Linux on Chromebooks

Booting Persistent USB

It is possible to boot into a 3.1 USB stick with persistent data saved between sessions. There are plenty of cheap options out there for ultra portable USBs that you'll hardly notice due to their low-profiles. Even better, if you have a chromebook with USB C ports. The main limiter on your system will often be read/write speed, as you are funnelling all of your data through a USB device opposed to internal storage. Be careful to choose the port on your device with the best speed, you will be glad you did later on.

If you boot this way, you won't have to run or use crouton, or even boot into ChromeOS (CrOS). You'll need to enable developer mode, and press `CTRL+L` when rebooting the chromebook and the warning for 'OS Detection' being disabled. This is ok, and a needed result of turning on developer mode. To boot into CrOS instead, press `CTRL+D`. If you press nothing, a loud BEEP will happen and it will boot into CrOS.

If you have a windows machine handy, check out the links below to see how you can create a persistent USB for booting. This method does not install linux onto the USB, but rather creates a Live USB (Installation media) of your selected distribution. On this Live USB, if created following these directions, there exists a persistent filesystem, which allows you to retain settings and application data between reboots.

Windows ISO Writer

Normally, on a Live USB there is no persistence and all data will be lost between reboots, so be careful to follow the steps carefully when creating your USB.

Consider how much you plan to store on your system, for me 30GB storage is plenty to do all my programming and server administration from a Lubuntu installation.

My Toshiba 2 Chromebook is running on a massive 2GB of RAM, paired with a generation 6 Intel Duo ~2GhZ and 16GB internal storage. It ran me $100 in 2015 used off ebay and is still running strong. I run Lubuntu booting into a 3.1 USB, which uses i3wm and the bare minimum for packages installed / running. I have no issues in VS Code, Pycharm, LaTeX editors, and all office applications work fine. The main issue to note is web browsing. Chrome or any derivitive will consume nearly all of your RAM. Firefox does ok, and if you visit `about:memory` in your address bar it will allow you to 'Minimize Memory Usage' by clicking a button. Midori is my preferred browser when not dealing with personal accounts and just reading documentation.

Once you have the USB stick made, you can start your chromebook and open a terminal. For me,

this is `CTRL+ALT+t`. Once in the console window, you'll see a `crosh>` prompt. Type the commands below and read the output.

```
Welcome to crosh, the Chrome OS developer shell.

If you got here by mistake, don't panic!  Just close this tab and carry on.

Type 'help' for a list of commands.

If you want to customize the look/behavior, you can use the options page.
Load it by using the Ctrl-Shift-P keyboard shortcut.

crosh> shell
chronos@localhost / $ sudo crossystem
Password:
arch                    = x86                           # [RO/str] Platform architecture
backup_nvram_request    = 1                             # [RW/int] Backup the nvram
somewhere at the next boot.  Cleared on success.
battery_cutoff_request  = 0                             # [RW/int] Cut off battery and
shutdown on next boot
block_devmode           = 0                             # [RW/int] Block all use of
developer mode
clear_tpm_owner_done    = 0                             # [RW/int] Clear TPM owner done
clear_tpm_owner_request = 0                             # [RW/int] Clear TPM owner on next
boot
cros_debug              = 1                             # [RO/int] OS should allow debug
features
dbg_reset               = 0                             # [RW/int] Debug reset mode request
debug_build             = 0                             # [RO/int] OS image built for debug
features
dev_boot_altfw          = 0                             # [RW/int] Enable developer mode
alternate bootloader
dev_boot_signed_only    = 0                             # [RW/int] Enable developer mode
boot only from official kernels
dev_boot_usb            = 0                             # [RW/int] Enable developer mode
boot from external disk (USB/SD)
dev_default_boot        = disk                          # [RW/str] Default boot from disk,
altfw or usb
dev_enable_udc          = 0                             # [RW/int] Enable USB Device
Controller
```

```
devsw_boot              = 1                             # [RO/int] Developer switch position
at boot
devsw_cur               = 1                             # [RO/int] Developer switch current
position
diagnostic_request      = 0                             # [RW/int] Request diagnostic rom
run on next boot
disable_dev_request     = 0                             # [RW/int] Disable virtual dev-mode
on next boot
ecfw_act                = RW                            # [RO/str] Active EC firmware
post_ec_sync_delay      = 0                             # [RW/int] Short delay after EC
software sync (persistent, writable, eve only)
fw_prev_result          = unknown                       # [RO/str] Firmware result of
previous boot (vboot2)
fw_prev_tried           = A                             # [RO/str] Firmware tried on
previous boot (vboot2)
fw_result               = unknown                       # [RW/str] Firmware result this boot
(vboot2)
fw_tried                = A                             # [RO/str] Firmware tried this boot
(vboot2)
fw_try_count            = 0                             # [RW/int] Number of times to try
fw_try_next
fw_try_next             = A                             # [RW/str] Firmware to try next
(vboot2)
fw_vboot2               = 0                             # [RO/int] 1 if firmware was
selected by vboot2 or 0 otherwise
fwb_tries               = 0                             # [RW/int] Try firmware B count
fwid                    = Google_Swanky.5216.238.150    # [RO/str] Active firmware ID
fwupdate_tries          = 0                             # [RW/int] Times to try OS firmware
update (inside kern_nv)
hwid                    = SWANKY E5A-E3P-A47            # [RO/str] Hardware ID
inside_vm               = 0                             # [RO/int] Running in a VM?
kern_nv                 = 0x0000                        # [RO/int] Non-volatile field for
kernel use
kernel_max_rollforward  = 0x00000000                    # [RW/int] Max kernel version to
store into TPM
kernkey_vfy             = sig                           # [RO/str] Type of verification done
on kernel keyblock
loc_idx                 = 0                             # [RW/int] Localization index for
firmware screens
mainfw_act              = A                             # [RO/str] Active main firmware
```

```
mainfw_type             = developer                     # [RO/str] Active main firmware type
nvram_cleared           = 0                             # [RW/int] Have NV settings been
lost?  Write 0 to clear
display_request         = 0                             # [RW/int] Should we initialize the
display at boot?
phase_enforcement       = (error)                       # [RO/int] Board should have full
security settings applied
recovery_reason         = 0                             # [RO/int] Recovery mode reason for
current boot
recovery_request        = 0                             # [RW/int] Recovery mode request
recovery_subcode        = 0                             # [RW/int] Recovery reason subcode
recoverysw_boot         = 0                             # [RO/int] Recovery switch position
at boot
recoverysw_cur          = (error)                       # [RO/int] Recovery switch current
position
recoverysw_ec_boot      = 0                             # [RO/int] Recovery switch position
at EC boot
ro_fwid                 = Google_Swanky.5216.238.5      # [RO/str] Read-only firmware ID
tpm_attack              = 0                             # [RW/int] TPM was interrupted since
this flag was cleared
tpm_fwver               = 0x00050003                    # [RO/int] Firmware version stored
in TPM
tpm_kernver             = 0x00030001                    # [RO/int] Kernel version stored in
TPM
tpm_rebooted            = 0                             # [RO/int] TPM requesting repeated
reboot (vboot2)
tried_fwb               = 0                             # [RO/int] Tried firmware B before A
this boot
try_ro_sync             = 0                             # [RO/int] try read only software
sync
vdat_flags              = 0x00002c56                    # [RO/int] Flags from VbSharedData
wipeout_request         = 0                             # [RW/int] Firmware requested
factory reset (wipeout)
wpsw_cur                = 1                             # [RO/int] Firmware write protect
hardware switch current position
```

The setting we are interested in is `dev_boot_usb`, so run `sudo crossystem dev_boot_usb=1` to enable -

```
chronos@localhost / $ sudo crossystem dev_boot_usb=1
chronos@localhost / $ sudo crossystem dev_boot_altfw=1
```

Now plug in the USB and reboot the chromebook. When you see the white screen warning about thrid party operating systems, press `CTRL+ALT+L` and you'll see a prompt to select the USB device to boot from.

NOTE: The old command for this was `dev_boot_legacy`, but it has since changed. We can see this when trying to set the old variable name, which leads us to setting `dev_boot_altfw`.

```
chronos@localhost / $ sudo crossystem dev_boot_legacy=1
Password:
!!!
!!! PLEASE USE 'dev_boot_altfw' INSTEAD OF 'dev_boot_legacy'
!!!
```

Using Crouton

Crouton allows you to install linux alongside ChromeOS on a chromebook. Click here to grab the latest crouton installer directly, or alternatively visit Crouton's Repository and click the goo link in the description for the same.

> To sync your Chromebook's local clipboard with your Linux install, grab the Crouton extension from the Chrome Web Store.

Once you have this file, be sure it is found in your Chromebook's `~/Downloads` directory using the file browser and run the commands below to install Linux -

```
# Install the crouton binary for use within your chromebook's shell
sudo install -Dt /usr/local/bin -m 755 ~/Downloads/crouton
# Passing -e for encryption, we install all the dependencies for X11 and name(-n) it i3
sudo crouton -e -t core,keyboard,audio,cli-extra,gtk-extra,extension,x11,xorg -n i3
# Enter the chroot
sudo enter-chroot -n i3
# Install i3
sudo apt install i3
# Tell Xorg to start i3 automatically
echo "exec i3" > ~/.xinitrc
# Exit the chroot
```

```
# Add an alias for starting i3 in crouton using X
sudo echo "alias starti3='sudo enter-chroot -n i3 xinit'" >> /home/chronos/user/.bashrc
```

Want i3-gaps instead? See the i3-gaps GitHub for instructions, or run the commands below.

```
sudo apt-get install software-properties-common
```

If you're unsure which Distro or DE to install, see the below commands for lists of supported versions.

```
# List supported Linux releases
sudo crouton -r list

# List supported Linux desktop environments
sudo crouton -t list

# Update chroot (you will need this eventually)
sudo crouton -u -n chrootname
```

Removing / editing chroots is done via the `edit-chroot` CLI -

```
# Print help text
sudo edit-chroot

# Remove chroot named i3
sudo edit-chroot -d i3

# Backup chroot
sudo edit-chroot -b chrootname
# Restore chroot from most recent tarball
sudo edit-chroot -r chrootname
# Restore from specific tarball (new machine?)
sudo edit-chroot -f mybackup.tar.gz
```

https://github.com/dnschneid/crouton https://github.com/pasiegel/i3-gaps-install-ubuntu/blob/master/i3-gaps https://launchpad.net/~simon-monette/+archive/ubuntu/i3-gaps https://stackoverflow.com/questions/53800051/repository-does-not-have-a-release-file-error

---

Revision #8
Created 7 January 2020 15:24:39 by Shaun Reed
Updated 18 December 2021 16:37:39 by Shaun Reed