Table of Contents

        - 菜单键
        - HOME键
        - 返回键
        - 音量控制
        - 媒体控制
        - 点亮/熄灭屏幕
        - 模拟滑动
        - 输入文本
    - 查看日志
        - Android日志
        - 按级别过滤日志
        - 按tag和级别过滤日志
        - 日志格式
        - 清空日志
        - 内核日志
    - 查看设备信息
        - 查看型号
        - 查看电池状况
        - 屏幕分辨率
        - 屏幕密度
        - 显示屏参数
        - android_id
        - Android系统版本
        - IP地址
        - Mac地址-需要修改
        - CPU信息
        - 内存信息
    - 修改设置
        - 修改系统默认设置项
        - 修改分辨率
        - 恢复原分辨率
        - 修改屏幕密度
        - 恢复原屏幕密度
        - 显示区域
        - 恢复显示区域
        - 关闭USB调试模式
    - 实用功能

Created by gh-md-toc

adb命令大全

基本用法

命令语法

adb 命令的基本语法如下： adb [-d|-e|-s <serialNumber>] <command>

如果只有一个设备/模拟器连接时，可以省略掉 [-d|-e|-s <serialNumber>] 这一部分，直接使用 adb <command>

为命令指定目标设备

如果有多个设备/模拟器连接，则需要为命令指定目标设备

参数	含义

-d	指定当前唯一通过 USB 连接的 Android 设备为命令目标
-e	指定当前唯一运行的模拟器为命令目标
-s <serialNumber>	指定相应 serialNumber 号的设备/模拟器为命令目标

在多个设备/模拟器连接的情况下较常用的是 -s <serialNumber> 参数，serialNumber 可以通过 adb devices 命令获取。如：

```
$ adb devices
List of devices attached
cf264b8f device
emulator-5554 device
10.129.164.6:5555 device
```

输出里的 cf264b8f、emulator-5554 和 10.129.164.6:5555 即为 serialNumber。
比如这时想指定 cf264b8f 这个设备来运行 adb 命令获取屏幕分辨率：

```
adb -s cf264b8f shell wm size
```

又如想给 10.129.164.6:5555 这个设备安装应用（这种形式的 serialNumber 格式为 :，为无线连接的设备）：

```
adb -s 10.129.164.6:5555 install test.apk
```

启动/停止

启动 adb server 命令：（一般无需手动执行此命令，在运行 adb 命令时若发现 adb server 没有启动会自动调起）

```
adb start-server
```

停止 adb server 命令：

```
adb kill-server
```

查看adb版本

命令：

```
adb version
```

示例输出：

```
Android Debug Bridge version 1.0.36
```

以root权限运行adbd

adb 的运行原理是 PC 端的 adb server 与手机端的守护进程 adbd 建立连接，然后 PC 端的 adb client 通过 adb server 转发命令，adbd 接收命令后解析运行。
所以如果 adbd 以普通权限执行，有些需要 root 权限才能执行的命令无法直接用 adb <command> 执行。这时可以 adb shell 然后 su 后执行命令，也可以让 adbd 以 root 权限执行，这个就能随意执行高权限命令了。
命令：

> adb root

正常输出：

> restarting adbd as root

现在再运行 adb shell，看看命令行提示符是不是变成 # 了？

相应地，如果要恢复 adbd 为非 root 权限的话，可以使用 adb unroot 命令.

注:部分机型不支持adb root

设备连接管理

查询已连接设备/模拟器

命令：

> adb devices

输出示例：

> List of devices attached
> cf264b8f device
> emulator-5554 device
> 10.129.164.6:5555 device

输出格式为 [serialNumber] [state]，serialNumber 即我们常说的 SN，state 有如下几种：

- offline —— 表示设备未连接成功或无响应
- device —— 设备已连接
- no device —— 没有设备/模拟器连接

以上输出显示当前已经连接了三台设备/模拟器，cf264b8f、emulator-5554 和 10.129.164.6:5555 分别是它们的 SN。从 emulator-5554 这个名字可以看出它是一个 Android 模拟器，而 10.129.164.6:5555 这种形为 : 的

serialNumber 无线连接设备.

常见异常输出：

1. 没有设备/模拟器连接成功.

   > List of devices attached

2. 设备/模拟器未连接到 adb 或无响应.

   > List of devices attached
   > cf264b8f offline

USB连接

通过 USB 连接来正常使用 adb 需要保证几点：

1. 硬件状态正常:
   包括 Android 设备处于正常开机状态，USB 连接线和各种接口完好。
2. Android 设备的开发者选项和 USB 调试模式已开启:
   可以到「设置」-「开发者选项」-「Android 调试」查看。
   如果在设置里找不到开发者选项，那需要通过一个彩蛋来让它显示出来：在「设置」-「关于手机」连续点击「版本号」7 次。
3. 设备驱动状态正常:
   这一点貌似在 Linux 和 Mac OS X 下不用操心，在 Windows 下有可能遇到需要安装驱动的情况，确认这一点可以右键「计算机」-「属性」，到「设备管理器」里查看相关设备上是否有黄色感叹号或问号，如果没有就说明驱动状态已经好了。否则可以下载一个手机助手类程序来安装驱动先。
4. 通过 USB 线连接好电脑和设备后确认状态:

   > adb devices

   说明连接成功。

   > xxxxxx device

   说明连接成功。

无线连接

除了可以通过 USB 连接设备与电脑来使用 adb，也可以通过无线连接——虽然连接过程中也有需要使用 USB 的步骤，但是连接成功之后你的设备就可以在一定范围内摆脱 USB 连接线的限制啦！　　操作步骤：

1. 将 Android 设备与要运行 adb 的电脑连接到同一个局域网内
2. 将设备与电脑通过 USB 线连接。 应确保连接成功（可运行 adb devices 看是否能列出该设备）。

3. 让设备在 5555 端口监听 TCP/IP 连接：

   > adb tcpip 5555

4. 断开 USB 连接。
5. 找到设备的 IP 地址:

   ```
   一般能在「设置」-「关于手机」-「状态信息」-「IP地址」找到，也可以使用下文里 查看设备信息 - IP 地址 一节里的方
   ```

6. 通过 IP 地址连接设备

   > adb connect <device-ip-address>

   这里的 <device-ip-address> 就是上一步中找到的设备 IP 地址。

7. 确认连接状态:

   > adb devices

   如果能看到

   > <device-ip-address>:5555 device

   说明连接成功。

如果连接不了，请确认 Android 设备与电脑是连接到了同一个网络内，然后再次执行 adb connect 那一步； 如果还是不行的话，通过 adb kill-server 重新启动 adb 然后从头再来一次试试。

断开无线连接

命令：

> adb disconnect <device-ip-address>

应用管理

查看应用列表

查看应用列表的基本命令格式是:

> adb shell pm list packages [-f] [-d] [-e] [-s] [-3] [-i] [-u] [--user USER_ID] [FILTER]

即在 adb shell pm list packages 的基础上可以加一些参数进行过滤查看不同的列表，支持的过滤参数如下：

参数	含义
无	所有应用
-f	显示应用关联的 apk 文件
-d	只显示 disabled 的应用
-e	只显示 enabled 的应用
-s	只显示系统应用
-3	只显示第三方应用
-i	显示应用的 installer
-u	包含已卸载应用
-3	只显示第三方应用
<FILTER>	包名包含 <FILTER>字符串

所有应用

命令：

```
adb shell pm list packages
```

输出示例：

```
package:com.android.smoketest
package:com.example.android.livecubes
package:com.android.providers.telephony
package:com.google.android.googlequicksearchbox
package:com.android.providers.calendar
package:com.android.providers.media
package:com.android.protips
package:com.android.documentsui
package:com.android.gallery
package:com.android.externalstorage
...
// other packages here
...
```

系统应用

命令：

```
adb shell pm list packages -s
```

第三方应用

命令：

```
adb shell pm list packages -3
```

包名包含某字符串的应用

比如要查看包名包含字符串 mazhuang 的应用列表，命令：

```
adb shell pm list packages mazhuang
```

当然也可以使用 grep 来过滤：

```
adb shell pm list packages | grep mazhuang
```

安装APK

命令格式：

```
adb install [-lrtsdg] <path_to_apk>
```

参数：
adb install 后面可以跟一些可选参数来控制安装 APK 的行为，可用参数及含义如下：

参数	含义
-l	将应用安装到保护目录 /mnt/asec
-r	允许覆盖安装
-t	允许安装 AndroidManifest.xml 里 application 指定 android:testOnly="true" 的应用
-s	将应用安装到 sdcard
-d	允许降级覆盖安装
-g	授予所有运行时权限

运行命令后如果见到类似如下输出（状态为 Success）代表安装成功：

```
[100%] /data/local/tmp/1.apk
pkg: /data/local/tmp/1.apk
```

> Success

而如果状态为 Failure 则表示安装失败,并显示失败原因，比如：

> [100%] /data/local/tmp/map-20160831.apk
> pkg: /data/local/tmp/map-20160831.apk
> Failure [INSTALL_FAILED_ALREADY_EXISTS]

常见安装失败输出代码、含义及可能的解决办法如下：

输出	含义
INSTALL_FAILED_ALREADY_EXISTS	应用已经存在，或卸载了但没卸载干
INSTALL_FAILED_INVALID_APK	无效的 APK 文件
INSTALL_FAILED_INVALID_URI	无效的 APK 文件名
INSTALL_FAILED_INSUFFICIENT_STORAGE	空间不足
INSTALL_FAILED_DUPLICATE_PACKAGE	已经存在同名程序
INSTALL_FAILED_NO_SHARED_USER	请求的共享用户不存在
INSTALL_FAILED_UPDATE_INCOMPATIBLE	以前安装过同名应用，但卸载时数据 或者已安装该应用，但签名不一致
INSTALL_FAILED_SHARED_USER_INCOMPATIBLE	请求的共享用户存在但签名不一致
INSTALL_FAILED_MISSING_SHARED_LIBRARY	安装包使用了设备上不可用的共享库
INSTALL_FAILED_REPLACE_COULDNT_DELETE	替换时无法删除
INSTALL_FAILED_DEXOPT	dex 优化验证失败或空间不足
INSTALL_FAILED_OLDER_SDK	设备系统版本低于应用要求
INSTALL_FAILED_CONFLICTING_PROVIDER	设备里已经存在与应用里同名的 con provider
INSTALL_FAILED_NEWER_SDK	设备系统版本高于应用要求
INSTALL_FAILED_TEST_ONLY	应用是 test-only 的，但安装时没有指 `t` 参数
INSTALL_FAILED_CPU_ABI_INCOMPATIBLE	包含不兼容设备 CPU 应用程序二进制 native code
INSTALL_FAILED_MISSING_FEATURE	应用使用了设备不可用的功能

INSTALL_FAILED_CONTAINER_ERROR	1. sdcard 访问失败; 2. 应用签名与 ROM 签名一致， 被当作内置应用。
INSTALL_FAILED_INVALID_INSTALL_LOCATION	1. 不能安装到指定位置; 2. 应用签名与 ROM 签名一致， 被当作内置应用。
INSTALL_FAILED_MEDIA_UNAVAILABLE	安装位置不可用
INSTALL_FAILED_VERIFICATION_TIMEOUT	验证安装包超时
INSTALL_FAILED_VERIFICATION_FAILURE	验证安装包失败
INSTALL_FAILED_PACKAGE_CHANGED	应用与调用程序期望的不一致
INSTALL_FAILED_UID_CHANGED	以前安装过该应用，与本次分配的 U
INSTALL_FAILED_VERSION_DOWNGRADE	已经安装了该应用更高版本
INSTALL_FAILED_PERMISSION_MODEL_DOWNGRADE	已安装 target SDK 支持运行时权限的 要安装的版本不支持运行时权限
INSTALL_PARSE_FAILED_NOT_APK	指定路径不是文件，或不是以 `.apk`
INSTALL_PARSE_FAILED_BAD_MANIFEST	无法解析的 AndroidManifest.xml 文
INSTALL_PARSE_FAILED_UNEXPECTED_EXCEPTION	解析器遇到异常
INSTALL_PARSE_FAILED_NO_CERTIFICATES	安装包没有签名
INSTALL_PARSE_FAILED_INCONSISTENT_CERTIFICATES	已安装该应用，且签名与 APK 文件不
INSTALL_PARSE_FAILED_CERTIFICATE_ENCODING	解析 APK 文件时遇到 `CertificateEncoding`
INSTALL_PARSE_FAILED_BAD_PACKAGE_NAME	manifest 文件里没有或者使用了无效
INSTALL_PARSE_FAILED_BAD_SHARED_USER_ID	manifest 文件里指定了无效的共享用
INSTALL_PARSE_FAILED_MANIFEST_MALFORMED	解析 manifest 文件时遇到结构性错误
INSTALL_PARSE_FAILED_MANIFEST_EMPTY	在 manifest 文件里找不到找可操作标 （instrumentation 或 application）
INSTALL_FAILED_INTERNAL_ERROR	因系统问题安装失败
INSTALL_FAILED_USER_RESTRICTED	

	用户被限制安装应用
INSTALL_FAILED_DUPLICATE_PERMISSION	应用尝试定义一个已经存在的权限名
INSTALL_FAILED_NO_MATCHING_ABIS	应用包含设备的应用程序二进制接口 native code
INSTALL_CANCELED_BY_USER	应用安装需要在设备上确认， 但未操作设备或点了取消
INSTALL_FAILED_ACWF_INCOMPATIBLE	应用程序与设备不兼容
does not contain AndroidManifest.xml	无效的 APK 文件
is not a valid zip file	无效的 APK 文件
Offline	设备未连接成功
unauthorized	设备未授权允许调试
error: device not found	没有连接成功的设备
protocol failure	设备已断开连接
Unknown option: -s	Android 2.2 以下不支持安装到 sdcar
No space left on device	空间不足
Permission denied ... sdcard ...	sdcard 不可用
signatures do not match the previously installed version; ignoring!	已安装该应用且签名不一致

参考：PackageManager.java

adb install 内部原理简介

adb install 实际是分三步完成：

> push apk 文件到 /data/local/tmp
> 调用 pm install 安装
> 删除 /data/local/tmp 下的对应 apk 文件。

所以，必要的时候也可以根据这个步骤，手动分步执行安装过程。

卸载应用

命令：

>

> adb uninstall [-k] <packagename>

<packagename> 表示应用的包名，-k 参数可选，表示卸载应用但保留数据和缓存目录。 命令示例：

> adb uninstall com.qihoo360.mobilesafe

表示卸载 360 手机卫士。

查看前台Activity

命令：

> adb shell dumpsys activity activities | grep mFocusedActivity

输出示例：

> mFocusedActivity: ActivityRecord{8079d7e u0
> com.cyanogenmod.trebuchet/com.android.launcher3.Launcher t42}

其中的 com.cyanogenmod.trebuchet/com.android.launcher3.Launcher 就是当前处于前台的 Activity。

查看正在运行的Services

命令：

> adb shell dumpsys activity services [<packagename>]

<packagename> 参数是可选的，指定 <packagename> 表示查看与某个包名相关的 Services，不指定表示查看所有 Services。

<packagename> 不用给出完整包名，比如运行 adb shell dumpsys activity services org.mazhuang，那么包名 org.mazhuang.demo1、org.mazhuang.demo2 和 org.mazhuang123 等相关的 Services 都会列出来。

查看应用详细信息

命令：

> adb shell dumpsys package <packagename>

输出中包含很多信息，包括 Activity Resolver Table、Registered ContentProviders、包名、userId、安装后的文件资源代码等路径、版本信息、权限信息和授予状态、签名版本信息等。

<packagename> 表示应用包名。

输出示例：

```
Activity Resolver Table:
  Non-Data Actions:
      android.intent.action.MAIN:
        5b4cba8 org.mazhuang.guanggoo/.SplashActivity filter 5ec9dcc
          Action: "android.intent.action.MAIN"
          Category: "android.intent.category.LAUNCHER"
          AutoVerify=false

Registered ContentProviders:
  org.mazhuang.guanggoo/com.tencent.bugly.beta.utils.BuglyFileProvider:
    Provider{7a3c394 org.mazhuang.guanggoo/com.tencent.bugly.beta.utils.BuglyFileProvider}

ContentProvider Authorities:
  [org.mazhuang.guanggoo.fileProvider]:
    Provider{7a3c394 org.mazhuang.guanggoo/com.tencent.bugly.beta.utils.BuglyFileProvider}
      applicationInfo=ApplicationInfo{7754242 org.mazhuang.guanggoo}

Key Set Manager:
  [org.mazhuang.guanggoo]
      Signing KeySets: 501

Packages:
  Package [org.mazhuang.guanggoo] (c1d7f):
    userId=10394
    pkg=Package{55f714c org.mazhuang.guanggoo}
    codePath=/data/app/org.mazhuang.guanggoo-2
    resourcePath=/data/app/org.mazhuang.guanggoo-2
    legacyNativeLibraryDir=/data/app/org.mazhuang.guanggoo-2/lib
    primaryCpuAbi=null
    secondaryCpuAbi=null
    versionCode=74 minSdk=15 targetSdk=25
    versionName=1.1.74
    splits=[base]
    apkSigningVersion=2
    applicationInfo=ApplicationInfo{7754242 org.mazhuang.guanggoo}
    flags=[ HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
    privateFlags=[ RESIZEABLE_ACTIVITIES ]
    dataDir=/data/user/0/org.mazhuang.guanggoo
    supportsScreens=[small, medium, large, xlarge, resizeable, anyDensity]
    timeStamp=2017-10-22 23:50:53
    firstInstallTime=2017-10-22 23:50:25
    lastUpdateTime=2017-10-22 23:50:55
    installerPackageName=com.miui.packageinstaller
    signatures=PackageSignatures{af09595 [53c7caa2]}
    installPermissionsFixed=true installStatus=1
    pkgFlags=[ HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
    requested permissions:
      android.permission.READ_PHONE_STATE
      android.permission.INTERNET
      android.permission.ACCESS_NETWORK_STATE
      android.permission.ACCESS_WIFI_STATE
      android.permission.READ_LOGS
      android.permission.WRITE_EXTERNAL_STORAGE
      android.permission.READ_EXTERNAL_STORAGE
    install permissions:
      android.permission.INTERNET: granted=true
      android.permission.ACCESS_NETWORK_STATE: granted=true
      android.permission.ACCESS_WIFI_STATE: granted=true
    User 0: ceDataInode=1155675 installed=true hidden=false suspended=false stopped=true notLaunched=false enabled
      gids=[3003]
```

```
      runtime permissions:
        android.permission.READ_EXTERNAL_STORAGE: granted=true
        android.permission.READ_PHONE_STATE: granted=true
        android.permission.WRITE_EXTERNAL_STORAGE: granted=true
    User 999: ceDataInode=0 installed=false hidden=false suspended=false stopped=true notLaunched=true enabled=0
      gids=[3003]
      runtime permissions:


Dexopt state:
  [org.mazhuang.guanggoo]
    Instruction Set: arm64
      path: /data/app/org.mazhuang.guanggoo-2/base.apk
      status: /data/app/org.mazhuang.guanggoo-2/oat/arm64/base.odex [compilation_filter=speed-profile, status=kOatU
      te]
```

与应用交互

主要是使用 am <command> 命令，常用的<command> 如下：

command	用途
start [options] <INTENT>	启动 <INTENT> 指定的 Activity
startservice [options] <INTENT>	启动 <INTENT> 指定的 Service
broadcast [options] <INTENT>	发送 <INTENT> 指定的广播
force-stop <packagename>	停止 <packagename>相关的进程

参数很灵活，和写 Android 程序时代码里的 Intent 相对应。

用于决定 intent 对象的选项如下：

参数	含义
-a <ACTION>	指定 action，比如 android.intent.action.VIEW
-c <CATEGORY>	指定 category，比如 android.intent.category.APP_CONTACTS
-n <COMPONENT>	指定完整 component 名，用于明确指定启动哪个 Activity，如 com.example.app/.ExampleActivity

<INTENT> 里还能带数据，就像写代码时的 Bundle 一样：

参数	含义
--esn <EXTRA_KEY>	null 值（只有 key 名）
-e	--es <EXTRA_KEY> <EXTRA_STRING_VALUE>

--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE>	boolean 值
--ei <EXTRA_KEY> <EXTRA_INT_VALUE>	integer 值
--el <EXTRA_KEY> <EXTRA_LONG_VALUE>	long 值
--ef <EXTRA_KEY> <EXTRA_FLOAT_VALUE>	float 值
--eu <EXTRA_KEY> <EXTRA_URI_VALUE>	URI
--ecn <EXTRA_KEY> <EXTRA_COMPONENT_NAME_VALUE>	component name
--eia <EXTRA_KEY> <EXTRA_INT_VALUE>[, <EXTRA_INT_VALUE...]	integer 数组
--ela <EXTRA_KEY> <EXTRA_LONG_VALUE>[, <EXTRA_LONG_VALUE...]	long 数组

启动Activity

命令格式：

> adb shell am start [options] <INTENT>

例如：

> adb shell am start -n com.tencent.mm/.ui.LauncherUI

表示调起微信主界面.

> adb shell am start -n org.mazhuang.boottimemeasure/.MainActivity --es "toast" "hello, world"

表示调起 org.mazhuang.boottimemeasure/.MainActivity 并传给它 string 数据键值对 toast - hello, world。

启动Service

命令格式：

> adb shell am startservice [options] <INTENT>

例如：

> adb shell am startservice -n com.tencent.mm/.plugin.accountsync.model.AccountAuthenticatorService

表示调起微信的某 Service

发送广播

命令格式：

```
adb shell am broadcast [options]
```

可以向所有组件广播，也可以只向指定组件广播。
例如，向所有组件广播 BOOT_COMPLETED：

```
adb shell am broadcast -a android.intent.action.BOOT_COMPLETED
```

又例如，只向 org.mazhuang.boottimemeasure/.BootCompletedReceiver 广播 BOOT_COMPLETED：

```
adb shell am broadcast -a android.intent.action.BOOT_COMPLETED -n
org.mazhuang.boottimemeasure/.BootCompletedReceiver
```

这类用法在测试的时候很实用，比如某个广播的场景很难制造，可以考虑通过这种方式来发送广播。 既能发送系统预定义的广播，也能发送自定义广播。如下是部分系统预定义广播及正常触发时机：

action	触发时机
android.net.conn.CONNECTIVITY_CHANGE	网络连接发生变化
android.intent.action.SCREEN_ON	屏幕点亮
android.intent.action.SCREEN_OFF	屏幕熄灭
android.intent.action.BATTERY_LOW	电量低，会弹出电量低提示框
android.intent.action.BATTERY_OKAY	电量恢复了
android.intent.action.BOOT_COMPLETED	设备启动完毕
android.intent.action.DEVICE_STORAGE_LOW	存储空间过低
android.intent.action.DEVICE_STORAGE_OK	存储空间恢复
android.intent.action.PACKAGE_ADDED	安装了新的应用
android.net.wifi.STATE_CHANGE	WiFi 连接状态发生变化
android.net.wifi.WIFI_STATE_CHANGED	WiFi 状态变为启用/关闭/正在启动/正在关闭/未知
android.intent.action.BATTERY_CHANGED	电池电量发生变化
android.intent.action.INPUT_METHOD_CHANGED	系统输入法发生变化
android.intent.action.ACTION_POWER_CONNECTED	外部电源连接
android.intent.action.ACTION_POWER_DISCONNECTED	外部电源断开连接
android.intent.action.DREAMING_STARTED	系统开始休眠

android.intent.action.DREAMING_STOPPED	系统停止休眠
android.intent.action.WALLPAPER_CHANGED	壁纸发生变化
android.intent.action.HEADSET_PLUG	插入耳机
android.intent.action.MEDIA_UNMOUNTED	卸载外部介质
android.intent.action.MEDIA_MOUNTED	挂载外部介质
android.os.action.POWER_SAVE_MODE_CHANGED	省电模式开启

强制停止应用

命令：

> adb shell am force-stop <packagename>

命令示例：

> adb shell am force-stop com.qihoo360.mobilesafe

表示停止 360 安全卫士的一切进程与服务。

文件管理

复制设备里的文件到电脑

命令：

> adb pull <设备里的文件路径> [电脑上的目录]

其中 电脑上的目录 参数可以省略，默认复制到当前目录. 例：

> adb pull /sdcard/sr.mp4 ~/tmp/

*小技巧：*设备上的文件路径可能需要 root 权限才能访问，如果你的设备已经 root 过，可以先使用 adb shell 和 su 命令在 adb shell 里获取 root 权限后，先 cp /path/on/device /sdcard/filename 将文件复制到 sdcard，然后 adb pull /sdcard/filename /path/on/pc。

复制电脑里的文件到设备

命令：

adb push <电脑上的文件路径> <设备里的目录> 例：

adb push ~/sr.mp4 /sdcard/ *小技巧：*设备上的文件路径普通权限可能无法直接写入，如果你的设备已经 root 过，可以先 adb push /path/on/pc /sdcard/filename，然后 adb shell 和 su 在 adb shell 里获取 root 权限后，cp /sdcard/filename /path/on/device。

模拟按键/输入

在 adb shell 里有个很实用的命令叫 input，通过它可以做一些有趣的事情

input 命令的完整 help 信息如下：

Usage: input [] [...]

```
The sources are:
      mouse
      keyboard
      joystick
      touchnavigation
      touchpad
      trackball
      stylus
      dpad
      gesture
      touchscreen
      gamepad

The commands and default sources are:
      text <string> (Default: touchscreen)
      keyevent [--longpress] <key code number or name> ... (Default: keyboard)
      tap <x> <y> (Default: touchscreen)
      swipe <x1> <y1> <x2> <y2> [duration(ms)] (Default: touchscreen)
      press (Default: trackball)
      roll <dx> <dy> (Default: trackball)
```

比如使用 adb shell input keyevent 命令，来模拟按键，完整的 keycode 列表详见 KeyEvent

下面是 input 命令的一些用法举例.

电源键

> adb shell input keyevent 26
>
> 执行效果相当于按电源键。

菜单键

> adb shell input keyevent 82

HOME键

> adb shell input keyevent 3

返回键

> adb shell input keyevent 4

音量控制

增加音量：

> adb shell input keyevent 24

降低音量：

> adb shell input keyevent 25

静音：

> adb shell input keyevent 164

媒体控制

播放/暂停：

> adb shell input keyevent 85

停止播放：

> adb shell input keyevent 86

播放下一首：

> adb shell input keyevent 87

播放上一首：

> adb shell input keyevent 88

恢复播放：

> adb shell input keyevent 126

暂停播放：

> adb shell input keyevent 127

点亮/熄灭屏幕

我们可以通过模拟电源键来点亮和熄灭屏幕，但如果明确地想要点亮或者熄灭屏幕，那可以使用如下方法.
点亮屏幕：

> adb shell input keyevent 224

熄灭屏幕：

> adb shell input keyevent 223

模拟滑动

如果锁屏没有密码，是通过滑动手势解锁，那么可以通过 input swipe 来解锁.

> adb shell input swipe 300 1000 300 500

参数 300 1000 300 500 分别表示起始点x坐标 起始点y坐标 结束点x坐标 结束点y坐标。

输入文本

在焦点处于某文本框时，可以通过 input 命令来输入文本。

> adb shell input text hello
> 输入文本hello

查看日志

Android 系统的日志分为两部分，底层的 Linux 内核日志输出到 /proc/kmsg，Android 的日志输出到 /dev/log.

Android日志

命令格式：

> [adb] logcat [<option>] ... [<filter-spec>] ...

按级别过滤日志

Android 的日志分为如下几个优先级（priority）：

- V —— Verbose（最低，输出得最多）
- D —— Debug
- I —— Info
- W —— Warning
- E —— Error

- F —— Fatal
- S —— Silent（最高，啥也不输出）

按某级别过滤日志则会将该级别及以上的日志输出.
比如，命令：

> adb logcat *:W

会将 Warning、Error、Fatal 和 Silent 日志输出.

按tag和级别过滤日志

<filter-spec> 可以由多个 <tag>[:priority] 组成.
比如，命令：

> adb logcat ActivityManager:I MyApp:D *:S

表示输出 tag ActivityManager 的 Info 以上级别日志，输出 tag MyApp 的 Debug 以上级别日志，及其它 tag 的 Silent 级别日志（即屏蔽其它 tag 日志）。

日志格式

可以用 adb logcat -v <format> 选项指定日志输出格式.
日志支持按以下几种 <format>：

- brief
  默认格式。格式为：

  > <priority>/<tag>(<pid>): <message>

  示例：

  > D/HeadsetStateMachine(1785): Disconnected process message: 10, size: 0

- process 格式为：

  > <priority>(<pid>) <message>

  示例：

  > D(1785) Disconnected process message: 10, size: 0 (HeadsetStateMachine)

- tag
  格式为：

> <priority>/<tag>: <message>

示例：

> D/HeadsetStateMachine: Disconnected process message: 10, size: 0

- raw 格式为：

> <message>

示例：

> Disconnected process message: 10, size: 0

- time
  格式为：

> <datetime> <priority>/<tag>(<pid>): <message>

示例：

> 08-28 22:39:39.974 D/HeadsetStateMachine(1785): Disconnected process message: 10, size: 0

- threadtime 格式为：

> <datetime> <pid> <tid> <priority> <tag>: <message>
> 示例：
> 08-28 22:39:39.974 1785 1832 D HeadsetStateMachine: Disconnected process message: 10, size: 0

- long
  格式为：

> [<datetime> <pid>:<tid> <priority>/<tag>] <message&gt

示例：

> [08-28 22:39:39.974 1785: 1832 D/HeadsetStateMachine] Disconnected process message: 10, size: 0

指定格式可与上面的过滤同时使用。比如：

> adb logcat -v long ActivityManager:I *:S

清空日志

> adb logcat -c

内核日志

命令：

> adb shell dmesg

输出示例：

```
<6>[14201.684016] PM: noirq resume of devices complete after 0.982 msecs
<6>[14201.685525] PM: early resume of devices complete after 0.838 msecs
<6>[14201.753642] PM: resume of devices complete after 68.106 msecs
<4>[14201.755954] Restarting tasks ... done.
<6>[14201.771229] PM: suspend exit 2016-08-28 13:31:32.679217193 UTC
<6>[14201.872373] PM: suspend entry 2016-08-28 13:31:32.780363596 UTC
<6>[14201.872498] PM: Syncing filesystems ... done.
```

中括号里的 [14201.684016] 代表内核开始启动后的时间，单位为秒。

通过内核日志我们可以做一些事情，比如衡量内核启动时间，在系统启动完毕后的内核日志里找到 Freeing init memory 那一行前面的时间就是.

查看设备信息

查看型号

> adb shell getprop ro.product.model

输出示例:

Firefly-RK3288

查看电池状况

> adb shell dumpsys battery

输入示例：

```
Current Battery Service state:
  AC powered: false
  USB powered: true
  Wireless powered: false
  status: 2
  health: 2
  present: true
  level: 44
  scale: 100
  voltage: 3872
```

```
  temperature: 280
  technology: Li-poly
```

其中 scale 代表最大电量，level 代表当前电量。上面的输出表示还剩下 44% 的电量。

屏幕分辨率

> adb shell wm size

输出示例：

> Physical size: 1080x1920

该设备屏幕分辨率为 1080 * 1920

如果使用命令修改过，那输出可能是：

Physical size: 1080x1920

Override size: 480x1024

表明设备的屏幕分辨率原本是 1080 *1920，当前被修改为 480* 1024

屏幕密度

> adb shell wm density

输出示例： Physical density: 420 该设备屏幕密度为 420dpi。

如果使用命令修改过，那输出可能是：

Physical density: 480

Override density: 160

表明设备的屏幕密度原来是 480dpi，当前被修改为 160dpi.

显示屏参数

> adb shell dumpsys window displays

输出示例：

```
WINDOW MANAGER DISPLAY CONTENTS (dumpsys window displays)
  Display: mDisplayId=0
    init=1080x1920 420dpi cur=1080x1920 app=1080x1794 rng=1080x1017-1810x1731
    deferred=false layoutNeeded=false
```

其中 mDisplayId 为 显示屏编号，init 是初始分辨率和屏幕密度，app 的高度比 init 里的要小，表示屏幕底部有虚拟按键，高度为 1920 - 1794 = 126px 合 42dp。

android_id

> adb shell settings get secure android_id

输出示例：

> 51b6be48bac8c569

Android系统版本

> adb shell getprop ro.build.version.release

输出示例：

> 5.1.1

IP地址

每次想知道设备的 IP 地址的时候都得「设置」-「关于手机」-「状态信息」-「IP地址」很烦对不对？通过 adb 可以方便地查看。

> adb shell ifconfig | grep Mask

输出示例：

```
inet addr:10.130.245.230  Mask:255.255.255.252
inet addr:127.0.0.1  Mask:255.0.0.0
```

那么 10.130.245.230 就是设备 IP 地址.
在有的设备上这个命令没有输出，如果设备连着 WiFi，可以使用如下命令来查看局域网 IP：

> adb shell ifconfig wlan0

输出示例：

```
wlan0: ip 10.129.160.99 mask 255.255.240.0 flags [up broadcast running multicast]
```

或

```
wlan0     Link encap:UNSPEC
          inet addr:10.129.168.57  Bcast:10.129.175.255  Mask:255.255.240.0
          inet6 addr: fe80::66cc:2eff:fe68:b6b6/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:496520 errors:0 dropped:0 overruns:0 frame:0
          TX packets:68215 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:3000
          RX bytes:116266821 TX bytes:8311736
```

如果以上命令仍然不能得到期望的信息，那可以试试以下命令（部分系统版本里可用）：

> adb shell netcfg

输出示例：

```
wlan0    UP                                10.129.160.99/20  0x00001043 f8:a9:d0:17:42:4d
lo       UP                                   127.0.0.1/8   0x00000049 00:00:00:00:00:00
p2p0     UP                                     0.0.0.0/0   0x00001003 fa:a9:d0:17:42:4d
sit0     DOWN                                   0.0.0.0/0   0x00000080 00:00:00:00:00:00
rmnet0   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet1   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet3   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet2   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet4   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet6   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet5   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rmnet7   DOWN                                   0.0.0.0/0   0x00000000 00:00:00:00:00:00
rev_rmnet3 DOWN                                   0.0.0.0/0   0x00001002 4e:b7:e4:2e:17:58
rev_rmnet2 DOWN                                   0.0.0.0/0   0x00001002 4e:f0:c8:bf:7a:cf
rev_rmnet4 DOWN                                   0.0.0.0/0   0x00001002 a6:c0:3b:6b:c4:1f
rev_rmnet6 DOWN                                   0.0.0.0/0   0x00001002 66:bb:5d:64:2e:e9
rev_rmnet5 DOWN                                   0.0.0.0/0   0x00001002 0e:1b:eb:b9:23:a0
rev_rmnet7 DOWN                                   0.0.0.0/0   0x00001002 7a:d9:f6:81:40:5a
rev_rmnet8 DOWN                                   0.0.0.0/0   0x00001002 4e:e2:a9:bb:d0:1b
rev_rmnet0 DOWN                                   0.0.0.0/0   0x00001002 fe:65:d0:ca:82:a9
rev_rmnet1 DOWN                                   0.0.0.0/0   0x00001002 da:d8:e8:4f:2e:fe
```

可以看到网络连接名称、启用状态、IP 地址和 Mac 地址等信息。

Mac地址-需要修改

> adb shell cat /sys/class/net/wlan0/address

输出示例：

> f8:a9:d0:17:42:4d

这查看的是局域网 Mac 地址，移动网络或其它连接的信息可以通过前面的小节「IP 地址」里提到的 adb shell netcfg 命令来查看。

CPU信息

> adb shell cat /proc/cpuinfo

输出示例：

```
processor    : 0
BogoMIPS    : 48.00
Features    : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer    : 0x41
CPU architecture: 8
CPU variant    : 0x0
CPU part    : 0xd03
CPU revision    : 4

processor    : 1
BogoMIPS    : 48.00
Features    : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer    : 0x41
CPU architecture: 8
CPU variant    : 0x0
CPU part    : 0xd03
CPU revision    : 4

processor    : 2
BogoMIPS    : 48.00
Features    : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer    : 0x41
CPU architecture: 8
CPU variant    : 0x0
CPU part    : 0xd03
CPU revision    : 4

processor    : 3
BogoMIPS    : 48.00
Features    : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer    : 0x41
CPU architecture: 8
CPU variant    : 0x0
CPU part    : 0xd03
CPU revision    : 4

processor    : 4
BogoMIPS    : 48.00
Features    : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer    : 0x41
CPU architecture: 8
CPU variant    : 0x0
CPU part    : 0xd08
CPU revision    : 2

processor    : 5
BogoMIPS    : 48.00
Features    : fp asimd evtstrm aes pmull sha1 sha2 crc32
CPU implementer    : 0x41
CPU architecture: 8
CPU variant    : 0x0
CPU part    : 0xd08
CPU revision    : 2

Serial        : 732bc487542e2683
```

内存信息

> adb shell cat /proc/meminfo

输出示例：

```
MemTotal:        2021012 kB
MemFree:          609396 kB
MemAvailable:    1268992 kB
Buffers:            1460 kB
Cached:           739608 kB
SwapCached:            0 kB
Active:           606456 kB
Inactive:         550048 kB
Active(anon):     419560 kB
Inactive(anon):    69168 kB
Active(file):     186896 kB
Inactive(file):   480880 kB
Unevictable:         256 kB
Mlocked:             256 kB
SwapTotal:        520908 kB
SwapFree:         520908 kB
Dirty:                 4 kB
Writeback:             0 kB
AnonPages:        415692 kB
Mapped:           550476 kB
Shmem:             73308 kB
Slab:             126040 kB
SReclaimable:      85024 kB
SUnreclaim:        41016 kB
KernelStack:       17216 kB
PageTables:        24024 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1531412 kB
Committed_AS:   56809824 kB
VmallocTotal:   258867136 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
CmaTotal:          16384 kB
CmaFree:           13928 kB
```

其中，MemTotal 就是设备的总内存，MemFree 是当前空闲内存.

修改设置

修改系统默认设置项

系统的默认设置项可修改的部分是存储在SettingsProvider的，修改的原理是通过settings命令修改SettingsProvider存储的设置项来实现 注： 1.修改设置之后，可能需要重启设备才能生效;2.需要root权限

可以使用 adb shell settings -h 查看 settings 命令的帮助信息：

```
$ settings
usage:  settings [--user <USER_ID> | current] get namespace key
        settings [--user <USER_ID> | current] put namespace key value
        settings [--user <USER_ID> | current] delete namespace key
        settings [--user <USER_ID> | current] list namespace

'namespace' is one of {system, secure, global}, case-insensitive
```

从中可以看出，修改系统默认项，只需要通过命令:

> settings put namespace key value

其中　namespace　的参数为{system, secure, global}的数据表之一.具体需要的namespace和key详见frameworks/base/core/java/android/provider/Settings.java.

下面给出一个简单的范例，修改屏幕亮度，经查Settings.java可知道屏幕亮度的key为screen_brightness，存放在system数据表中:

- 设置屏幕亮度

  > settings put system screen_brightness 102

- 获取屏幕亮度

  > settings get system screen_brightness

  返回结果:

  > 102

修改分辨率

> adb shell wm size 480x1024

表示将分辨率修改为 480px * 1024px。

恢复原分辨率

> adb shell wm size reset

修改屏幕密度

> adb shell wm density 160

表示将屏幕密度修改为 160dpi

恢复原屏幕密度

> adb shell wm density reset

显示区域

> adb shell wm overscan 0,0,0,200 四个数字分别表示距离左、上、右、下边缘的留白像素，以上命令表示将屏幕底部 200px 留白。

恢复显示区域

> adb shell wm overscan reset

关闭USB调试模式

> adb shell settings put global adb_enabled 0

实用功能

屏幕截图

> adb shell screencap -p /sdcard/sc.png

可以使用 adb shell screencap -h 查看 screencap 命令的帮助信息：

```
usage: screencap [-hp] [-d display-id] [FILENAME]
   -h: this message
   -p: save the file as a png.
   -d: specify the display id to capture, default 0.
If FILENAME ends with .png it will be saved as a png.
If FILENAME is not given, the results will be printed to stdout.
```

指定文件名以 .png 结尾时可以省略 -p 参数；否则需要使用 -p 参数。如果不指定文件名，截图文件的内容将直接输出到 stdout。

录制屏幕

录制屏幕以 mp4 格式保存到 /sdcard：

> adb shell screenrecord /sdcard/filename.mp4

需要停止时按 Ctrl-C，默认录制时间和最长录制时间都是 180 秒。

可以使用 adb shell screenrecord --help 查看 screenrecord 命令的帮助信息：

```
Usage: screenrecord [options] <filename>

Android screenrecord v1.2.  Records the device's display to a .mp4 file.

Options:
--size WIDTHxHEIGHT //视频的尺寸，比如 1280x720，默认是屏幕分辨率.
    Set the video size, e.g. "1280x720".  Default is the device's main
    display resolution (if supported), 1280x720 if not.  For best results,
    use a size supported by the AVC encoder.
--bit-rate RATE // 视频的比特率，默认是 4Mbps.
    Set the video bit rate, in bits per second.  Value may be specified as
    bits or megabits, e.g. '4000000' is equivalent to '4M'.  Default 4Mbps.
--bugreport //类似时间戳，在屏幕上显示附加信息
    Add additional information, such as a timestamp overlay, that is helpful
    in videos captured to illustrate bugs.
--time-limit TIME //录制时长，单位秒.
    Set the maximum recording time, in seconds.  Default / maximum is 180.
--verbose  //输出更多信息
    Display interesting information on stdout.
--help
    Show this message.

Recording continues until Ctrl-C is hit or the time limit is reached.
```

重新挂载System分区为可写

重新挂载system分区需要 root 权限.在默认打开adb root的机器上

> adb root adb remount 即可挂载system为可写

或者我们也可以如下方式进行重新挂载

1. 进入 shell 并切换到 root 用户权限

   > adb shell
   >
   > su

2. 查看当前分区挂载情况

   > cat /proc/mounts

   输出示例：

```
rootfs / rootfs ro,seclabel,size=1001040k,nr_inodes=250260 0 0
tmpfs /dev tmpfs rw,seclabel,nosuid,relatime,size=1010504k,nr_inodes=252626,mode=755 0 0
devpts /dev/pts devpts rw,seclabel,relatime,mode=600 0 0
proc /proc proc rw,relatime,gid=3009,hidepid=2 0 0
sysfs /sys sysfs rw,seclabel,relatime 0 0
selinuxfs /sys/fs/selinux selinuxfs rw,relatime 0 0
/sys/kernel/debug /sys/kernel/debug debugfs rw,seclabel,relatime,mode=755 0 0
/sys/kernel/debug/tracing /sys/kernel/debug/tracing tracefs rw,seclabel,relatime,mode=755 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
```

```
none /dev/stune cgroup rw,relatime,schedtune 0 0
tmpfs /mnt tmpfs rw,seclabel,relatime,size=1010504k,nr_inodes=252626,mode=755,gid=1000 0 0
none /config configfs rw,relatime 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
none /dev/cpuset cgroup rw,relatime,cpuset,noprefix,release_agent=/sbin/cpuset_release_agent 0 0
pstore /sys/fs/pstore pstore rw,seclabel,relatime 0 0
/dev/block/platform/fe330000.sdhci/by-name/system /system ext4 ro,seclabel,noatime,nodiratime,noauto_da_alloc,
/dev/block/platform/fe330000.sdhci/by-name/cache /cache ext4 rw,seclabel,nosuid,nodev,noatime,nodiratime,disca
/dev/block/platform/fe330000.sdhci/by-name/metadata /metadata ext4 rw,seclabel,nosuid,nodev,noatime,nodiratime
/dev/block/platform/fe330000.sdhci/by-name/userdata /data f2fs rw,seclabel,nosuid,nodev,noatime,nodiratime,bac
tmpfs /storage tmpfs rw,seclabel,relatime,size=1010504k,nr_inodes=252626,mode=755,gid=1000 0 0
adb /dev/usb-ffs/adb functionfs rw,relatime 0 0
/dev/fuse /mnt/runtime/default/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default
/dev/fuse /storage/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_permissions
/dev/fuse /mnt/runtime/read/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_pe
/dev/fuse /mnt/runtime/write/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_p
```

找到 /system 的那一行：

```
/dev/block/platform/fe330000.sdhci/by-name/system /system ext4 ro,seclabel,noatime,nodiratime,noauto_da_alloc,
```

3. 重新挂载。

> mount -o remount,rw /dev/block/platform/fe330000.sdhci/by-name/system /system

这里的　/dev/block/platform/fe330000.sdhci/by-name/system 就是我们从上一步的输出里得到的文件路径.

其中第 2 步也可以用mount命令，但是在firefly平台上这两个命令输出的结果是不同的.使用mount中获取的system(/dev/block/mmcblk1p10)地址是无法成功重新挂载的

直接运行ROOT SHELL

当我们需要用到su权限运行shell的时候，常用的步骤为:

> adb shell
>
> su
>
> cat /proc/mounts

这样的话需要输入三次命令 实际上，我们使用su -h，可以看到

```
k3399_firefly_box:/ $ su -c
su: option requires an argument -- c

Usage: su [options] [--] [-] [LOGIN] [--] [args...]

Options:
  --daemon                      start the su daemon agent
  -c, --command COMMAND         pass COMMAND to the invoked shell
```

```
  -h, --help                  display this help message and exit
  -, -l, --login              pretend the shell to be a login shell
  -m, -p,
  --preserve-environment      do not change environment variables
  -s, --shell SHELL           use SHELL instead of the default /system/bin/sh
  -u                          display the multiuser mode and exit
  -v, --version               display version number and exit
  -V                          display version code and exit,
```

su支持　su -c command来直接运行命令，所以上面cat /proc/mounts的命令可以简化为:

> adb shell su -c "cat /proc/mounts"

查看连接过的WIFI密码

注：需要 root 权限。

> adb shell su -c "cat /data/misc/wifi/*.conf"

输出示例：

```
network={
    ssid="ASUS"
    psk="1234567890"
    key_mgmt=WPA-PSK
    disabled=1
    id_str="%7B%22creatorUid%22%3A%221000%22%2C%22configKey%22%3A%22%5C%22ASUS%5C%22WPA_PSK%22%7D"
}
```

ssid 即为我们在 WLAN 设置里看到的名称，psk 为密码，key_mgmt 为安全加密方式

设置系统日期和时间

注：需要 root 权限。

> adb shell su -c "date -s 20160823.131500"

表示将系统日期和时间更改为 2016年08月23日13点15分00秒.

刷机相关命令

重启到Recovery模式

> adb reboot recovery

从Recovery重启到Android

```
adb reboot
```

重启到Fastboot模式

```
adb reboot bootloader
```

使用adb升级ota包或固件包

升级ＯＴＡ包update.zip

1 .使用adb push　ota包update.zip到/sdcard/目录

```
adb push update.zip /sdcard/update.zip
```

2 .写如下字段到/cache/recovery/last_flag

```
updating$path=/mnt/internal_sd/update.zip
```

3 .写如下字段到/cache/recovery/command

```
--update_package=/mnt/internal_sd/update.zip
--locale=en_US
```

4.重启升级

```
adb shell reboot recovery
```

升级固件包update.img(rk方案)

1 .使用adb push　ota包update.zip或固件包update.img到/sdcard/目录

```
adb push update.zip /sdcard/update.zip
```

2 .写如下字段到/cache/recovery/last_flag

```
updating$path=/mnt/internal_sd/update.img
```

3 .写如下字段到/cache/recovery/command --update_rkimage=/mnt/internal_sd/update.img --locale=en_US

4.重启升级

```
adb shell reboot recovery
```

安全相关命令

启用/禁用 SELinux

启用 SELinux

> adb root
>
> adb shell setenforce 1

禁用 SELinux

> adb root
>
> adb shell setenforce 0

启用/禁用 dm_verity

启用 dm_verity

> adb root
>
> adb enable-verity

禁用 dm_verity

> adb root
>
> adb disable-verity

系统svc命令

系统svc命令，位置在/system/bin目录下，用来管理电源控制，无线数据，WIFI，USB,NFC

```
svc -h
Available commands:
    help     Show information about the subcommands
    power    Control the power manager
    data     Control mobile data connectivity
    wifi     Control the Wi-Fi manager
    usb      Control Usb state
    nfc      Control NFC functions
```

注:svc的相关命令都需要root权限

首先使用adb root切换到root模式，或者使用adb shell su -c command来运行命令

电源控制

可以使用 adb shell svc power -h 查看管理电源控制的帮助信息，下面是常见参数及含义：

```
$ svc power
Control the power manager
usage: svc power stayon [true|false|usb|ac|wireless]
```

```
          Set the 'keep awake while plugged in' setting.
        svc power reboot [reason]
          Perform a runtime shutdown and reboot device with specified reason.
        svc power shutdown
          Perform a runtime shutdown and power off the device.
```

1. 屏幕相关:

   > svc power stayon [true|false|usb|ac]

   设置屏幕的常亮，true保持常亮，false不保持，usb当插入usb时常亮，ac当插入电源时常亮

2. 关机:

   > svc power shutdown

3. 重启

   > svc power reboot

数据连接

可以使用 adb shell svc data -h 查看管理电源控制的帮助信息，下面是常见参数及含义：

```
$ svc data
Control mobile data connectivity

usage: svc data [enable|disable]
         Turn mobile data on or off.
```

1. 关闭数据连接:

   > svc data enable

2. 启用数据连接:

   > svc data disable

WIFI

可以使用 adb shell svc wifi -h 查看管理电源控制的帮助信息，下面是常见参数及含义：

```
$ svc wifi
Control the Wi-Fi manager

usage: svc wifi [enable|disable]
         Turn Wi-Fi on or off.
```

1. 关闭WIFI:

   > svc wifi enable

2. 打开WIFI:

   > svc wifi disable

USB

可以使用 adb shell svc usb -h 查看管理电源控制的帮助信息，下面是常见参数及含义：

```
$ svc usb
Control Usb state

usage: svc usb setFunction [function]
         Set the current usb function.

       svc usb getFunction
          Gets the list of currently enabled functions
```

1. 设置USB连接方式:

   > svc usb setFunction mtp,adb

2. 获取USB连接方式:

   > svc usb getFunction

   返回结果：

   > mtp,adb

NFC

可以使用 adb shell svc nfc -h 查看管理电源控制的帮助信息，下面是常见参数及含义：

```
$ svc nfc
Control NFC functions

usage: svc nfc [enable|disable]
         Turn NFC on or off.
```

1. 关闭NFC:

   > svc nfc enable

2. 打开NFC:

> svc nfc disable