scrcpy (v1.23)

scrcpy

*pronounced "**scr**een **copy**"*

[Read in another language]

This application provides display and control of Android devices connected via USB (or [over TCP/IP]). It does not require any *root* access. It works on *GNU/Linux*, *Windows* and *macOS*.

screenshot

It focuses on:

- **lightness**: native, displays only the device screen
- **performance**: 30~120fps, depending on the device
- **quality**: 1920×1080 or above
- **low latency**: [35~70ms]
- **low startup time**: ~1 second to display the first image
- **non-intrusiveness**: nothing is left installed on the device
- **user benefits**: no account, no ads, no internet required
- **freedom**: free and open source software

Its features include:

- [recording]
- mirroring with [device screen off]
- [copy-paste] in both directions
- [configurable quality]
- device screen [as a webcam (V4L2)] (Linux-only)
- [physical keyboard simulation (HID)]
- [physical mouse simulation (HID)]
- [OTG mode]
- and more…

Requirements

The Android device requires at least API 21 (Android 5.0).

Make sure you [enabled adb debugging] on your device(s).

On some devices, you also need to enable [an additional option] to control it using keyboard and mouse.

Get the app

Summary

- Linux: `apt install scrcpy`
- Windows: [download]
- macOS: `brew install scrcpy`

Build from sources: [BUILD] ([simplified process])

Linux

On Debian and Ubuntu:

```
apt install scrcpy
```

On Arch Linux:

```
pacman -S scrcpy
```

A Snap package is available: `scrcpy` .

For Fedora, a COPR package is available: `scrcpy` .

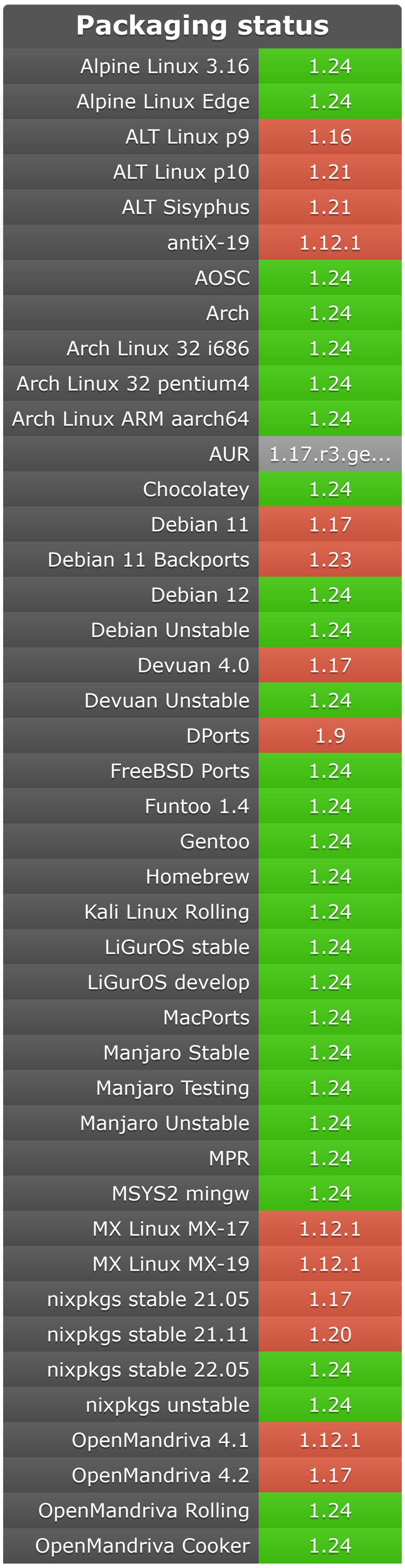
For Gentoo, an Ebuild is available: `scrcpy/` .

You could also build the app manually (simplified process).

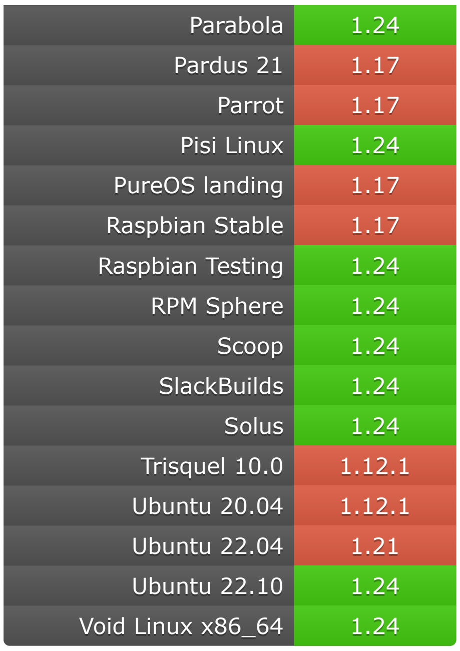
Windows

For Windows, for simplicity, a prebuilt archive with all the dependencies (including `adb`) is available:

- `scrcpy-win64-v1.23.zip`
  (SHA-256:



Packaging status	
Alpine Linux 3.16	1.24
Alpine Linux Edge	1.24
ALT Linux p9	1.16
ALT Linux p10	1.21
ALT Sisyphus	1.21
antiX-19	1.12.1
AOSC	1.24
Arch	1.24
Arch Linux 32 i686	1.24
Arch Linux 32 pentium4	1.24
Arch Linux ARM aarch64	1.24
AUR	1.17.r3.ge...
Chocolatey	1.24
Debian 11	1.17
Debian 11 Backports	1.23
Debian 12	1.24
Debian Unstable	1.24
Devuan 4.0	1.17
Devuan Unstable	1.24
DPorts	1.9
FreeBSD Ports	1.24
Funtoo 1.4	1.24
Gentoo	1.24
Homebrew	1.24
Kali Linux Rolling	1.24
LiGurOS stable	1.24
LiGurOS develop	1.24
MacPorts	1.24
Manjaro Stable	1.24
Manjaro Testing	1.24
Manjaro Unstable	1.24
MPR	1.24
MSYS2 mingw	1.24
MX Linux MX-17	1.12.1
MX Linux MX-19	1.12.1
nixpkgs stable 21.05	1.17
nixpkgs stable 21.11	1.20
nixpkgs stable 22.05	1.24
nixpkgs unstable	1.24
OpenMandriva 4.1	1.12.1
OpenMandriva 4.2	1.17
OpenMandriva Rolling	1.24
OpenMandriva Cooker	1.24



Parabola	1.24
Pardus 21	1.17
Parrot	1.17
Pisi Linux	1.24
PureOS landing	1.17
Raspbian Stable	1.17
Raspbian Testing	1.24
RPM Sphere	1.24
Scoop	1.24
SlackBuilds	1.24
Solus	1.24
Trisquel 10.0	1.12.1
Ubuntu 20.04	1.12.1
Ubuntu 22.04	1.21
Ubuntu 22.10	1.24
Void Linux x86_64	1.24

d2f601b1d0157faf65153d8a093d827fd65aec5d5842d677ac86fb2b5b7704cc)

It is also available in Chocolatey:

```
choco install scrcpy
choco install adb    # if you don't have it yet
```

And in Scoop:

```
scoop install scrcpy
scoop install adb    # if you don't have it yet
```

You can also build the app manually.

macOS

The application is available in Homebrew. Just install it:

```
brew install scrcpy
```

You need `adb`, accessible from your `PATH`. If you don't have it yet:

```
brew install android-platform-tools
```

It's also available in MacPorts, which sets up adb for you:

```
sudo port install scrcpy
```

You can also build the app manually.

Run

Plug an Android device, and execute:

```
scrcpy
```

It accepts command-line arguments, listed by:

```
scrcpy --help
```

Features

Capture configuration

Reduce size

Sometimes, it is useful to mirror an Android device at a lower definition to increase performance.

To limit both the width and height to some value (e.g. 1024):

```
scrcpy --max-size 1024
scrcpy -m 1024  # short version
```

The other dimension is computed to that the device aspect ratio is preserved. That way, a device in 1920×1080 will be mirrored at 1024×576.

Change bit-rate

The default bit-rate is 8 Mbps. To change the video bitrate (e.g. to 2 Mbps):

```
scrcpy --bit-rate 2M
scrcpy -b 2M  # short version
```

Limit frame rate

The capture frame rate can be limited:

```
scrcpy --max-fps 15
```

This is officially supported since Android 10, but may work on earlier versions.

The actual capture framerate may be printed to the console:

```
scrcpy --print-fps
```

It may also be enabled or disabled at any time with `MOD`+`i`.

Crop

The device screen may be cropped to mirror only part of the screen.

This is useful for example to mirror only one eye of the Oculus Go:

```
scrcpy --crop 1224:1440:0:0   # 1224x1440 at offset (0,0)
```

If `--max-size` is also specified, resizing is applied after cropping.

Lock video orientation

To lock the orientation of the mirroring:

```
scrcpy --lock-video-orientation     # initial (current) orientation
scrcpy --lock-video-orientation=0   # natural orientation
scrcpy --lock-video-orientation=1   # 90° counterclockwise
scrcpy --lock-video-orientation=2   # 180°
scrcpy --lock-video-orientation=3   # 90° clockwise
```

This affects recording orientation.

The window may also be rotated independently.

Encoder

Some devices have more than one encoder, and some of them may cause issues or crash. It is possible to select a different encoder:

```
scrcpy --encoder OMX.qcom.video.encoder.avc
```

To list the available encoders, you could pass an invalid encoder name, the error will give the available encoders:

```
scrcpy --encoder _
```

Capture

Recording

It is possible to record the screen while mirroring:

```
scrcpy --record file.mp4
scrcpy -r file.mkv
```

To disable mirroring while recording:

```
scrcpy --no-display --record file.mp4
scrcpy -Nr file.mkv
# interrupt recording with Ctrl+C
```

"Skipped frames" are recorded, even if they are not displayed in real time (for performance reasons). Frames are *timestamped* on the device, so packet delay variation does not impact the recorded file.

v4l2loopback

On Linux, it is possible to send the video stream to a v4l2 loopback device, so that the Android device can be opened like a webcam by any v4l2-capable tool.

The module `v4l2loopback` must be installed:

```
sudo apt install v4l2loopback-dkms
```

To create a v4l2 device:

```
sudo modprobe v4l2loopback
```

This will create a new video device in `/dev/videoN`, where `N` is an integer (more options are available to create several devices or devices with specific IDs).

To list the enabled devices:

```
# requires v4l-utils package
v4l2-ctl --list-devices

# simple but might be sufficient
ls /dev/video*
```

To start scrcpy using a v4l2 sink:

```
scrcpy --v4l2-sink=/dev/videoN
scrcpy --v4l2-sink=/dev/videoN --no-display  # disable mirroring window
scrcpy --v4l2-sink=/dev/videoN -N            # short version
```

(replace `N` by the device ID, check with `ls /dev/video*`)

Once enabled, you can open your video stream with a v4l2-capable tool:

```
ffplay -i /dev/videoN
vlc v4l2:///dev/videoN   # VLC might add some buffering delay
```

For example, you could capture the video within OBS.

Buffering

It is possible to add buffering. This increases latency but reduces jitter (see #2464).

The option is available for display buffering:

```
scrcpy --display-buffer=50  # add 50 ms buffering for display
```

and V4L2 sink:

```
scrcpy --v4l2-buffer=500    # add 500 ms buffering for v4l2 sink
```

Connection

TCP/IP (wireless)

Scrcpy uses `adb` to communicate with the device, and `adb` can connect to a device over TCP/IP. The device must be connected on the same network as the computer.

Automatic

An option `--tcpip` allows to configure the connection automatically. There are two variants.

If the device (accessible at 192.168.1.1 in this example) already listens on a port (typically 5555) for incoming adb connections, then run:

```
scrcpy --tcpip=192.168.1.1       # default port is 5555
scrcpy --tcpip=192.168.1.1:5555
```

If adb TCP/IP mode is disabled on the device (or if you don't know the IP address), connect the device over USB, then run:

```
scrcpy --tcpip    # without arguments
```

It will automatically find the device IP address, enable TCP/IP mode, then connect to the device before starting.

Manual

Alternatively, it is possible to enable the TCP/IP connection manually using `adb`:

1. Plug the device into a USB port on your computer.
2. Connect the device to the same Wi-Fi network as your computer.
3. Get your device IP address, in Settings → About phone → Status, or by executing this command:

   ```
   adb shell ip route | awk '{print $9}'
   ```

4. Enable adb over TCP/IP on your device: `adb tcpip 5555`.
5. Unplug your device.
6. Connect to your device: `adb connect DEVICE_IP:5555` *(replace `DEVICE_IP` with the device IP address you found)*.
7. Run `scrcpy` as usual.

Since Android 11, a Wireless debugging option allows to bypass having to physically connect your device directly to your computer.

If the connection randomly drops, run your `scrcpy` command to reconnect. If it says there are no devices/emulators found, try running `adb connect DEVICE_IP:5555` again, and then `scrcpy` as usual. If it still says there are none found, try running `adb disconnect` and then run those two commands again.

It may be useful to decrease the bit-rate and the definition:

```
scrcpy --bit-rate 2M --max-size 800
scrcpy -b2M -m800  # short version
```

Multi-devices

If several devices are listed in `adb devices`, you can specify the *serial*:

```
scrcpy --serial 0123456789abcdef
scrcpy -s 0123456789abcdef  # short version
```

If the device is connected over TCP/IP:

```
scrcpy --serial 192.168.0.1:5555
scrcpy -s 192.168.0.1:5555  # short version
```

If only one device is connected via either USB or TCP/IP, it is possible to select it automatically:

```
# Select the only device connected via USB
scrcpy -d             # like adb -d
scrcpy --select-usb   # long version

# Select the only device connected via TCP/IP
scrcpy -e             # like adb -e
scrcpy --select-tcpip # long version
```

You can start several instances of *scrcpy* for several devices.

Autostart on device connection

You could use AutoAdb:

```
autoadb scrcpy -s '{}'
```

Tunnels

To connect to a remote device, it is possible to connect a local `adb` client to a remote `adb` server (provided they use the same version of the *adb* protocol).

Remote ADB server

To connect to a remote ADB server, make the server listen on all interfaces:

```
adb kill-server
adb -a nodaemon server start
# keep this open
```

Warning: all communications between clients and ADB server are unencrypted.

Suppose that this server is accessible at 192.168.1.2. Then, from another terminal, run scrcpy:

```
export ADB_SERVER_SOCKET=tcp:192.168.1.2:5037
scrcpy --tunnel-host=192.168.1.2
```

By default, scrcpy uses the local port used for `adb forward` tunnel establishment (typically `27183`, see `--port`). It is also possible to force a different tunnel port (it may be useful in more complex situations, when more redirections are involved):

```
scrcpy --tunnel-port=1234
```

SSH tunnel

To communicate with a remote ADB server securely, it is preferable to use a SSH tunnel.

First, make sure the ADB server is running on the remote computer:

```
adb start-server
```

Then, establish a SSH tunnel:

```
# local  5038 --> remote  5037
# local 27183 <-- remote 27183
ssh -CN -L5038:localhost:5037 -R27183:localhost:27183 your_remote_computer
# keep this open
```

From another terminal, run scrcpy:

```
export ADB_SERVER_SOCKET=tcp:localhost:5038
scrcpy
```

To avoid enabling remote port forwarding, you could force a forward connection instead (notice the `-L` instead of `-R`):

```
# local  5038 --> remote  5037
# local 27183 --> remote 27183
ssh -CN -L5038:localhost:5037 -L27183:localhost:27183 your_remote_computer
# keep this open
```

From another terminal, run scrcpy:

```
export ADB_SERVER_SOCKET=tcp:localhost:5038
scrcpy --force-adb-forward
```

Like for wireless connections, it may be useful to reduce quality:

```
scrcpy -b2M -m800 --max-fps 15
```

Window configuration

Title

By default, the window title is the device model. It can be changed:

```
scrcpy --window-title 'My device'
```

Position and size

The initial window position and size may be specified:

```
scrcpy --window-x 100 --window-y 100 --window-width 800 --window-height 600
```

Borderless

To disable window decorations:

```
scrcpy --window-borderless
```

Always on top

To keep the scrcpy window always on top:

```
scrcpy --always-on-top
```

Fullscreen

The app may be started directly in fullscreen:

```
scrcpy --fullscreen
scrcpy -f  # short version
```

Fullscreen can then be toggled dynamically with `MOD+f`.

Rotation

The window may be rotated:

```
scrcpy --rotation 1
```

Possibles values are:

- `0` : no rotation
- `1` : 90 degrees counterclockwise
- `2` : 180 degrees
- `3` : 90 degrees clockwise

The rotation can also be changed dynamically with `MOD+←` *(left)* and `MOD+→` *(right)*.

Note that *scrcpy* manages 3 different rotations:

- `MOD+r` requests the device to switch between portrait and landscape (the current running app may refuse, if it does not support the requested orientation).
- `--lock-video-orientation` changes the mirroring orientation (the orientation of the video sent from the device to the computer). This affects the recording.
- `--rotation` (or `MOD+←`/`MOD+→`) rotates only the window content. This affects only the display, not the recording.

Other mirroring options

Read-only

To disable controls (everything which can interact with the device: input keys, mouse events, drag&drop files):

```
scrcpy --no-control
scrcpy -n
```

Display

If several displays are available, it is possible to select the display to mirror:

```
scrcpy --display 1
```

The list of display ids can be retrieved by:

```
adb shell dumpsys display   # search "mDisplayId=" in the output
```

The secondary display may only be controlled if the device runs at least Android 10 (otherwise it is mirrored in read-only).

Stay awake

To prevent the device to sleep after some delay when the device is plugged in:

```
scrcpy --stay-awake
scrcpy -w
```

The initial state is restored when scrcpy is closed.

Turn screen off

It is possible to turn the device screen off while mirroring on start with a command-line option:

```
scrcpy --turn-screen-off
scrcpy -S
```

Or by pressing `MOD`+`o` at any time.

To turn it back on, press `MOD`+`Shift`+`o`.

On Android, the `POWER` button always turns the screen on. For convenience, if `POWER` is sent via scrcpy (via right-click or `MOD`+`p`), it will force to turn the screen off after a small delay (on a best effort basis). The physical `POWER` button will still cause the screen to be turned on.

It can also be useful to prevent the device from sleeping:

```
scrcpy --turn-screen-off --stay-awake
scrcpy -Sw
```

Power off on close

To turn the device screen off when closing scrcpy:

```
scrcpy --power-off-on-close
```

Show touches

For presentations, it may be useful to show physical touches (on the physical device).

Android provides this feature in *Developers options*.

Scrcpy provides an option to enable this feature on start and restore the initial value on exit:

```
scrcpy --show-touches
scrcpy -t
```

Note that it only shows *physical* touches (with the finger on the device).

Disable screensaver

By default, scrcpy does not prevent the screensaver to run on the computer.

To disable it:

```
scrcpy --disable-screensaver
```

Input control

Rotate device screen

Press `MOD`+`r` to switch between portrait and landscape modes.

Note that it rotates only if the application in foreground supports the requested orientation.

Copy-paste

Any time the Android clipboard changes, it is automatically synchronized to the computer clipboard.

Any `Ctrl` shortcut is forwarded to the device. In particular:

- `Ctrl`+`c` typically copies
- `Ctrl`+`x` typically cuts
- `Ctrl`+`v` typically pastes (after computer-to-device clipboard synchronization)

This typically works as you expect.

The actual behavior depends on the active application though. For example, *Termux* sends SIGINT on `Ctrl`+`c` instead, and *K-9 Mail* composes a new message.

To copy, cut and paste in such cases (but only supported on Android >= 7):

- `MOD`+`c` injects `COPY`
- `MOD`+`x` injects `CUT`
- `MOD`+`v` injects `PASTE` (after computer-to-device clipboard synchronization)

In addition, `MOD`+`Shift`+`v` allows to inject the computer clipboard text as a sequence of key events. This is useful when the component does not accept text pasting (for example in *Termux*), but it can break non-ASCII content.

WARNING: Pasting the computer clipboard to the device (either via `Ctrl`+`v` or `MOD`+`v`) copies the content into the device clipboard. As a consequence, any Android application could read its content. You should avoid to paste sensitive content (like passwords) that way.

Some devices do not behave as expected when setting the device clipboard programmatically. An option `--legacy-paste` is provided to change the behavior of `Ctrl`+`v` and `MOD`+`v` so that they also inject the computer clipboard text as a sequence of key events (the same way as `MOD`+`Shift`+`v`).

To disable automatic clipboard synchronization, use `--no-clipboard-autosync`.

Pinch-to-zoom

To simulate "pinch-to-zoom": `Ctrl`+*click-and-move*.

More precisely, hold `Ctrl` while pressing the left-click button. Until the left-click button is released, all mouse movements scale and rotate the content (if supported by the app) relative to the center of the screen.

Concretely, scrcpy generates additional touch events from a "virtual finger" at a location inverted through the center of the screen.

Physical keyboard simulation (HID)

By default, scrcpy uses Android key or text injection: it works everywhere, but is limited to ASCII.

Alternatively, scrcpy can simulate a physical USB keyboard on Android to provide a better input experience (using USB HID over AOAv2): the virtual keyboard is disabled and it works for all characters and IME.

However, it only works if the device is connected by USB.

Note: On Windows, it may only work in OTG mode, not while mirroring (it is not possible to open a USB device if it is already open by another process like the adb daemon).

To enable this mode:

```
scrcpy --hid-keyboard
scrcpy -K  # short version
```

If it fails for some reason (for example because the device is not connected via USB), it automatically fallbacks to the default mode (with a log in the console). This allows to use the same command line options when connected over USB and TCP/IP.

In this mode, raw key events (scancodes) are sent to the device, independently of the host key mapping. Therefore, if your keyboard layout does not match, it must be configured on the Android device, in Settings → System → Languages and input → Physical keyboard.

This settings page can be started directly:

```
adb shell am start -a android.settings.HARD_KEYBOARD_SETTINGS
```

However, the option is only available when the HID keyboard is enabled (or when a physical keyboard is connected).

Physical mouse simulation (HID)

Similarly to the physical keyboard simulation, it is possible to simulate a physical mouse. Likewise, it only works if the device is connected by USB.

By default, scrcpy uses Android mouse events injection, using absolute coordinates. By simulating a physical mouse, a mouse pointer appears on the Android device, and relative mouse motion, clicks and scrolls are injected.

To enable this mode:

```
scrcpy --hid-mouse
scrcpy -M  # short version
```

You could also add `--forward-all-clicks` to forward all mouse buttons.

When this mode is enabled, the computer mouse is "captured" (the mouse pointer disappears from the computer and appears on the Android device instead).

Special capture keys, either `Alt` or `Super`, toggle (disable or enable) the mouse capture. Use one of them to give the control of the mouse back to the computer.

OTG

It is possible to run *scrcpy* with only physical keyboard and mouse simulation (HID), as if the computer keyboard and mouse were plugged directly to the device via an OTG cable.

In this mode, *adb* (USB debugging) is not necessary, and mirroring is disabled.

To enable OTG mode:

```
scrcpy --otg
# Pass the serial if several USB devices are available
scrcpy --otg -s 0123456789abcdef
```

It is possible to enable only HID keyboard or HID mouse:

```
scrcpy --otg --hid-keyboard              # keyboard only
scrcpy --otg --hid-mouse                 # mouse only
scrcpy --otg --hid-keyboard --hid-mouse  # keyboard and mouse
# for convenience, enable both by default
scrcpy --otg                             # keyboard and mouse
```

Like `--hid-keyboard` and `--hid-mouse`, it only works if the device is connected by USB.

Text injection preference

There are two kinds of events generated when typing text:

- *key events*, signaling that a key is pressed or released;
- *text events*, signaling that a text has been entered.

By default, letters are injected using key events, so that the keyboard behaves as expected in games (typically for WASD keys).

But this may cause issues. If you encounter such a problem, you can avoid it by:

```
scrcpy --prefer-text
```

(but this will break keyboard behavior in games)

On the contrary, you could force to always inject raw key events:

```
scrcpy --raw-key-events
```

These options have no effect on HID keyboard (all key events are sent as scancodes in this mode).

Key repeat

By default, holding a key down generates repeated key events. This can cause performance problems in some games, where these events are useless anyway.

To avoid forwarding repeated key events:

```
scrcpy --no-key-repeat
```

This option has no effect on HID keyboard (key repeat is handled by Android directly in this mode).

Right-click and middle-click

By default, right-click triggers BACK (or POWER on) and middle-click triggers HOME. To disable these shortcuts and forward the clicks to the device instead:

```
scrcpy --forward-all-clicks
```

File drop

Install APK

To install an APK, drag & drop an APK file (ending with `.apk`) to the *scrcpy* window.

There is no visual feedback, a log is printed to the console.

Push file to device

To push a file to `/sdcard/Download/` on the device, drag & drop a (non-APK) file to the *scrcpy* window.

There is no visual feedback, a log is printed to the console.

The target directory can be changed on start:

```
scrcpy --push-target=/sdcard/Movies/
```

Audio forwarding

Audio is not forwarded by *scrcpy*. Use sndcpy.

Also see issue #14.

Shortcuts

In the following list, `MOD` is the shortcut modifier. By default, it's (left) `Alt` or (left) `Super`.

It can be changed using `--shortcut-mod`. Possible keys are `lctrl`, `rctrl`, `lalt`, `ralt`, `lsuper` and `rsuper`. For example:

```
# use RCtrl for shortcuts
scrcpy --shortcut-mod=rctrl

# use either LCtrl+LAlt or LSuper for shortcuts
scrcpy --shortcut-mod=lctrl+lalt,lsuper
```

`Super` is typically the `Windows` or `Cmd` key.

Action	Shortcut
Switch fullscreen mode	`MOD`+`f`
Rotate display left	`MOD`+`←` *(left)*
Rotate display right	`MOD`+`→` *(right)*
Resize window to 1:1 (pixel-perfect)	`MOD`+`g`
Resize window to remove black borders	`MOD`+`w` | *Double-left-click*[1]
Click on `HOME`	`MOD`+`h` | *Middle-click*
Click on `BACK`	`MOD`+`b` | *Right-click*[2]
Click on `APP_SWITCH`	`MOD`+`s` | *4th-click*[3]
Click on `MENU` (unlock screen)[4]	`MOD`+`m`
Click on `VOLUME_UP`	`MOD`+`↑` *(up)*
Click on `VOLUME_DOWN`	`MOD`+`↓` *(down)*
Click on `POWER`	`MOD`+`p`
Power on	*Right-click*[2]
Turn device screen off (keep mirroring)	`MOD`+`o`
Turn device screen on	`MOD`+`Shift`+`o`
Rotate device screen	`MOD`+`r`
Expand notification panel	`MOD`+`n` | *5th-click*[3]
Expand settings panel	`MOD`+`n`+`n` | *Double-5th-click*[3]
Collapse panels	`MOD`+`Shift`+`n`
Copy to clipboard[5]	`MOD`+`c`

Cut to clipboard[5]	`MOD`+`x`
Synchronize clipboards and paste[5]	`MOD`+`v`
Inject computer clipboard text	`MOD`+`Shift`+`v`
Enable/disable FPS counter (on stdout)	`MOD`+`i`
Pinch-to-zoom	`Ctrl`+*click-and-move*
Drag & drop APK file	Install APK from computer
Drag & drop non-APK file	Push file to device

[1]Double-click on black borders to remove them.
[2]Right-click turns the screen on if it was off, presses BACK otherwise.
[3]4th and 5th mouse buttons, if your mouse has them.
[4]For react-native apps in development, `MENU` triggers development menu.
[5]Only on Android >= 7.

Shortcuts with repeated keys are executted by releasing and pressing the key a second time. For example, to execute "Expand settings panel":

1. Press and keep pressing `MOD`.
2. Then double-press `n`.
3. Finally, release `MOD`.

All `Ctrl`+*key* shortcuts are forwarded to the device, so they are handled by the active application.

Custom paths

To use a specific *adb* binary, configure its path in the environment variable `ADB`:

```
ADB=/path/to/adb scrcpy
```

To override the path of the `scrcpy-server` file, configure its path in `SCRCPY_SERVER_PATH`.

To override the icon, configure its path in `SCRCPY_ICON_PATH`.

Why *scrcpy*?

A colleague challenged me to find a name as unpronounceable as gnirehtet.

`strcpy` copies a **str**ing; `scrcpy` copies a **scr**een.

How to build?

See BUILD.

Common issues

See the FAQ.md).

Developers

Read the developers page.

Licence

Articles

- Introducing scrcpy
- Scrcpy now works wirelessly

Contact

If you encounter a bug, please read the FAQ first, then open an issue.

For general questions or discussions, you could also use:

- Reddit: `r/scrcpy`
- Twitter: `@scrcpy_app`

Translations

This README is available in other languages:

- Deutsch (German, `de`) - v1.22
- Indonesian (Indonesia, `id`) - v1.16
- Italiano (Italiano, `it`) - v1.19
- 日本語 (Japanese, `jp`) - v1.19
- 한국어 (Korean, `ko`) - v1.11
- Português Brasileiro (Brazilian Portuguese, `pt-BR`) - v1.19
- Español (Spanish, `sp`) - v1.21
- 简体中文 (Simplified Chinese, `zh-Hans`) - v1.22
- 繁體中文 (Traditional Chinese, `zh-Hant`) - v1.15
- Turkish (Turkish, `tr`) - v1.18

Only this README file is guaranteed to be up-to-date.