UBports Documentation

Release 1.0

Marius Gripsgard

Jan 21, 2018

About

Welcome to the official documentation of the UBports project!

UBports develops the mobile phone operating system Ubuntu Touch. Ubuntu Touch is a mobile operating system focused on ease of use, privacy, and convergence.

On this website you find instructions how to install Ubuntu Touch on your mobile phone, user guides and detailed documentation on all system components. If this is your first time here, please consider reading our *introduction*.

If you want to help improving this documentation, *the Documentation contribute page* will get you started.

CHAPTER 1

Introduction

This is the documentation for the UBports project. Our goal is to create an open-source (GPL if possible) mobile operating system that converges and respects your freedom.

1.1 About UBports

The UBports project was founded by Marius Gripsgard in 2015 and in its infancy was a place where developers could share ideas and educate each other in hopes of bringing the Ubuntu Touch platform to more and more devices.

After Canonical suddenly announced their plans to terminate support of Ubuntu Touch in April of 2017, UBports and its sister projects began work on the open-source code, maintaining and expanding its possiblilities for the future.

1.2 About the Documentation

This documentation is always improving thanks to the members of the UBports community. It is written in ReStructuredText and converted into this readable form by Sphinx, ReCommonMark, and Read the Docs. You can start contributing by checking out the *Documentation intro*.

1.3 Attribution

This documentation was heavily modeled after the Godot Engine's Documentation, attribution to Juan Linietsky, Ariel Manzur and the Godot community.

CHAPTER 2

Processes

This section of the documentation details standardized processes for different teams.

Note: The process definitions are still a work in progress and need to be completed by the respective teams.

2.1 Issue-Tracking Guidelines

This document describes the standard process of dealing with new issues in UBports projects. Not to be confused with the *guide on writing a good bugreport*.

2.1.1 Where are bugs tracked?

Since our quality assurance depends heavily on the community, we try to track issues where the user would expect them, instead of seperated by repository. This means, that issues of almost all components that are distributed as with the system-image are tracked in the Ubuntu Touch tracker. An exception of this are click-apps that can be updated independently through the OpenStore.

Most other repositories track isues locally. If you're unsure whether a repository uses its own tracker or not, consult the README.md file. Repositories that don't track issues locally have their bugtracker disabled.

This page is mainly about the Ubuntu Touch tracker, but most principles apply to other projects as well.

Note: Practicality beats purity! Exceptions might apply and should be described in the projects README.md file.

2.1.2 GitHub projects

To increase transparency and communication, GitHub projects (Kanban-Boards) are used whereever practical. In case of github.com/ubports/ubuntu-touch, a single project is used for all issues. Projects support filtering by labels, so that

only issues that belong to a specific team or affect a specific device can be viewed.

These are the standard columns:

- **None (awaiting triage)**: The issue has been approved by a member of the qa team and is awaiting review from the responsible development team. if the issue is a bug, instructions to reproduce are included in the issue description. if the issue is a feature request, it has passed a primary sanity check by the qa-team but has not yet been accepted by the responsible development-team.
- **Accepted**: The issue has been accepted by the responsible development-team. If the issue is a bugreport, the team has decided that it should be fixable and accepts the responsibility. If the issue is a featrue request, the team thinks it should be implemented as described.
- **In Development**: A patch is in development. Usually means that a developer is assigned to the issue.
- **Quality Assurance**: The patch is completed and has passed initial testing. The QA team will now review it on all devices and provide feedback. If problems are found, the issue is moved back to "In Development".
- **Release Candidate**: The patch has passed QA and is ready for release. In case of deb-packages that are included in the system-image, the patch will be included in the next over-the-air update on the rc channel, and, if everything goes well, in the next release of the stable channel.
- **None (removed from the project)**: A patch has been released (issue closed with a message linking to the patch) or the issue has been rejected (issue closed and labeled "wontfix").

2.1.3 Labels

All issues - even closed ones - should be labeled to allow the use of GitHub's global filtering. For example, these are all of the issues labeled 'enhancement' inside @ubports. Consult the GitHub help pages for more information on search and filtering.

Here's a list of labels that are normally used by all repositories.

- **needs confirmation**: The bug needs confirmation and / or further information from affected users
- **bug**: This issue is a confirmed bug. If it's reproducable, reproduction steps are described.
- **opinion**: This issue needs further discussion.
- **enhancement**: This issue is a feature request.
- **question**: This issue is a support request or general question.
- **invalid**: This issue can not be confirmed or was reported in the wrong tracker.
- **duplicate**: This has already been reported somewhere else. Please provide a link and close.
- **help wanted**: This issue is ready to be picked up by a community developer.
- **good first issue**: This issue is not critical and trivial to fix. It is reserved for new contributors as a place to start.
- **wontfix**: It does not make sense to fix this bug, since it will probably resolve itself, it will be too much work to fix it, it's not fixable, or an underlying component will soon change.

Additional special labels can be defined. As an example, here's the labels used in the Ubuntu Touch tracker:

- **critical (devel)**: This critical issue that only occurs on the devel channel is blocking the release of the next rc image.
- **critical (rc)**: This critical issue that only occurs on the devel and rc channel is blocking the release of the next stable release. Usually, issues that can not simply be moved to a different release and have the power to postpone the release are labeled this.
- **device: [DEVICE CODENAME]**: This issue affects only the specified device(s).

- **team: [TEAM NAME]**: This issue is falls under the responsibility of a specific team (hal, middleware, UI).

Note: If a repository that tracks issues locally defines it's own labels, they should be documented in the README.md.

2.1.4 Milestones

Milestones are used for stable OTA releases only. In general, milestones for the work-in-progress OTA and the next OTA are created. The ETA is set, once the work on the release starts (that is 6 weeks from start date), but can be adjusted afterwards. See the *release-schedule* for more info.

2.1.5 Assignees

To make it transparent who's working on an issue, the developer should be assigned. This also allows the use of GitHub's global filtering as a type of TODO list. For example, this is everything assigned to mariogrip in @ubports.

Developers are asked to keep their list short and to keep the status of their issues up-to-date.

2.1.6 Examples

Bug Lifecycle

Note: The same principle applies to feature requests. The only difference is, that instead of the label **bug**, the label **enhancement** is used. The **needs confirmation** label is not applicable for feature requests.

- A *User* files a new bug using the issue-template.
- The *QA-Team* adds the label **needs confirmation** and tries to work with the user to confirm the bug and add potentially missing information to the report. Once the report is complete a **team-label** will be added to the issue, the issue will be put on the **awaiting-triage-list** of the project and the label needs confirmation will be replaced with **bug**.
- The affected *Team* will triage the issue and either reject (label **wontfix**, close and remove from the project) or accept (move to "Accepted") the issue. If no team member can be assigned and the issue can be handled by the community, it is labeled **help wanted**. If possible, the team will provide hints on how to resolve the issue and further details on the how the fix should be implemented. For non-critical issues that are trivial to fix, the label **good first issue** can be added as well.
- Once a *developer* is assigned and starts working on the issue, it is moved to "In Development". As soon as he has something to show for, the issue is moved to "Quality Assurance" for feedback. If neccessary, the developer should provide hints on how to test his patch in a comment on the issue.
- The *QA-Team* tests the fix on all devices and provides feedback to the developer. If problems are found, the issue goes back to "In Development", else it's closed and moved to "Release Candidate" to be included in the next release.
- Once the fix is included in the rc image, the issue is closed and (if not done already) added to a milestone. Once the milestone is released, the issue is removed from the project.

2.2 Release Schedule

OTA updates usually follow this rhythm:

- devel: daily builds
- rc: weekly if no critical issue exists in the devel channel
- stable: every six through eight weeks, if no critical issue exists in the rc channel

This is not a definitive cycle. Stable releases are ready when they're ready and should not introduce new bugs or ship very incomplete features. Since the OTA update can not be released before all critical issues are closed, the ETA might have to be moved by making an educated guess for when all the issues can be handled. If there are too many issues added to a milestone, they are either removed or added to the next milestone.

CHAPTER 3

Install Ubuntu Touch

Installing Ubuntu Touch is easy, and a lot of work has gone in to making the installation process less intimidating to less technical users. The UBports Installer is a nice graphical tool that you can use to install Ubuntu Touch on a supported device from your Linux, Mac or Windows computer.

Warning: If your're switching your device over from Android, you will not be able to keep any data that is currently on the device. Create an external backup if you want to keep it.

Go to the download page and download the version of the installer for your operating system:

- Windows: ubports-installer-<version-number>.exe
- macOS: ubports-installer-<version-number>.dmg
- Linux: ubports-installer-<version-number>.deb, ubports-installer-<version-number>.snap or ubports-installer-<version-number>.AppImage

Start the installer and follow the on-screen instructions that will walk you through the installation process. That's it! Have fun exploring Ubuntu Touch!

If you're an experienced android developer and want to help us bring Ubuntu Touch to more devices, visit the *porting section*.

3.1 Install on legacy Android devices

While the installation process is fairly simple on most devices, some legacy Bq and Meizu devices require special steps due to their closed nature. This part of the guide does not apply to any other devices or bq devices that are running the Canonical version of Ubuntu Touch.

Note: This is more or less uncharted territory. If the device manufacturers don't want you to install an alternative operating system on their devices, there's not a lot we can do about it. The instructions below are very vague and

should only be followed by experienced users. While we appreciate that lots of people want to use our OS, flashing a device with OEM tools shouldn't be done without a bit of know-how and plenty of research.

- BQ devices: Download the official Ubuntu Edition firmware from their website and use their SP Flash Tool to flash it.
- Meizu devices: You are pretty much stuck on Flyme. For the MX4, there are some instructions floating around for downgrading your OS, gaining root with an exploit, unlocking your bootloader, and so on. We aren't going to link to them here for obvious reasons. The Pro5 is Exynos-based and has its own headaches. You're even more at your own risk on these.

Warning: BE VERY CAREFUL! Getting this part wrong can permanantly damage or brick your device. NEVER check the "Format All" option in SP Flash Tool and carefully read everything that it tells you. Some users have destroyed the partition that holds their hardware IDs and can no longer connect to Wi-Fi or cellular networks.

CHAPTER 4

Daily use

This section of the documentation details common tasks that users may want to perform while using their Ubuntu Touch device.

4.1 Run desktop applications

Libertine allows you to use standard desktop applications in Ubuntu Touch.

To display and launch applications you need the *Desktop Apps Scope* which is available in the Canonical App Store. To install applications you need to use the commandline as described below.

4.1.1 Manage containers

Create a container

The first step is to create a container where applications can be installed:

```
libertine-container-manager create -i CONTAINER-IDENTIFIER
```

You can add extra options such as:

- `-n name` name is a more user friendly name of the container
- `-t type` type can be either `chroot` or `lxc`. Default is `chroot` and is compatible with every device. If the kernel of your device supports it then `lxc` is suggested.

The creating process can take some time, due to the size of the container (some hundred of megabytes).

Note: The `create` command shown above cannot be run directly in the terminal app, due apparmor restrictions. You can run it from another device using either `adb` or `ssh` connection. Alternatively, you can run it from the terminal app using a loopback ssh connection running this command: `ssh localhost`.

List containers

To list all containers created run: `libertine-container-manager list`

Destroy a container

```
libertine-container-manager destroy -i CONTAINER-IDENTIFIER
```

4.1.2 Manage applications

Once a container is set up, you can list the installed applications:

```
libertine-container-manager list-apps
```

Install a package:

```
libertine-container-manage install-package -p PACKAGE-NAME
```

Remove a package:

```
libertine-container-manager remove-package -p PACKAGE-NAME
```

Note: If you have more than one container, then you can use the option `-i CONTAINER-IDENTIFIER` to specify for which container you want to perform an operation.

4.1.3 Files

Libertine applications do have access to these folders:

- Documents
- Music
- Pictures
- Downloads
- Videos

4.1.4 Tipps

Locations

For every container you create there will be two directories created:

- A root directory `~/.cache/libertine-container/CONTAINER-IDENTIFIER/rootfs/` and
- a user directory `~/.local/share/libertine-container/user-data/CONTAINER-IDENTIFIER/`

Shell access

To execute any arbitrary command as root inside the container run:

```
libertine-container-manager exec -c COMMAND
```

For example, to get a shell into your container you can run:

```
libertine-container-manager exec -c /bin/bash
```

Note: When you launch bash in this way you will not get any specific feedback to confirm that you are now *inside* the container. You can check `ls /` to confirm for yourself that you are inside the container. The listing of `ls /` will be different inside and outside of the container.

To get a shell as user `phablet` run:

```
DISPLAY= libertine-launch -i CONTAINER-IDENTIFIER /bin/bash
```

4.1.5 Background

A display server coordinates input and output of an operating system. Most Linux distributions today use the X server. Ubuntu Touch does not use X, but a new display server called Mir. This means that standard X applications are not directly compatible with Ubuntu Touch. A compatibility layer called XMir resolves this. Libertine relies on XMir to display desktop applications.

Another challenge is that Ubuntu Touch system updates are released as OTA images. A consequence of this is that the root filesystem is read only. Libertine provides a container with a read-write filesystem to allow the installation of regular Linux desktop applications.

4.2 Run android applications

Anbox is a minimal android container and compatibility layer that allows you to run android applications on GNU/Linux operating systems.

Note:

- Anbox is in early development
- Anbox for Ubuntu Touch is in even more early development
- Anbox only works on the 16.04 version of Ubuntu Touch, which is also in early development
- Only selected devices are supported for the moment, more will be added in the future.

4.2.1 Supported devices

Make sure your device is supported:

Meizu Pro 5 (codename: *turbo*, name of the boot partition: *bootimg*) BQ M10 HD (codename: *cooler*, name of the boot partition: *boot*) BQ M10 FHD (codename: *frieza*, name of the boot partition: *boot*)

You will need the device codename and the name of your boot partition for the installation.

4.2.2 How to install

Warning: Because this feature is in such an early stage of development, the installation is only recommended for experienced users.

- *Install* the 16.04/devel channel on your supported device
- Open a terminal and run `export CODENAME="turbo" && export PARTITIONNAME="bootimg"`, but replace the part between the quotes respectively with the codename and name of the boot partition for your device. See the above list.
- Activate developer mode on your device.
- Connect the device to your computer computer and run the following commands:

```
adb shell sudo reboot -f bootloader
wget http://cdimage.ubports.com/anbox-images/anbox-boot-$CODENAME.img
sudo fastboot flash $PARTITIONNAME anbox-boot-$CODENAME.img
sudo fastboot reboot
rm anbox-boot-$CODENAME.img
```

- wait for the device to reboot, then run:

```
adb shell
sudo mount -o rw,remount /
sudo apt update
sudo apt install anbox-ubuntu-touch
anbox-setup
exit
```

- Done! Select “Anbox” in the apps scope to use android applications. You might have to refresh the apps scope (pull down from the center of the screen and release) for the app to show up.

4.2.3 Reporting bugs

Please *report any bugs* you come accross. Bugs with Ubuntu Touch 16.04 are reported in the normal Ubuntu Touch tracker and issues with anbox are reported on our downstream fork. Thank you!

CHAPTER 5

Advanced use

This section of the documentation details advanced tasks that power users may want to perform on their Ubuntu Touch device.

Note: Some of these guides involve making your system image writable, which may break OTA updates. The guides may also reduce the overall security of your Ubuntu Touch device. Please consider these ramifications before hacking on your device too much!

5.1 Shell access via adb

You can put your UBports device into developer mode and access a Bash shell from your PC. This is useful for debugging or more advanced shell usage.

5.1.1 Install ADB

First, you'll need ADB installed on your computer.

On Ubuntu:

```
sudo apt install android-tools-adb
```

On Fedora:

```
sudo dnf install android-tools
```

And on MacOS with Homebrew:

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

For Windows, grab the command-line tools only package from here.

5.1.2 Enable developer mode

Next, you'll need to turn on Developer Mode.

1. Reboot your device
2. Place your device into developer mode (Settings - About - Developer Mode - check the box to turn it on)
3. Plug the device into a computer with adb installed
4. Open a terminal and run `adb devices`.

Note: When you're done using the shell, it's a good idea to turn Developer Mode off again.

If there's a device in the list here (The command doesn't print "List of devices attached" and a blank line), you are able to use ADB successfully. If not, continue to the next section.

5.1.3 Add hardware IDs

ADB doesn't always know what devices on your computer it should or should not talk to. You can manually add the devices that it does not know how to talk to.

Just run the command for your selected device if it's below. Then, run `adb kill-server` followed by the command you were initially trying to run.

Fairphone 2:

```
printf "0x2ae5 \n" >> ~/.android/adb_usb.ini
```

Oneplus One:

```
printf "0x9d17 \n" >> ~/.android/adb_usb.ini
```

5.2 Shell access via ssh

You can use ssh to access a shell from your PC. This is useful for debugging or more advanced shell usage.

You need a ssh key pair for this. Logging in via password is disabled by default.

5.2.1 Create your public key

If not already created, create your public key, default choices should be fine for LAN, you can leave empty password if you don't want to deal with it each time:

```
ssh-keygen
```

5.2.2 Copy the public key to your device

You need then to transfer your public key to your device. There are multiple ways to do this. For example:

- Connect the ubports device and the PC with a USB cable. Then copy the file using your filemanager.

- Or transfer the key via the internet by mailing it to yourself, or uploading it to your own cloud storage, or webserver, etc.
- You can also connect via *adb* and use the following command to copy it:

```
adb push ~/.ssh/id_rsa.pub /home/phablet/
```

5.2.3 Configure your device

Now you have the public key on the UBports device. Let's assume it's stored as `/home/phablet/id_rsa.pub`. Use the terminal app or and adb connection to perform the following steps on your phone.

```
mkdir /home/phablet/.ssh
chmod 700 /home/phablet/.ssh
cat /home/phablet/id_rsa.pub >> /home/phablet/.ssh/authorized_keys
chmod 600 /home/phablet/.ssh/authorized_keys
chown -R phablet.phablet /home/phablet/.ssh
```

Now start the ssh server:

```
sudo android-gadget-service enable ssh
```

5.2.4 Connect

Now everything is set up and you can use `ssh`

```
ssh phablet@<ip-address>
```

Of course you can now also use `scp` or `sshfs` to transfer files.

5.2.5 References

- askubuntu.com: How can I access my Ubuntu phone over ssh?
- gurucubano: BQ Aquaris E 4.5 Ubuntu phone: How to get SSH access to the ubuntu-phone via Wifi

5.3 Switch release channels

5.4 Screen Casting your UT device to your computer

Ubuntu Touch comes with a command line utility called `mirscreencast` which dumps screen frames to a file. The idea here is to stream UT display to a listening computer over the network or directy trough adb so that we can either watch it live or record it to a file.

5.4.1 Using adb

You can catch output directly from `adb exec-out` command and forward it to mplayer:

```
adb exec-out timeout 120 mirscreencast -m /run/mir_socket --stdout --cap-interval 2 -
↪s 384 640 | mplayer -demuxer rawvideo -rawvideo w=384:h=640:format=rgba -
```

NB: `timeout` here is used in order to kill process properly on device (here 120 seconds). Otherwise process still continuing even if killed on computer. You can reduce or increase frame per second with‘‘–cap-interval‘‘ (1 = 60fps, 2=30fps, . . .) and size of frames `384 640` means width=384 height=640

5.4.2 Via network

On receiver

For real time casting:

Prepare your computer to listen to a tcp port(here 1234) and forward raw stream to a video player (here mplayer) with a frame size of 384x640:

```
nc -l -p 1234 | gzip -dc | mplayer -demuxer rawvideo -rawvideo␣
↪w=384:h=640:format=rgba -
```

For stream recording:

Prepare your computer to listen to a tcp port(here 1234), ungzip and forward raw stream to a video encoder (here mencoder):

```
nc -l -p 1234 | gzip -dc | mencoder -demuxer rawvideo -rawvideo␣
↪fps=60:w=384:h=640:format=rgba -ovc x264 -o out.avi -
```

On device

Forward and gzip stream with 60fps (–cap-interval 1) and frame size of 384x640 to computer 10.42.0.209 on port 1234

```
mirscreencast -m /run/mir_socket --stdout --cap-interval 1 -s 384 640 | gzip -c | nc␣
↪10.42.0.209 1234
```

Example script

This script allows you to screen cast remote UT device to your local PC (must have ssh access to UT and mplayer installed on PC), run it on your computer:

```
#!/bin/bash
SCREEN_WIDTH=384
SCREEN_HEIGHT=640
PORT=1234
FORMAT=rgba


if [[ $# -eq 0 ]] ; then
    echo 'usage: ./mircast.sh UT_IP_ADDRESS , e.g: ./mircast.sh 192.168.1.68'
    exit 1
fi

IP=$1
```

```
LOCAL_COMMAND='nc -l -p $PORT | gzip -dc | mplayer -demuxer rawvideo -rawvideo w=
↪$SCREEN_WIDTH:h=$SCREEN_HEIGHT:format=$FORMAT -'
REMOTE_COMMAND="mirscreencast -m /run/mir_socket --stdout --cap-interval 1 -s $SCREEN_
↪WIDTH $SCREEN_HEIGHT | gzip -c | nc \$SSH_CLIENT $PORT"
ssh -f phablet@$IP "$REMOTE_COMMAND"
eval $LOCAL_COMMAND
```

You can download it here: `files/mircast.sh`

5.4.3 References

- initial source: https://wiki.ubuntu.com/Touch/ScreenRecording
- demo: https://www.youtube.com/watch?v=HYm4RUwwo5Q

5.5 Reverse tethering

Some users may not have an available wifi connection for their phone nor a data subscription from their mobile provider. This short tutorial will help you to connect your Ubuntu Touch to your computer to access internet.

Prerequisite: Phone is connected to the computer with usb and developer mode enabled.

5.5.1 Steps

1. On phone: `android-gadget-service enable rndis`
2. On computer: get your rndis ip address e.g:`hostname -I`
3. On phone:

   add gateway: `sudo route add default gw YOUR_COMPUTER_RNDIS_IP`

   add nameservers: `echo "nameserver 8.8.8.8" | sudo tee /etc/resolv.conf`

   refresh dns table: `resolvconf -u`
4. On computer:

   enable ip forwarding: `echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward`

   apply NAT: `sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/8 -o eth0 -j MASQUERADE`

5.5.2 References

- askubuntu: https://askubuntu.com/questions/655321/ubuntu-touch-reverse-tethering-and-click-apps-updates

5.6 CalDAV and CardDAV synchronization

CalDAV and CardDAV are protocols to syncronize calendars and contacts with a remote server. Many email-hosters provide a CalDAV and CardDAV interface.

Note: CalDAV Sync can also be set up in using the calendar app. Open the app, click on the little calendar icon in the top right corner and select "Add internet calendar > Generic CalDAV". Enter your calendar URL as well as your username and password to complete the process.

At the moment, there is no carddav implemention directly accessible from the Ubuntu Touch graphical user-interface, so the only way to sync carddav is by using syncevolution + cron. However, there is a simple way to do that with a script that you can run in the terminal or via phablet SSH connection. These instructions work for caldav as well.

1. Follow this guide to activate Developer Mode and ADB (or SSH) connection.
2. Download this script (let's call it dav.sh) and edit the following variables:
   - server side : CAL_URL, **CONTACTS_URL**, USERNAME, PASSWORD (of your ownCloud/nextCloud/baikal/SOGO/... server)
   - CONTACT and CALENDAR _ NAME / VISUAL_NAME / CONFIG_NAME (it's more cosmetic)
   - CRON_FREQUENCY (for the frequency of synchronisation)
   - Line 61: write `sudo sh -c "echo '$COMMAND_LINE' > /sbin/sogosync"` , instead of `sudo echo "$COMMAND_LINE" > /sbin/sogosync`, to avoid permission denied error
3. Move the file to your UbuntuTouch device, either by file manager or with adb:

```
adb push dav.sh /home/phablet
```

4. Connect with the phablet shell (`adb shell`) or directly on the phone Terminal app and type the following:

```
chmod +x dav.sh
./dav.sh
```

5.6.1 Sources:

- https://askubuntu.com/questions/616081/ubuntu-touch-add-contact-list-and-calendars/664834#664834
- https://gist.github.com/boTux/069b53d8e06bdb9b9c97
- https://gist.github.com/tcarrondo
- https://gist.github.com/bastos77
- https://askubuntu.com/questions/601910/ssh-ubuntu-touch

CHAPTER 6

Contributing to UBports

Welcome! You're probably here because you want to contribute to UBports. The pages you'll find below here will help you do this in a way that's helpful to both the project and yourself.

If you're just getting started, we always need help with *thorough bug reporting*. If you are multilingual, *translations* are also a great place to start.

If those aren't enough for you, see our contribute page for an introduction of our focus groups.

6.1 Bug reporting

This page contains information to help you help us by reporting an actionable bug for Ubuntu Touch. It does NOT contain information on reporting bugs in apps, most of the time their entry in the OpenStore will specify where and how to do that.

6.1.1 Get the latest Ubuntu Touch

This might seem obvious, but it's easy to miss. Go to (Settings - Updates) and make sure that your device doesn't have any Ubuntu updates available. If not, continue through this guide. If so, update your device and try to reproduce the bug. If it still occurs, continue through this guide. If not, do a little dance! The bug has already been fixed and you can continue using Ubuntu Touch.

6.1.2 Check if the bug is already reported

Open up the bug tracker for ubuntu-touch.

First, you'll need to make sure that the bug you're trying to report hasn't been reported before. Search through the bugs reported. When searching, use a few words that describe what you're seeing. For example, "Lock screen transparent" or "Lock screen shows activities".

If you find that a bug report already exists, select the "Add your Reaction" button (it looks like a smiley face) and select the +1 (thumbs up) reaction. This shows that you are also experiencing the bug.

If the report is missing any of the information specified later in this document, please add it yourself to help the developers fix the bug.

6.1.3 Reproduce the issue you've found

Next, find out exactly how to recreate the bug that you've found. Document the exact steps that you took to find the problem in detail. Then, reboot your phone and perform those steps again. If the problem still occurs, continue on to the next step. If not...

6.1.4 Getting Logs

We appreciate as many good logs as we can get when you report a bug. In general, /var/log/dmesg and the output of /android/system/bin/logcat are helpful when resolving an issue. I'll show you how to get these logs.

To get set ready, follow the steps to *set up ADB*.

Now, you can get the two most important logs.

dmesg

The **dmesg** (*display message* or *driver message*) command displays debug messages from the kernel.

1. Using the steps you documented earlier, reproduce the issue you're reporting
2. cd to a folder where you're able to write the log
3. Run the command: *adb shell dmesg > UTdmesg.txt*

This log should now be located at UTdmesg.txt under your working directory, ready for uploading later.

logcat

The **logcat** (*log concatenator*) command displays debug information from various parts of the underlying android system.

1. cd to a folder where you're able to write the log
2. Run the command: *adb shell /android/system/bin/logcat -d > UTlogcat.txt*
3. Using the steps you documented earlier, reproduce the issue you're reporting

This log will be located at UTlogcat.txt in your current working directory, so you'll be able to upload it later.

6.1.5 Making the bug report

Now it's time for what you've been waiting for, the bug report itself! Bug reports need to be filed in English.

First, pull up the bug tracker and click "New Issue". Log in to GitHub if you haven't yet.

Next, you'll need to name your bug. Pick a name that says what's happening, but don't be too wordy. Four to eight words should be enough.

Now, write your bug report. A good bug report includes the following:

- What happened: A synopsis of the erroneous behavior
- What I expected to happen: A synopsis of what should have happened, if there wasn't an error

- Steps to reproduce: You wrote these down earlier, right?
- Logs: Attach your logs by clicking and dragging them into your GitHub issue.
- Software Version: Go to (Settings - About) and list what appears on the "OS" line of this screen. Also include the release channel that you used when you installed Ubuntu on this phone.

Once you're finished with that, post the bug. You can't add labels yourself, so please don't forget to state the device you're experiencing the issue on in the description so a moderator can easily add the correct tags later.

A developer or QA-team member will confirm and triage your bug, then work can begin on it. If you are missing any information, you will be asked for it, so make sure to check in often!

6.2 Quality Assurance

This page explains how to help the UBports QA team, both as an official member or a new contributor. Please also read the *issue tracking* and *bugreporting* guides to better understand the workflow.

6.2.1 Confirming bug reports

Unconfirmed bugreports are labeld **needs confirmation** to enable global filtering. Browse through the list, read the bugreports and try to reproduce the issues that are described. If neccessary, add *missing information or logs, or improve the quality of the report by other means*. Leave a comment stating your device, channel, build number and whether or not you were able to reproduce the issue.

If you have write-access to the repository, you can replace the **needs confirmation** label with **bug** (to mark it confirmed) or **invalid** (if the issue is definitely not reproducable). In that case it should be closed.

If you find two issues describing the same problem, leave a comment and try to find their differences. If they are in fact identical, close the newer one and label it **duplicate**.

6.2.2 Testing patches

Once a developer finished working on an issue, it's moved to the quality assurance column of the GitHub project. Check if the issue is still present in the latest update on the devel channel and try to find any problems it is causing. Check if the developer mentioned specific things to look out for when testing and leave a comment detailing your experience. If you have write-access to the repository, you can move the issue back to **In Development** or forward to **Release Candidate** as specified by the *issue tracking guidelines*.

6.2.3 Initial triaging of issues

Initial triaging of new issues is done by QA-team members with write-access to the repository. If a new issue is filed, read the report and add the correct labels as specified by the *issue tracking guidelines*. You can also immediately start confirming the bugreport.

If the new issue has already been reported elsewhere, label it **duplicate** and close it.

6.3 Documentation

Tip: Documentation on this site is written in ReStructuredText, or RST for short. Please check the RST Primer if you are not familiar with RST.

This page will guide you through writing great documentation for the UBports project that can be featured on this site.

6.3.1 Documentation guidelines

These rules govern *how* you should write your documentation to avoid problems with style, format, or linking. If you don't follow these guidelines, we will not accept your document.

Title

All pages must have a document title that will be shown in the table of contents (left sidebar) and at the top of the page. This title is underlined with equals signs.

Titles should be sentence cased rather than Title Cased. For example:

```
Incorrect casing:
    Writing A Good Bug Report
Correct casing:
    Writing a good bug report
Correct casing when proper nouns are involved:
    Installing Ubuntu Touch on your phone
```

There isn't a single definition of title casing that everyone follows, but sentence casing is easy. This helps keep capitalization in the table of contents consistent.

Table of contents

People can't navigate to your new page if they can't find it. Neither can Sphinx. That's why you need to add new pages to Sphinx's table of contents.

You can do this by adding the page to the `index.rst` file in the same directory that you created it. For example, if you create a file called "newpage.rst", you would add the line marked with a chevron (>) in the nearest index:

```
.. toctree::
    :maxdepth: 1
    :name: example-toc

    oldpage
    anotheroldpage
>   newpage
```

The order matters. If you would like your page to appear in a certain place in the table of contents, place it there. In the previous example, newpage would be added to the end of this table of contents.

Warnings

Your edits must not introduce any warnings into the documentation build. If any warnings occur, the build will fail and your pull request will be marked with a red 'X'. Please ensure that your RST is valid and correct before you create a pull request. This is done automatically (via sphinx-build crashing with your error) if you follow our build instructions below.

6.3.2 Contribution workflow

Note: You will need a GitHub account to complete these steps. If you do not have one, click here to begin the process of making an account.

Directly on GitHub

Read the Docs and GitHub make it fairly simple to contribute to this documentation. This section will show you the basic workflow to get started by editing an existing page on GitHub

1. Find the page you would like to edit
2. Click the "Edit on GitHub" link to the right of the title
3. Make your changes to the document. Remember to write in ReStructuredText!
4. Propose your changes as a Pull Request.

If there are any errors with your proposed changes, the documentation team will ask you to make some changes and resubmit. This is as simple as editing the file on GitHub from your fork of the repository.

Manually forking the repository

You can make more advanced edits to our documentation by forking ubports/docs.ubports.com on GitHub. If you're not sure how to do this, check out the excellent GitHub guide on forking projects.

6.3.3 Building this documentation locally

If you'd like to build this documentation *before* sending a PR (which you should), follow these instructions on your *local copy* of your fork of the repository.

The documentation can be built by running `./build.sh` in the root of this repository. The script will also create a virtual build environment in `~/ubportsdocsenv` if none is present.

If all went well, you can enter the `_build` directory and open `index.html` to view the UBports documentation.

6.3.4 Current TODOs

This page lists the TODOs that have been included in this documentation. If you know how to fix one, please send us a PR to make it better!

Documentation TODOs

Todo: This is a workaround for ubports/ubports-boot #1 and ubports/ubports-boot #2. It should not be considered a permanent fix. Be prepared to revert this later when these bugs have been fixed.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/docsubportscom/checkouts/latest/porting/building-ubports-boot.rst, line 47.)

Todo: Change the rootstock link to point to UBports once the actuallyfixit PR is merged.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/docsubportscom/checkouts/latest/porting/installing-16-04.rst, line 29.)

Todo: This should be a little heavier on "What to do when something goes wrong" content.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/docsubportscom/checkouts/latest/porting/installing-16-04.rst, line 74.)

To create a todo, add this markup to your page:

```
.. todo:

   My todo text
```

6.4 Translations

Although English is the official base language for all UBports projects we believe you have the right to use it in any language you want.

We are working hard to meet that goal, and you can help as well.

There are two levels for this:

- A casual approach, as a translator volunteer.
- A fully committed approach as a UBports Member, filling in this application.

6.4.1 Tools for Translation

For everyone: A web based translation tool called Weblate. This is the recommended way.

- For advanced users: Working directly on .po files with the editor of your choice, and a GitHub account. The .po files for each project are in their repository on our GitHub organization.

A Translation Forum to discuss on translating Ubuntu Touch and its core apps.

6.4.2 How-To

UBports Weblate

You can go to UBports Weblate, click on "Dashboard" button, go to a project, and start making anonymous suggestions without being registered. If you want to save your translations, you must be logged in.

For that, go to UBports Weblate and click on the "Register" button. Once in the "Registration" page, you'll find two options:

- Register using a valid email address, a username, and your full name. You'll need to resolve an easy control question too.
- Register using a third party registration. Currently the system supports accounts from openSUSE, GitHub, Fedora, and Ubuntu.

Once you're logged in, the site is self-explanatory and you'll find there all the options and customization you can do.

Now, get on with it. The first step is to search if your language already exists in the project of your choice.

If your language is not available for a specific project, you can add it yourself.

You decide how much time you can put into translation. From minutes to hours, everything counts.

.po file editor

As was said up above, you need a file editor of your choice and a GitHub account to translate .po files directly.

There are online gettext .po editors and those you can install in your computer.

You can choose whatever editor you want, but we prefer to work with free software only. There are too many plain text editors and tools to help you translate .po files to put down a list here.

If you want to work with .po files directly you know what you're doing for sure.

6.4.3 Translation Team Communication

The straightforward and recommended way is to use the forum category that UBports provides for this task.

To use it you need to register, or login if you're registered already.

The only requirement is to start your post putting down your language in brackets in the "Enter your topic title here" field. For example, `[Spanish] How to translate whatever?`

Just for your information, some projects are using Telegram groups too, and some teams are still using the Ubuntu Launchpad framework.

In your interactions with your team you'll find the best way to coordinate your translations.

6.4.4 License

All the translation projects, and all your contributions to this project, are under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) license that you explicitly accept by contributing to the project.

Go to that link to learn what this exactly means.

6.5 Monetary support

You can help us keep the lights on at UBports by becoming a patron on Liberapay or Patreon!

Your contribution finances our server infrastructure and debug services, and helps covering additional costs.

CHAPTER 7

App development

Welcome to an open source and free platform under constant scrutiny and improvement by a vibrant global community, whose energy, connectedness, talent and commitment is unmatched. Ubuntu is also the third most deployed desktop OS in the world.

7.1 Introduction

7.2 Clickable

Clickable is a meta-build system for Ubuntu Touch applications that allows you to compile, build, test and publish click packages. It is currently the easiest and most convenient way of building click packages for Ubuntu Touch.

See here for getting started Read the Docs.

7.3 Ubuntu UI-Toolkit

Here you can find the API documentation for the Ubuntu UI toolkit.

- QML API
- Cordova HTML5 API
- Click packages

7.4 Ubuntu SDK IDE (unsupported)

The Ubuntu SDK IDE is no longer supported by Canonical, and UBports does not currently have the the time and manpower to get it to a working state.

It can still be installed in Ubuntu 16.04, but issues are expected.

```
sudo add-apt-repository ppa:ubuntu-sdk-team/ppa
sudo apt update && sudo apt dist-upgrade
sudo apt install ubuntu-sdk
sudo reboot # or logout/login
```

7.5 Cookbook

A collection of external resources

7.5.1 App development cookbook

Unofficial resources

In this section you will find links to external resources about creating applications for Ubuntu Touch.

- Ubuntu Touch programming book

Playground

In a completely free and open source community, it is natural to have community members exploring the limits of the platform in many many directions. In this section you will find links to external resources that do exactly that: Explore. The purpose of this list is to show the unlimited possibilities of an open platform like Ubuntu Touch.

Note: The resources listed here do not neccessarily represent the officially endorsed way of developing applications for Ubuntu Touch, but are interesting experiments.

- Free Pascal development for Ubuntu Touch
- Lazarus development for Ubuntu Touch
- Geany on Ubuntu Touch device as text editor, source code editor, debugger and compiler for multiple languages

CHAPTER 8

Porting information

Note: If you are looking for information on installing Ubuntu Touch on a supported device, or if you would like to check if your device is supported, please see this page.

This section will introduce you to some of the specifics of porting Ubuntu Touch to an Android device.

Before you begin, you'll want to head over to the Halium porting guide and get your `systemimage` built without errors. Once you're at the point of installing a rootfs, you can come back here.

Start at *Building ubports-boot* if you'd like to install the UBports Ubuntu Touch 16.04 rootfs.

8.1 Building ubports-boot

Ubuntu Touch uses Upstart rather than Systemd for its init daemon. Because of this, it is not fully Halium-compatible and is not able to use the vanilla hybris-boot that Halium produces. For this reason, we need to build ubports-boot.

8.1.1 Add source to Halium tree

The first thing that needs to be done is adding the ubports-boot source to your Halium tree. You may choose to do this by adding it to your local manifests (recommended) or simply cloning it in place.

Add to local manifest

Create a new file in `.repo/local_manifests` called `ubports-boot.xml`. Add the following to the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
    <project name="ubports/ubports-boot" path="halium/ubports-boot" remote="hal"␣
↪revision="master" />
</manifest>
```

Now, just do a `repo sync` to download the new source.

Manual clone

You may also choose to clone the ubports-boot repository into your tree manually. It should be placed into `BUILDDIR/halium/ubports-boot`.

If you share your code and build instructions, please note that you've done this.

8.1.2 Include in build system

The android build system won't know where to find ubports-boot normally. To fix this, open the file `BUILDDIR/build/core/main.mk` in a text editor. Find the comment `# Specific projects for Halium` and add `halium/ubports-boot` to the list of `subdirs`

```
subdirs += \
        halium/hybris-boot \
        halium/ubports-boot
```

8.1.3 Fix mounts

Todo: This is a workaround for ubports/ubports-boot #1 and ubports/ubports-boot #2. It should not be considered a permanent fix. Be prepared to revert this later when these bugs have been fixed.

The initramfs of the ubports-boot image takes a commandline parameter, `datapart=`, which points it to the block device that is normally mounted at `/data` in Android. This parameter is automatically embedded into the image at build time by finding `/data` in your device's default fstab and using its source. Unfortunately, this source is generally a node in `/dev/block/bootdevice/by-name` or something else strange, which is only available in an Android initramfs. To fix this, Halium uses the fixup-mountpoints script, which you are probably familiar with by now.

The ubports-boot initramfs does not use the fixup-mountpoints script, nor does it put block devices in the proper place for mounting without modification. For this reason, we are going to have to edit your device's fstab.

The first step to this process is figuring out where your fstab actually is. For most, this is inside `BUILDDIR/device/MANUFACTUER/CODENAME/rootdir/etc` and it is named either `fstab.qcom` or `fstab.devicename`. Open the file for editing.

With the file open, change the `src`, or first attribute, of the `/data` mountpoint to what you put in fixup-mountpoints for `/data`, but without the `/block` folder. For example, on the Moto G5 Plus, the block device is `/dev/block/mmcblk0p54`, so I put `/dev/mmcblk0p54` in this place. Also change the type to ext4, if it is not already.

Now, remove all `context=` options from all block devices in the file. Save and exit.

For an example of this, see this commit on universalsuperbox/android_device_motorola_potter. Removing the `wait` flag is not required and was an accident.

8.1.4 Edit init.rc

Some Android services conflict with ofono in 16.04 and will cause your device to reboot without warning, about 30-60 seconds after it boots. We will need to disable these services until the issue is resolved.

To do this, open up your device's default `init.rc` (this is likely init.qcom.rc or init.[codename].rc), comment out any `import` statements, and add `disabled` to services like rild, qti, and others that interface with the

radio. Most of them have a `user radio` line. For an example, see commit 7875b48b on UniversalSuperBox/android_device_motorola_potter

8.1.5 Edit kernel config

Ubuntu Touch requires a slightly different kernel config than Halium, including enabling Apparmor. Luckily, we have a nice script for this purpose, `check-kernel-config`. It's included in the ubports-boot repository. Simply run it on your config as follows:

```
./halium/ubports-boot/check-kernel-config path/to/my/defconfig -w
```

You may have to do this twice. It will likely fix things both times. Then, run the script without the `-w` flag to see if there are any more errors. If there are, fix them manually. Once finished, run the script without the `-w` flag one more time to make sure everything is correct.

8.1.6 Build the image

Once ubports-boot is in place, you can build it quite simply. You will also need to rebuild system.img due to our changes.

1. `cd` to your Halium BUILDDIR
2. `source build/envsetup.sh`
3. Run `breakfast` or `lunch`, whichever you use for your device
4. `mka ubports-boot`
5. `mka systemimage`

8.1.7 Continue on

Now that you have ubports-boot built, you can move on to *Installing Ubuntu Touch 16.04 images on Halium*.

8.2 Installing Ubuntu Touch 16.04 images on Halium

Warning: These steps will wipe all of the data on your device. If there is anything that you would like to keep, ensure it is backed up and copied off of the device before continuing. It is not a good idea to port Ubuntu Touch using the device you rely on every day as a testing device. You will lose data. I speak from personal experience.

Now that you've *built ubports-boot*, we'll use a script called `rootstock-touch-install` to install an Ubuntu Touch rootfs on your device.

In order to install Ubuntu Touch, you will need a recovery with Busybox, such as TWRP, installed on your phone. You will also need ensure the /data partition is formatted with ext4 and does not have any encryption on it.

8.2.1 Install ubports-boot

We'll need to install the ubports-boot image before installing an image. Reboot your phone into fastboot mode, then do the following from your Halium tree:

```
cout
fastboot flash boot ubports-boot.img
```

8.2.2 Download the rootfs

Next we'll need to download the rootfs (root filesystem) that's appropriate for your device. Right now, we only have one available. Simply download `ubports-touch.rootfs-xenial-armhf.tar.gz` from our CI server. If you have a 64-bit ARM (aarch64) device, this same rootfs should work for you. If you have an x86 device, let us know. We do not have a rootfs available for these yet.

8.2.3 Install system.img and rootfs

Todo: Change the rootstock link to point to UBports once the actuallyfixit PR is merged.

Download the rootstock-touch-install script from universalsuperbox/rootstock-ng. Boot your device into recovery and run the script as follows:

```
rootstock-touch-install path/to/rootfs.tar.gz path/to/system.img
```

The script will copy and extract the files to their proper places, then allow you to set the phablet user's password. If it gets all the way to `rebooting device` and doesn't seem to produce any errors, it's time to continue to the next step. If something goes wrong, please get in touch with us. If your device doesn't reboot automatically, reboot it using your recovery's interface.

If you get errors similar to `broken pipe` or `out of memory`, use the `-b` option to push the busybox or toybox build that came from your tree. These may have fewer bugs than your recovery's busybox.

8.2.4 Get SSH access

When your device boots, it will likely stay at the bootloader screen. However, you should also get a new network connection on the computer you have it plugged in to. We will use this to debug the system.

To confirm that your device has booted correctly, run `dmesg -w` and watch for "GNU/Linux device" in the output. If you instead get something similar to "ubports initrd i hit a nail", please get in contact with us so we can find out why. You may also choose to run `watch ip link` and look for changes in network devices.

Similar to the Halium reference rootfs, you should set your computer's IP on the newly connected RNDIS interface to `10.15.19.100` if you don't get one automatically. Then, run the following to access your device:

```
ssh phablet@10.15.19.82
```

The password will be the one that you set while running rootstock.

8.2.5 Make / writeable

Before we make any changes to the rootfs (which is in the next step), you'll need to remount it with write permissions. To do that, run the following command:

```
sudo mount -o remount,rw /
```

8.2.6 Add udev rules

Now that you're logged in, you must create some udev rules to allow Ubuntu Touch software to access your hardware. Run the following command, replacing [codename] with your device's codename.:

```
sudo -i # And enter your password
cat /var/lib/lxc/android/rootfs/ueventd*.rc|grep ^/dev|sed -e 's/^\/dev\///'|awk '
↪{printf "ACTION==\"add\", KERNEL==\"%s\", OWNER=\"%s\", GROUP=\"%s\", MODE=\"%s\"\n
↪",$1,$3,$4,$2}' | sed -e 's/\r//' >/usr/lib/lxc-android-config/70-[codename].rules
```

Now, reboot the device. If all has gone well, you will eventually see the Ubuntu Touch spinner followed by Unity 8. Your lock password is the same as you set for SSH.

Todo: This should be a little heavier on "What to do when something goes wrong" content.
