found
-- Looking for remove
-- Looking for remove - found
-- Looking for shmat
-- Looking for shmat - found
-- XCB: XFIXES requires XCB;RENDER;SHAPE
-- XCB: XFIXES requires XCB;RENDER;SHAPE
-- XCB: XFIXES requires XCB;RENDER;SHAPE
-- Found PkgConfig: /usr/bin/pkg-config (found version "0.29.1")
-- Found XCB_XCB: /usr/lib/x86_64-linux-gnu/libxcb.so (found version "1.14")
-- Found XCB_RENDER: /usr/lib/x86_64-linux-gnu/libxcb-render.so (found version "1.14")
-- Found XCB_SHAPE: /usr/lib/x86_64-linux-gnu/libxcb-shape.so (found version "1.14")
-- Found XCB_XFIXES: /usr/lib/x86_64-linux-gnu/libxcb-xfixes.so (found version "1.14")
-- Found XCB:
/usr/lib/x86_64-linux-gnu/libxcb.so;/usr/lib/x86_64-linux-gnu/libxcb-render.so;/usr/lib/x86_64-linux-gnu/libxcb-s
hape.so;/usr/lib/x86_64-linux-gnu/libxcb-xfixes.so (found version "1.14") found components: XFIXES
-- Configuring done
-- Generating done
-- Build files have been written to: /home/kylin/ksnip/build
# 查看生成的文件
$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  desktop  Makefile  src  translations
```

至此，Makefile 生成步骤全部完成，后续可执行 make 命令进行源码编译

2.4 make 编译

Makefile 生成完成后，通过 make 命令执行源码编译。为提升编译效率，推荐使用多进程编译，使用 make -j$(nproc) 启动多进程编译，其中 $(nproc) 会自动识别当前系统的最大可用 CPU 核心数，实现编译加速（避免手动指定核心数导致资源浪费或系统卡顿）。

```
$ make -j$(nproc)
# 编译成功日志示例
Scanning dependencies of target translations
[  1%] Automatic MOC for target ksnip
******（中间编译过程省略）******
[100%] Linking CXX executable ksnip
[100%] Built target ksnip
```

2.5 执行 make install 进行安装

编译完成后，需将应用文件安装到指定目录（而非系统默认路径），才能实现后续分发与打包。直接执行 sudo make install 会将应用安装到系统 /usr 等默认目录，仅适用于本地使用，无法用于分发，因此需通过参数指定安装目录。

2.5.1 不同构建工具的安装命令

需根据生成 Makefile 时使用的工具（qmake/cmake），选择对应的目录指定参数：

- qmake 构建的应用：使用 INSTALL_ROOT 参数指定安装目录，示例：

```
make install INSTALL_ROOT=AppDir  # 将应用安装到当前目录下的 AppDir 文件夹
```

- cmake 构建的应用（本文 ksnip 场景）：使用 DESTDIR 参数指定安装目录，示例：

```
make install DESTDIR=AppDir
```

2.5.2 安装成功日志与目录结构

2.5.2.1 安装成功日志示例

执行命令后，终端会输出文件安装路径，表明应用及相关资源成功安装到 AppDir 目录：

```
$ make install DESTDIR=AppDir
[  1%] Automatic MOC for target ksnip
[  1%] Built target ksnip_autogen
*****省略********
[ 84%] Built target ksnip
[100%] Built target translations
Install the project...
-- Install configuration: ""
-- Installing: AppDir/usr/bin/ksnip # 主程序
-- Installing: AppDir/usr/share/ksnip/translations/ksnip_ar.qm # 多语言文件
*****省略********
-- Installing: AppDir/usr/share/ksnip/translations/ksnip_zh_Hant.qm
-- Installing: AppDir/usr/share/applications/org.ksnip.ksnip.desktop # 桌面快捷方式
-- Installing: AppDir/usr/share/icons/hicolor/scalable/apps/ksnip.svg # 图标文件
-- Installing: AppDir/usr/share/metainfo/org.ksnip.ksnip.appdata.xml # 应用元信息
```

2.5.2.2 查看文件目录结构

通过 ls 和 tree 命令可查看 AppDir 的目录结构，确认文件安装完整性（符合 Linux 应用标准目录布局，便于后续打包）：

```
# 查看 AppDir 根目录
$ ls AppDir
# 输出：usr  # 所有文件均在 usr 子目录下，与系统目录结构一致
usr

# 查看完整目录树
$ tree AppDir
# 输出示例（关键结构）
AppDir/
└── usr
    ├── bin                                  # 主程序目录
    │   └── ksnip                            #  应用程序可执行文件
    └── share
        ├── applications
        │   └── org.ksnip.ksnip.desktop      # 桌面快捷方式
        ├── icons
        │   └── hicolor
        │       └── scalable
        │           └── apps
        │               └── ksnip.svg   # 图标文件
        ├── ksnip                            # 应用专属资源
        │   └── translations
        │       ├── ksnip_ar.qm              #多语言文件（36 个.qm 文件，支持多语言）
        └── metainfo
            └── org.ksnip.ksnip.appdata.xml #应用商店元信息

11 directories, 38 files
```

至此，应用已按标准结构安装到指定目录，可进入下一步的离线打包流程。

2.6 打 unpack 包

执行 make install DESTDIR=AppDir 命令后，可以看到生成了 AppDir 目录，AppDir 目录已包含应用所有文件（如 AppDir/usr/bin/ksnip 可执行程序），但直接打包此目录分发到其他机器，会因依赖库缺失导致无法启动。

针对此问题，我们可以通过分析或在干净的系统环境中使用 ldd 逐步排查出应用运行时所需依赖，将这些依赖写入 control 文件的 Depends 中，这样在其他联网的机器上就可以正常获取安装依赖并且成功运行此应用了。

但对于离线系统环境，上面的方法不适用。用户在离线的系统环境下双击安装时可能会因为依赖不满足导致应用安装失败。针对此种问题，我们可以通过 linuxdeploy 或 linuxdeployqt 打包工具将应用运行所需依赖库全部打入包内，生成可离线安装运行的 unpack 包，这样提供给客户的应用就可以直接安装运行了。

下面我们介绍如何通过 linuxdeploy 或 linuxdeployqt 工具对应用进行打包，打包时选择其中一种即可。推荐使用 linuxdeploy，其对 AppDirs 结构规则更严格，且支持 Qt 插件（如 QML、翻译文件）的自动捆绑，灵活性优于 linuxdeployqt。

打 unpack 包前需要先介绍一下这两种打包工具，官网仅提供 x86-64 版本，其他架构需使用预编译包，具体编译方法可以参考官方文档，此处不再赘述。可以直接下载并安装我们编译好的包。

2.6.1 linuxdeploy 工具介绍

2.6.1.1 下载链接

- x86-64 架构：

  官网：https://github.com/linuxdeploy/linuxdeploy/releases/download/continuous/linuxdeploy-x86_64.AppImage

  预编译包：x86-64 架构：https://wwpp.lanzoum.com/iRuWd0o9v2jg 密码:23a9

- arm64 架构：https://wwpp.lanzoum.com/iZBsE0o9v1qh 密码:6v02
- loongarch64 架构：https://wwpp.lanzoum.com/i1UYs0o9v1ab 密码:aqfb

2.6.1.2 安装步骤

```
# 1. 重命名下载的 AppImage 包（以 x86-64 为例，其他架构同理）
$ mv linuxdeploy-x86_64.AppImage linuxdeploy
# 2. 赋予可执行权限
$ chmod +x linuxdeploy
# 3. 移动到系统可执行目录（全局可用）
$ sudo cp linuxdeploy /usr/bin/
```

2.6.1.3 必须插件：linuxdeploy-plugin-qt

Qt 应用需此插件捆绑 Qt 插件（如翻译文件、QML 库），否则运行时可能缺失界面资源

- 源码地址：https://github.com/linuxdeploy/linuxdeploy-plugin-qt
- 下载链接（按架构选择）：
  - x86-64 架构：https://wwpp.lanzoum.com/ibRv50o9vb7i 密码:i3zz
  - arm64 架构：https://wwpp.lanzoum.com/i6WkE0o9vaej 密码:ddi3
  - loongarch64 架构：https://wwpp.lanzoum.com/ijR3z0o9v9qf 密码:bo5b
- 插件安装步骤

```
# 1. 重命名插件包
$ mv linuxdeploy-plugin-qt-x86_64.AppImage linuxdeploy-plugin-qt
# 2. 赋予可执行权限
$ chmod +x linuxdeploy-plugin-qt
# 3. 移动到系统可执行目录
$ sudo cp linuxdeploy-plugin-qt /usr/bin/
```

2.6.2 linuxdeployqt 工具介绍

2.6.2.1 下载链接

- x86-64 架构：https://wwpp.lanzoum.com/ipftx0o9tura 密码:630z
- arm64 架构：https://wwpp.lanzoum.com/iElNK0o9tuve 密码:4vhg
- mips64el 架构：https://wwpp.lanzoum.com/ilv9V0o9tule 密码:8j2v
- loongarch64 架构：https://wwpp.lanzoum.com/iq4000o9tung 密码:az5r

2.6.2.2 安装步骤

```
# 1. 重命名
$ mv linuxdeployqt-continuous-x86_64.AppImage linuxdeployqt
# 2. 赋予权限并移动
$ chmod +x linuxdeployqt
$ sudo cp linuxdeployqt /usr/bin/
```

2.6.2.3 附加工具：appimagetool(仅打包 AppImage 时使用)

若需生成 AppImage 格式包（本文目标为 unpack 包，此步骤可选），需安装 appimagetool

- 源码地址：https://github.com/AppImage/AppImageKit
- 下载链接（按架构选择）
  - x86-64 架构：https://wwpp.lanzoum.com/iES3j0o9t5wf 密码:dqwo
  - arm64 架构：https://wwpp.lanzoum.com/igZig0o9t5qj 密码:68ek
  - loongarch64 架构：https://wwpp.lanzoum.com/iUzkL0o9t5oh 密码:1jtx
- 安装步骤

```
# 1. 重命名
$ mv appimagetool-x86_64.AppImage appimagetool
# 2. 赋予权限并移动
$ chmod +x appimagetool
$ sudo cp appimagetool /usr/bin/
```

2.6.3 使用 linuxdeploy 打 unpack 包

由于此程序为 Qt 应用，打包时需要添加--plugin qt 参数，强制调用 Qt 插件，确保 Qt 相关依赖（如翻译文件、界面插件）全量打包。

本文目标是生成 unpack 包（目录形式），用于后续封装 deb，无需生成单一 AppImage 文件，所以没有添--output appimage 参数。

确保当前目录在 ksnip 的 build 目录（AppDir 所在路径），执行打包命令

```
$ linuxdeploy --appdir AppDir --plugin qt
```

成功日志（核心片段）

```
linuxdeploy version 1-alpha (git commit ID 1d3d974), GitHub actions build 112 built on 2022-09-01 01:48:43
UTC


-- Creating basic AppDir structure --
```

```
Creating directory AppDir/usr/bin/
Creating directory AppDir/usr/lib/
******（依赖拷贝过程省略）******
[qt/stdout] Done!     # Qt 插件处理完成
-- Copying files into AppDir --
-- Deploying files into AppDir root directory --
WARNING: No desktop file specified, using first desktop file found:
AppDir/usr/share/applications/org.ksnip.ksnip.desktop
Deploying files to AppDir root using desktop file: AppDir/usr/share/applications/org.ksnip.ksnip.desktop
Deploying desktop file to AppDir root: AppDir/usr/share/applications/org.ksnip.ksnip.desktop
Creating symlink for file AppDir/usr/share/applications/org.ksnip.ksnip.desktop in/as AppDir
Deploying icon to AppDir root: AppDir/usr/share/icons/hicolor/scalable/apps/ksnip.svg
Creating symlink for file AppDir/usr/share/icons/hicolor/scalable/apps/ksnip.svg in/as AppDir
Deploying AppRun symlink for executable in AppDir root: AppDir/usr/bin/ksnip
Creating symlink for file AppDir/usr/bin/ksnip in/as AppDir/AppRun
```

查看文件目录结构

```
$ ls AppDir/
AppRun  apprun-hooks  AppRun.wrapped  ksnip.svg  org.ksnip.ksnip.desktop  usr
kylin@kylin-v10sp1-2203-amd64:~/ksnip/build$ tree AppDir/
AppDir/
├── AppRun
├── apprun-hooks
│   └── linuxdeploy-plugin-qt-hook.sh
├── AppRun.wrapped -> usr/bin/ksnip
├── ksnip.svg -> usr/share/icons/hicolor/scalable/apps/ksnip.svg
├── org.ksnip.ksnip.desktop -> usr/share/applications/org.ksnip.ksnip.desktop
└── usr
    ├── bin
    │   ├── ksnip
    │   └── qt.conf
    ├── lib
    │   ├── libavahi-client.so.3
    │   ├── libavahi-common.so.3
    │   ├── libbsd.so.0
 ******
    ├── plugins
    │   ├── bearer
    │   │   ├── libqconnmanbearer.so
    │   │   ├── libqgenericbearer.so
    │   │   └── libqnmbearer.so
    │   ├── iconengines
    │   │   └── libqsvgicon.so
    │   ├── imageformats
    │   │   ├── libqapng.so
```

```
    │   │   ├── libqgif.so
    │   │   ├── libqicns.so
    │   │   ├── libqico.so
    │   │   ├── libqjpeg.so
    │   │   ├── libqmng.so
    │   │   ├── libqsvg.so
    │   │   ├── libqtga.so
    │   │   ├── libqtiff.so
    │   │   ├── libqwbmp.so
    │   │   └── libqwebp.so
    │   ├── platforminputcontexts
    │   │   ├── libcomposeplatforminputcontextplugin.so
    │   │   ├── libfcitxplatforminputcontextplugin.so
    │   │   └── libibusplatforminputcontextplugin.so
    │   ├── platforms
    │   │   └── libqxcb.so
    │   ├── printsupport
    │   │   └── libcupsprintersupport.so
    │   └── xcbglintegrations
    │       ├── libqxcb-egl-integration.so
    │       └── libqxcb-glx-integration.so
    ├── share
    │   ├── applications
    │   │   └── org.ksnip.ksnip.desktop
    │   ├── doc
******
    │   ├── icons
    │   │   └── hicolor
    │   │       ├── 128x128
    │   │       │   └── apps
    │   │       ├── 16x16
    │   │       │   └── apps
    │   │       ├── 256x256
    │   │       │   └── apps
    │   │       ├── 32x32
    │   │       │   └── apps
    │   │       ├── 64x64
    │   │       │   └── apps
    │   │       └── scalable
    │   │           └── apps
    │   │               └── ksnip.svg
    │   ├── ksnip
    │   │   └── translations
    │   │       ├── ksnip_ar.qm
```

```
******
    │   └── metainfo
    │       └── org.ksnip.ksnip.appdata.xml
    └── translations
        ├── qtbase_ar.qm
        ├── qtbase_bg.qm
        ├── qtbase_ca.qm
        ├── qtbase_cs.qm
        ├── qtbase_da.qm
        ├── qtbase_de.qm
        ├── qtbase_en.qm
        ├── qtbase_es.qm
        ├── qtbase_fi.qm
        ├── qtbase_fr.qm
        ├── qtbase_gd.qm
        ├── qtbase_he.qm
        ├── qtbase_hu.qm
        ├── qtbase_it.qm
        ├── qtbase_ja.qm
        ├── qtbase_ko.qm
        ├── qtbase_lv.qm
        ├── qtbase_pl.qm
        ├── qtbase_ru.qm
        ├── qtbase_sk.qm
        ├── qtbase_uk.qm
        └── qtbase_zh_TW.qm

99 directories, 225 files
```

打包后 AppDir/usr/lib/ 目录会新增大量依赖库文件，表明依赖已全量打入，此时 AppDir 即为可离线运行的 unpack 包，可进入下一步 deb 封装。

2.6.4 使用 linuxdeployqt 打 unpack 包

使用 linuxdeploy 打包时只需要指定含有程序二进制文件的目录即可，但使用 linuxdeployqt 打包时需要指定 desktop 文件或二进制文件，部分应用可能未自带 desktop 文件，需要手动编写，编写规范可以参考《银河麒麟桌面操作系统 V10-DEB 包打包规范》。

此处我们以 desktop 文件作为打包的关联文件，执行 linuxdeployqt *.desktop -appimage 进行打包。打包的时候可能会有如下报错：

```
$ linuxdeployqt AppDir/usr/share/applications/org.ksnip.ksnip.desktop -appimage
linuxdeployqt 8 (commit d6ac06c), build <local dev build> built on 2022-07-06 08:53:10 UTC
FHS-like mode with PREFIX, fhsPrefix: "/home/kylin/ksnip/build/AppDir/usr"
app-binary: "/home/kylin/ksnip/build/AppDir/usr/bin/ksnip"
appDirPath: "/home/kylin/ksnip/build/AppDir"
relativeBinPath: "usr/bin/ksnip"
ERROR: Could not start patchelf.
```

```
ERROR: Make sure it is installed on your $PATH.
ERROR: Error reading rpath with patchelf "libQt5Network.so" : ""
ERROR: Error reading rpath with patchelf "libQt5Network.so" : ""
```

解决方案

```
sudo apt install patchelf -y
```

重新打包

```
$ linuxdeployqt AppDir/usr/share/applications/org.ksnip.ksnip.desktop -appimage
linuxdeployqt 8 (commit d6ac06c), build <local dev build> built on 2022-07-06 08:53:10 UTC
Desktop file as first argument: "AppDir/usr/share/applications/org.ksnip.ksnip.desktop"
desktopExecEntry: "ksnip"
desktopIconEntry: "ksnip"
******
Parallel mksquashfs: Using 2 processors
Creating 4.0 filesystem on ksnip-7520bf33-x86_64.AppImage, block size 131072.
[==========================================================================================
=======|] 835/835 100%

Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 131072
******
Marking the AppImage as executable...
Embedding MD5 digest
Success

Please consider submitting your AppImage to AppImageHub, the crowd-sourced
central directory of available AppImages, by opening a pull request
at https://github.com/AppImage/appimage.github.io
```

查看文件目录结构

```
$ ls AppDir/
AppRun    ksnip.svg    org.ksnip.ksnip.desktop    usr
$ tree AppDir/
AppDir/
├── AppRun -> usr/bin/ksnip
├── ksnip.svg
├── org.ksnip.ksnip.desktop
└── usr
    ├── bin
    │   ├── ksnip
    │   └── qt.conf
    ├── lib
    │   ├── libavahi-client.so.3
******
    ├── plugins
    │   ├── bearer
```

```
    │   │   ├── libqconnmanbearer.so
    │   │   ├── libqgenericbearer.so
    │   │   └── libqnmbearer.so
    │   ├── iconengines
    │   │   └── libqsvgicon.so
    │   ├── imageformats
    │   │   ├── libqapng.so
    │   │   ├── libqgif.so
    │   │   ├── libqicns.so
    │   │   ├── libqico.so
    │   │   ├── libqjpeg.so
    │   │   ├── libqmng.so
    │   │   ├── libqsvg.so
    │   │   ├── libqtga.so
    │   │   ├── libqtiff.so
    │   │   ├── libqwbmp.so
    │   │   └── libqwebp.so
    │   ├── platforminputcontexts
    │   │   ├── libcomposeplatforminputcontextplugin.so
    │   │   ├── libfcitxplatforminputcontextplugin.so
    │   │   └── libibusplatforminputcontextplugin.so
    │   ├── platforms
    │   │   └── libqxcb.so
    │   ├── printsupport
    │   │   └── libcupsprintersupport.so
    │   └── xcbglintegrations
    │       ├── libqxcb-egl-integration.so
    │       └── libqxcb-glx-integration.so
    ├── share
    │   ├── applications
    │   │   └── org.ksnip.ksnip.desktop
    │   ├── doc
******
    │   ├── icons
    │   │   └── hicolor
    │   │       └── scalable
    │   │           └── apps
    │   │               └── ksnip.svg
    │   ├── ksnip
    │   │   └── translations
    │   │       ├── ksnip_ar.qm
******
    │   └── metainfo
    │       └── org.ksnip.ksnip.appdata.xml
```

```
    └── translations
        ├── qt_ar.qm
******
        └── qt_zh_TW.qm

87 directories, 222 files
```

此时 AppDir 目录包含应用运行所需的全部依赖，可直接作为 unpack 包用于后续 deb 封装。

2.7 将 unpack 包封装为 deb 包

unpack 包虽可通过拷贝运行，但用户体验较差。需进一步封装为 deb 包，实现双击安装、系统菜单集成等功能，提示用户体验。

2.7.1 新建标准 deb 包目录结构

需按 Debian 打包规范新建 package 主目录，并创建以下子目录（确保文件安装路径符合系统约定）

```
# 1. 新建主目录 package
$ mkdir -p package
# 2. 创建核心子目录（按规范划分功能）
$ mkdir -p package/DEBIAN                    # 存放 deb 包配置文件（必需）
$ mkdir -p package/opt/apps                  # 存放应用主程序（自定义安装路径）
$ mkdir -p package/usr/share/applications    # 存放桌面快捷方式（desktop 文件）
$ mkdir -p package/usr/share/icons/hicolor/{16x16,32x32,48x48,64x64,128x128,256x256,scalable}/apps   # 存放多尺寸图标
```

执行 tree package 查看结构，确保符合规范：

```
$ tree package
package
├── DEBIAN
├── opt
│   └── apps
└── usr
    └── share
        ├── applications
        └── icons
            └── hicolor
                ├── 128x128
                │   └── apps
                ├── 16x16
                │   └── apps
                ├── 256x256
                │   └── apps
                ├── 32x32
                │   └── apps
                ├── 48x48
```

```
│   └── apps
├── 64x64
│   └── apps
└── scalable
    └── apps

22 directories, 0 files
```

2.7.2 编写 DEBIAN/control 文件

control 文件是 deb 包的 “身份信息”，需包含软件名称、版本、架构等必需字段，规范参考《银河麒麟桌面操作系统 V10-DEB 包打包规范》。

在 package/DEBIAN 目录下新建 control 文件：

```
$ touch package/DEBIAN/control
```

写入配置内容（以 ksnip 为例，需根据实际应用修改）：

```
# 编辑 control 文件
$ cat > package/DEBIAN/control << EOF
Package: ksnip                          # 软件包名称（小写，无特殊字符）
Version: 1.10.1                         # 版本号（需与应用版本一致）
Architecture: amd64                     # 架构（如 x86_64 对应 amd64，arm64 对应 arm64）
Maintainer: Wu Zhaohui <wuzhaohui@kylinos.cn>  # 维护者信息（姓名+邮箱）
Depends:                                # 依赖项（离线打包已内置依赖，此处可留空；在线包需填写系统依赖）
Section: utils                          # 软件分类（工具类填 utils）
Priority: optional                      # 优先级（可选填 optional）
Homepage: http://ksnip.org              # 应用官网（可选）
Description: Screenshot tool that provides many annotation features for your screenshots.  # 软件描述
EOF
```

2.7.3 迁移 unpack 包到指定目录

将之前生成的 AppDir/usr（含主程序、依赖、资源）整体拷贝到 package/opt/apps/ksnip（以应用名 ksnip 命名子目录，便于管理）

```
# -r 递归拷贝，-p 保留文件权限
$ cp -rp ksnip/build/AppDir/usr package/opt/apps/ksnip
```

2.7.4 整理桌面文件、图标与路径适配

unpack 包中已包含自动生成的 desktop 文件和图标，需迁移到 package/usr/share 对应目录（确保系统能识别桌面快捷方式和图标）

```
# 1. 迁移 desktop 文件到系统桌面目录
$ cp -rp package/opt/apps/ksnip/share/applications/* package/usr/share/applications/
# 2. 迁移图标到系统图标目录（按尺寸分类）
$ cp -rp package/opt/apps/ksnip/share/icons/hicolor/* package/usr/share/icons/hicolor/
# 3. 删除冗余目录（避免重复，减少包体积）
$ rm -rf package/opt/apps/ksnip/share   # 应用主目录仅保留 bin/lib，资源已迁移
```

```
$ rm -rf package/usr/share/doc              # 文档文件非必需，可删除
```

原始 desktop 文件中 Exec=ksnip 指向系统默认路径 /usr/bin/ksnip，但实际主程序存放在 package/opt/apps/ksnip/bin/ksnip，直接安装会导致 “找不到程序” 错误。

在 package/usr/bin 下创建链接，指向实际主程序路径：

```
# 1. 新建 /usr/bin 目录（若未存在）
$ mkdir -p package/usr/bin
# 2. 创建符号链接（让系统通过 /usr/bin/ksnip 找到实际程序）
$ ln -s ../../opt/apps/ksnip/bin/ksnip package/usr/bin/ksnip
```

查看 package/usr/share/applications/org.ksnip.ksnip.desktop，确认 Exec=ksnip 无需修改（链接已生效）

```
$ cat package/usr/share/applications/org.ksnip.ksnip.desktop
[Desktop Entry]
Type=Application
Exec=ksnip   # 链接已指向 /opt/apps/ksnip/bin/ksnip，无需修改
Icon=ksnip   # 图标已迁移到系统目录，可正常识别
Terminal=false
StartupNotify=false
Name=ksnip
# 其余字段省略...
```

deb 包需符合银河麒麟软件商店规范，推荐使用 全尺寸 PNG 图标（16x16 到 256x256）或 SVG 矢量图标（支持任意缩放，优先选择），确保在不同分辨率下显示清晰。

2.7.5 执行打包命令生成 deb 包

使用 dpkg-deb 工具打包，需通过 fakeroot 模拟 root 权限（确保文件权限正确）

```
# 格式：fakeroot dpkg-deb -b 源目录 输出目录（. 表示当前目录）
$ fakeroot   dpkg-deb -b package/ .
```

执行后终端输出如下，表明 deb 包已生成：

```
dpkg-deb: 正在 './ksnip_1.10.1_amd64.deb' 中构建软件包 'ksnip'。
```

此时当前目录会出现 ksnip_1.10.1_amd64.deb 文件，即最终的 deb 安装包。

2.8 deb 包验证

需在干净的银河麒麟 V10SP1 环境（避免本地依赖干扰）中验证

2.8.1 安装 deb 包

```
$ sudo dpkg -i ksnip_1.10.1_amd64.deb
```

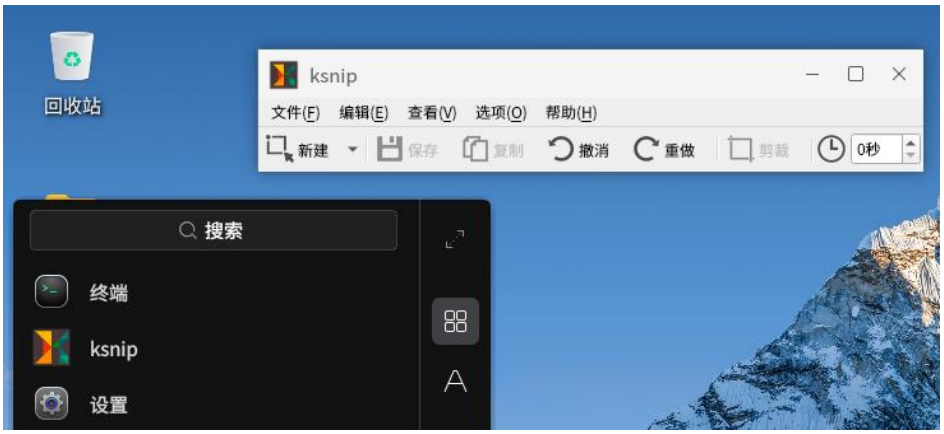
成功日志示例

```
正在选中未选择的软件包 ksnip。
(正在读取数据库 ... 系统当前共安装有 210418 个文件和目录。)
准备解压 ksnip_1.10.1_amd64.deb  ...
正在解压 ksnip (1.10.1) ...
正在设置 ksnip (1.10.1) ...
正在处理用于 desktop-file-utils (0.24-1kylin2) 的触发器 ...
```

```
正在处理用于 bamfdaemon (0.5.3+18.04.20180207.2-0kylin2) 的触发器 ...
Rebuilding /usr/share/applications/bamf-2.index...
正在处理用于 mime-support (3.64kylin1) 的触发器 ...
正在处理用于 hicolor-icon-theme (0.17-2) 的触发器 ...
```

2.8.2 验证运行

- 方法 1：从系统 “开始菜单” 找到 ksnip 图标，双击启动；
- 方法 2：终端执行 ksnip 命令，若能正常打开应用界面。



两种方式均能正常启动，即表示 deb 包打包成功，符合银河麒麟 V10 软件商店上架的基础要求。

3 Debian 打包

Debian 打包遵循标准 Debian 构建流程，生成的 deb 包体积小、维护便捷，适用于在线环境。以下是基于 ksnip 应用的完整打包步骤，含依赖处理、配置文件修改等关键操作。

3.1 安装 Debian 打包工具与核心依赖

需先部署打包工具链，并解决应用编译依赖（kimageannotator 与 kcolorpicker）。

3.1.1 安装打包基础工具

执行以下命令安装 debmake、devscripts 等必需打包工具

```
sudo apt install -y git debmake devscripts debhelper python3-debian build-essential
```

3.1.2 安装应用编译依赖（kimageannotator 与 kcolorpicker）

官方源无这两个库，需通过预编译 deb 包安装（避免手动编译）

3.1.2.1 下载预编译包

- kimageannotator 下载地址：参考 1.3.6 章节；
- kcolorpicker 下载地址：参考 1.3.7 章节。

3.1.2.2 统一存放与安装

```
# 1. 新建 deps 目录存放 deb 包（避免污染源码）
$ mkdir -p deps
# 2. 将下载的 deb 包移动到 deps 目录
$ mv 下载的 kimageannotator*.deb 下载的 kcolorpicker*.deb deps/
# 3. 离线安装所有依赖包
$ sudo apt install -y ./deps/*.deb
```

3.1.2.3 编译参考

部分应用（如 ksnip）在 GitHub 提供自动化打包配置（workflows），可参考其依赖清单与编译参数，链接：https://github.com/ksnip/ksnip/blob/master/.github/workflows/linux.yml

```
161       - name: Install Qt
162         uses: jurplel/install-qt-action@v2
163         with:
164           version: '5.12.7'
165           host: 'linux'
166           install-deps: 'true'
167
168       - name: Install dependencies
169         run: sudo apt-get install cmake extra-cmake-modules libxcb-xfixes0-dev libssl-dev devscripts debhelper
```

3.2 下载源码包或直接拉取源码

Debian 打包会在源码目录同级生成大量构建文件（如 .deb、.dsc），建议提前新建独立目录管理源码，避免文件混乱

```
# 1. 新建目录并进入
$ mkdir -p test && cd test
# 2. 递归拉取源码（--recursive 确保子模块/依赖库同步下载）
$ git clone --recursive https://github.com/ksnip/ksnip.git -b v1.10.0
```

3.3 源码目录重命名与压缩（符合 Debian 命名规范）

Debian 要求源码目录与压缩包命名格式为“包名-版本号”，且压缩包需包含隐藏文件（避免后续打包报错）

```
# 1. 重命名源码目录（包名 ksnip，版本 1.10.0）
$ mv ksnip ksnip-1.10.0
# 2. 压缩为 tar.gz 包（.[!.]* 确保隐藏文件被包含）
$ tar czf ksnip-1.10.0.tar.gz ksnip-1.10.0/* ksnip-1.10.0/.[!.]*
```

执行 ls 应看到以下文件，表明格式正确

```
$ ls
ksnip-1.10.0    ksnip-1.10.0.tar.gz
$ ls ksnip-1.10.0
CHANGELOG.md    CMakeLists.txt    CONTACT.md    icons            README.md    src       translations
cmake           CODINGSTYLE.md    desktop       LICENSE.txt      snap         tests
```

3.4 生成 debian 目录（打包配置核心）

进入源码目录，执行 debmake 工具自动生成标准化 debian 目录（含 control、rules 等模板文件）

```
# 进入源码目录
$ cd ksnip-1.10.0/
# 生成 debian 目录
$ debmake \
```

终端会输出依赖分析、文件扫描日志（如“build_type = Cmake”表明识别为 CMake 项目）；

执行 ls debian/ 查看生成的配置文件，确保包含以下关键文件

```
$ ls debian/
changelog    compat    control    copyright    patches    README.Debian    rules    source    watch
```

3.5 修改 debian 配置文件

debmake 生成的模板文件需按需修改，否则可能导致打包失败或不符合软件商店要求，重点修改以下文件。

3.5.1 修改 control 文件（包信息与依赖配置）

control 文件定义包的核心信息，需修正 Section、Build-Depends 等字段，默认模板中 Section: unknown、Build-Depends 缺失 Qt 依赖，需按以下内容修改

```
# 修改后的 control 文件：
$ cat > debian/control << EOF
Source: ksnip    # 源码包名（与应用名一致）
Section: graphics    # 应用分类（截图工具归为 graphics，非 utils）
Priority: optional
Maintainer: Wu Zhaohui <wuzhaohui@kylinos.cn>    # 维护者信息（姓名+邮箱）
# 编译依赖：含 CMake、Qt 模块、第三方库（版本需匹配）
Build-Depends: cmake,
```

```
                debhelper (>=11~),
                qtbase5-dev,
                qttools5-dev,
                extra-cmake-modules,
                libqt5x11extras5-dev,
                libkimageannotator-dev (>= 0.6.0~),
                libkcolorpicker-dev (>= 0.2.0~),
                libqt5svg5-dev,
                libx11-dev,
                libxcb-xfixes0-dev
Standards-Version: 4.1.4
Homepage: https://github.com/ksnip/ksnip   # 应用官网

Package: ksnip   # 二进制包名（与源码包名一致）
Architecture: any   # 架构（any 表示适配所有架构，也可指定 amd64/arm64）
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}   # 运行依赖（自动生成，无需手动填）
# 应用描述（需清晰说明功能，符合软件商店审核要求）
Description: Qt-based cross-platform screenshot tool
 Ksnip is a Qt-based cross-platform screenshot tool that provides many annotation features for your screenshots.
 It supports traditional UI and experimental GNOME/KDE Wayland integration.
EOF
```

3.5.2 修改 changelog 文件（版本与更新记录）

changelog 决定 deb 包版本号，需修正维护者信息与版本描述，默认模板含 “Closes: #nnnn”（bug 编号，本地打包无需）。

```
# 修改后的 changelog 文件
$ cat changelog
ksnip (1.10.0-1) UNRELEASED; urgency=low

  * Initial release.    # 首次打包，说明为初始版本

 -- Wu Zhaohui <wuzhaohui@kylinos.cn>  Thu, 16 Feb 2023 14:33:58 +0800
```

说明：

- 版本号格式：应用版本-修订号（如 1.10.0-1，修订号用于打包迭代）；
- 若需上传至 Launchpad 编译，需将 UNRELEASED 改为 focal（Ubuntu 20.04 代号，银河麒麟 V10SP1 基于此版本）

3.5.3 修改 copyright 文件（版本信息）

仅需修正 Source 字段（指向应用源码地址），其余字段（如文件版权、许可证）可保持自动识别结果。

```
# 修改 copyright 文件开头的 Source 字段
$ sed -i 's|Source: <insert the upstream URL, if relevant>|Source: https://github.com/ksnip/ksnip|' debian/copyright
```

```
# 修改后的 copyright 文件
$ cat copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: ksnip
Source: https://github.com/ksnip/ksnip
#
# Please double check copyright with the licensecheck(1) command.

Files:      src/backend/CapturePrinter.cpp
            src/backend/CapturePrinter.h
            src/backend/ITranslationLoader.h
******
```

3.5.4 patches 目录（可选：源码修改适配）

若需修改源码（如适配银河麒麟系统），需通过 patch 文件记录修改：

1、修改源码后执行 dpkg-source --commit 生成 patch；

2、patch 文件会自动存入 debian/patches/，并更新 series 文件（无需手动编辑）；

3、参考文档：https://www.debian.org/doc/manuals/debmake-doc/ch05.zh-cn.html#patches

3.5.5 rules 文件（编译规则，默认适配 CMake）

CMake 项目无需修改默认 rules 文件，若需自定义编译参数（如指定安装路径），可以参考 https://www.debian.org/doc/manuals/debmake-doc/ch05.zh-cn.html#rules，添加 override_dh_auto_configure 字段。

```
$ cat rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed


%:
	dh $@


#override_dh_auto_configure:
#	dh_auto_configure -- \
#	    -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
```

3.6 执行 debuild 打包（生成最终 deb 包）

由于未配置 GPG 签名（本地打包无需），需添加 -us -uc 参数跳过签名验证

```
$ debuild    -us -uc
```

打包成功标志

- 终端输出 “dpkg-deb: 正在 '../ksnip_1.10.0-1_amd64.deb' 中构建软件包 'ksnip'”；
- 回到上级目录（test/），执行 ls 可看到以下产物（核心为 .deb 文件）。

```
$ ls
ksnip-1.10.0                        ksnip_1.10.0-1_amd64.deb          ksnip-1.10.0.tar.gz
ksnip_1.10.0-1_amd64.build          ksnip_1.10.0-1.debian.tar.xz      ksnip-dbgsym_1.10.0-1_amd64.ddeb
ksnip_1.10.0-1_amd64.buildinfo      ksnip_1.10.0-1.dsc
ksnip_1.10.0-1_amd64.changes        ksnip_1.10.0.orig.tar.gz
```

3.7 验包（确保安装与运行正常）

需在 干净的银河麒麟 V10SP1 环境 中验证（避免本地依赖干扰）

3.7.1 安装运行依赖

运行环境仅需安装 kimageannotator 与 kcolorpicker 的运行库（无需 dev 包）。

3.7.1.1 下载运行库 deb 包

- kimageannotator 下载地址：https://wwpp.lanzoum.com/b03dzdmli 密码:h6wa
- kcolorpicker 下载地址：https://wwpp.lanzoum.com/b03dzdnzi 密码:bb2m

3.7.1.2 安装依赖

我们需要新建一个 deps 目录存放这些 deb 包，然后将对应架构的 deb 包下载到 deps 目录下，并使用 sudo apt install ./deps/*.deb 进行安装

```
$ mkdir -p deps
$ mv 下载的运行库 deb 包 deps/
$ sudo apt install -y ./deps/*.deb
```

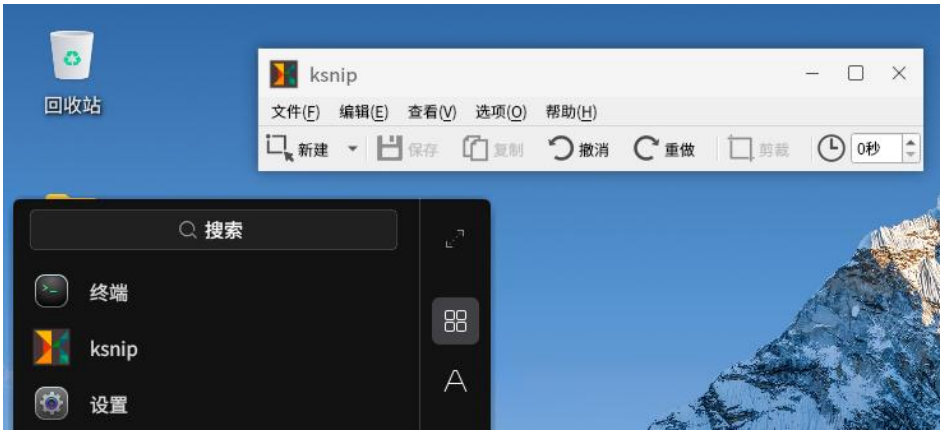
3.7.2 安装 deb 包

```
# 安装 deb 包
$ sudo apt install ./ksnip_1.10.0-1_amd64.deb
```

安装无报错，终端输出“正在设置 ksnip (1.10.0-1)”。

3.7.3 验证运行

- 方法 1：从系统 “开始菜单” 找到 ksnip 图标，双击启动；
- 方法 2：终端执行 ksnip 命令，若能正常打开应用界面。



两种方式均能正常启动，即表示 deb 包打包成功，符合银河麒麟 V10 软件商店上架的基础要求。

4 获取帮助

如有需要帮助，可联系

咨询热线	400-089-1870