

Linux Lab v0.9 Manual

[TinyLab Community](#) | [Tinylab.org](#)

January 13, 2022

目录

1. Linux Lab Overview	7
1.1 Project Introduction	7
1.2 Project Homepage	7
1.3 Demonstration	8
1.3.1 Free Video Courses In Chinese	8
1.3.2 Non-Free Video Courses In Chinese	8
1.4 Project Functions	9
1.5 Project History	9
1.5.1 Project Origins	9
1.5.2 Problems Solved	9
1.5.3 Project Born	10
2. Linux Lab Installation	11
2.1 Hardware and Software Requirement	12
2.2 Docker Installation	12
2.3 Choose a working directory	13
2.4 Switch to normal user	14
2.5 Download the lab	14
2.6 Run and login the lab	14
2.7 Update and rerun the lab	15
2.8 Quickstart: Boot a board	16
3. Linux Lab Kickstart	17
3.1 Using boards	17
3.1.1 List available boards	17
3.1.2 Choosing a board	19
3.1.3 Using as plugins	21
3.1.4 Configure boards	22
3.2 Build in one command	22
3.3 Detailed Operations	23

3.3.1	Downloading	23
3.3.2	Checking out	23
3.3.3	Patching	24
3.3.4	Configuration	24
3.3.5	Building	25
3.3.6	Saving	25
3.3.7	Booting	26
4.	Linux Lab Advance	28
4.1	Using Linux Kernel	28
4.1.1	non-interactive configuration	28
4.1.2	using kernel modules	29
4.1.3	using kernel features	30
4.1.4	Create new development branch	31
4.1.5	Use standalone git repository	32
4.2	Using Uboot Bootloader	32
4.3	Using Qemu Emulator	34
4.4	Using Toolchains	34
4.5	Using Rootfs	35
4.6	Debugging Linux and Uboot	36
4.6.1	Debugging Linux	36
4.6.2	Debugging Uboot	37
4.7	Test Automation	37
4.8	File Sharing	39
4.8.1	Install files to rootfs	39
4.8.2	Share with NFS	39
4.8.3	Transfer via tftp	40
4.8.4	Share with 9p virtio	40
4.9	Learning Assembly	41
4.10	Learning C	42
4.10.1	Host build and Run	42

4.10.2 Cross build and Run	42
4.11 Running any make goals	43
4.12 More Usage	43
5. Linux Lab Development	45
5.1 Choose a board supported by qemu	45
5.2 Create the board directory	45
5.3 Clone a Makefile from an existing board	45
5.4 Configure the variables from scratch	45
5.5 At the same time, prepare the configs	45
5.6 Choose the versions of kernel, rootfs and uboot	46
5.7 Configure, build and boot them	47
5.7.1 Speed up compiling and save disk life	47
5.8 Save the images and configs	48
5.9 Upload everything	48
6. FAQs	49
6.1 Docker Issues	49
6.1.1 Speed up docker images downloading	49
6.1.2 Docker network conflicts with LAN	49
6.1.3 Why not allow running Linux Lab in local host	49
6.1.4 Run tools without sudo	49
6.1.5 Network not work	50
6.1.6 Client.Timeout exceeded while waiting headers	50
6.1.7 Restart Linux Lab after host system shutdown or reboot	51
6.1.8 the following directives are specified both as a flag and in the configuration file	51
6.1.9 pathspec FETCH_HEAD did not match any file known to git	51
6.1.10 Docker not work in Ubuntu 20.04	52
6.1.11 Error creating aufs mount	52
6.2 Qemu Issues	52
6.2.1 Why kvm speeding up is disabled	52

6.2.2 Poweroff hang	53
6.2.3 How to exit qemu	53
6.2.4 Boot with missing sdl2 libraries failure	53
6.3 Environment Issues	54
6.3.1 NFS/tftpboot not work	54
6.3.2 How to switch windows in vim	54
6.3.3 How to delete typo in shell command line	54
6.3.4 Language input switch shortcuts	54
6.3.5 How to tune the screen size	54
6.3.6 How to work in fullscreen mode	55
6.3.7 How to record video	56
6.3.8 Linux Lab not response	56
6.3.9 VNC login with failures	56
6.3.10 Ubuntu Snap Issues	57
6.3.11 How to exit fullscreen mode of vnc clients	57
6.4 Lab Issues	57
6.4.1 No working init found	57
6.4.2 linux/compiler-gcc7.h: No such file or directory	58
6.4.3 linux-lab/configs: Permission denied	58
6.4.4 scripts/Makefile.headersinst: Missing UAPI file	58
6.4.5 unable to create file: net/netfilter/xt_dscp.c	58
6.4.6 how to run as root	58
6.4.7 not in supported list	59
6.4.8 is not a valid rootfs directory	59
7. Contact and Sponsor	60
Contact Us	60
Give me a star	60
Buy our products	60
Sponsor	61
Sponsor via wechat	61
Sponsor list	61



Figure 1: Linux Lab Logo

Subscribe Wechat :



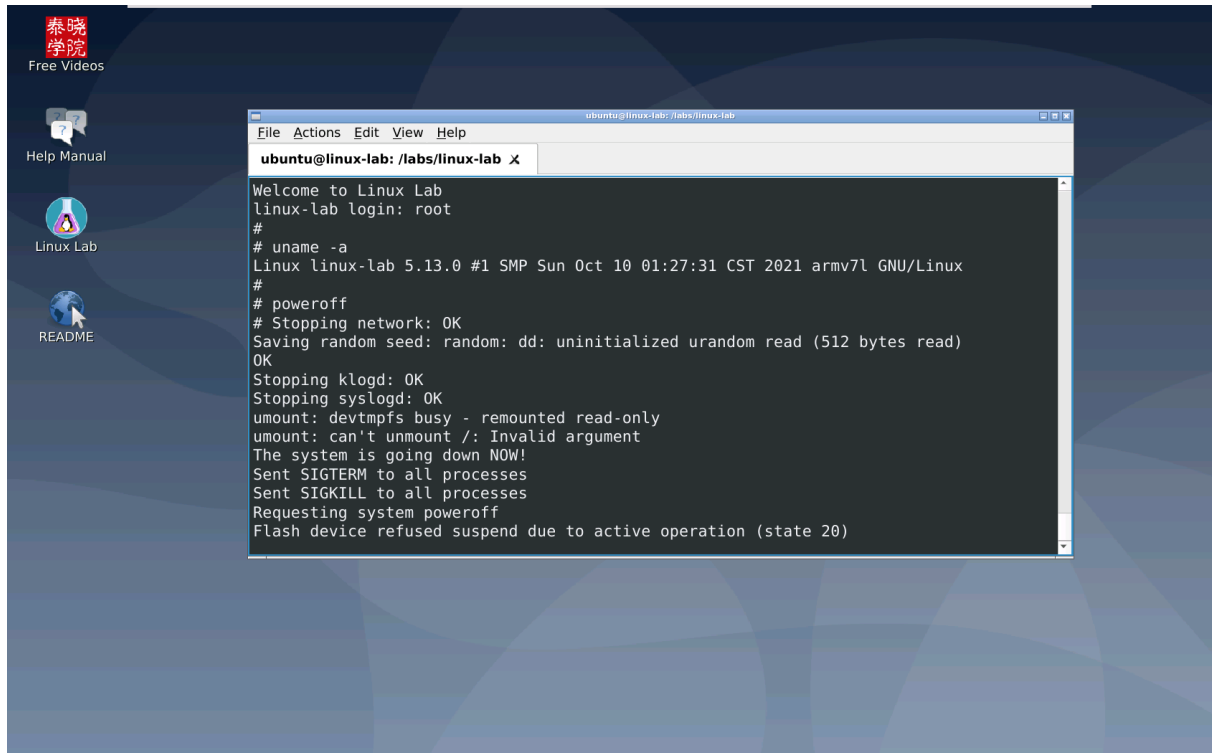
Figure 2: Wechat Public

1. Linux Lab Overview

1.1 Project Introduction

This project aims to create a Docker and Qemu based Linux development Lab to easier the learning, development and testing of [Linux Kernel](#).

Linux Lab is open source with no warranty – use at your own risk.



1.2 Project Homepage

- Homepage
 - <https://tinylab.org/linux-lab/>
 - <https://oschina.net/p/linux-lab>
- Repository
 - <https://gitee.com/tinylab/linux-lab>
 - <https://github.com/tinyclub/linux-lab>

Related Projects:

- Cloud Lab

- Linux Lab Running Environment Manager, provide GUI and CLI, support local and remote login
- <https://tinylab.org/cloud-lab>
- Linux 0.11 Lab
 - Learning Linux 0.11, only available in Linux Lab Disk from now on
 - Download it to `labs/linux-0.11-lab` and use it in Linux Lab directly
 - <https://tinylab.org/linux-0.11-lab>
- CS630 Qemu Lab
 - Learning X86 Linux Assembly, only available in Linux Lab Disk from now on
 - Download it to `labs/cs630-qemu-lab` and use it in Linux Lab directly
 - <https://tinylab.org/cs630-qemu-lab>

1.3 Demonstration

1.3.1 Free Video Courses In Chinese

- [Linux Lab Open Videos](#)
 - Linux Lab Introduction
 - Loongson Linux Development
 - Linux Lab Disk Demonstration
 - Linux Lab Release Meeting Replay Videos
 - Rust For Linux Introduction

1.3.2 Non-Free Video Courses In Chinese

- [The Perspective of Linux ELF](#)
 - Learn Linux ELF by practice, with hundreds of examples, all verified in Linux Lab
- [《Rust Language Quickstart》](#)
 - Rust course for C programmer, with examples verified in Linux Lab
- [《Software Reverse Engineering Quickstart》](#)
 - Learn reverse engineering by practice, with examples verified in Linux Lab
- [《Linux Kernel Livepatch Introduction》](#)
 - Learn Linux live patching in AArch64 by practice, with examples verified in Linux Lab

1.4 Project Functions

Now, Linux Lab becomes an intergrated Linux learning, development and testing environment, it supports:

Items	Description
Boards	Qemu based, 7+ main Architectures, 20+ popular boards; Several real boards supported
Components	Uboot, Linux / Modules, Buildroot, Qemu, Linux v0.11, v2.6.10 ~ 5.x supported
Prebuilt	All of above components have been prebuilt
Rootfs	Support include initrd, haddisk, mmc and nfs, Debian availab for ARM
Docker	Cross toolchains from gcc-4.3 available in one command, external ones configurable
Access	Accessible from local or remote, include CLI and GUI, support bash, ssh, vnc, web vnc
Network	Builtin bridge networking, every board has network (except Raspi3)
Boot	Support serial port, curses (bash/ssh friendly) and graphic booting
Testing	Support automatic testing via <code>make test</code> target
Debugging	debuggale via <code>make debug</code> target

Continue reading for more features and usage.

1.5 Project History

1.5.1 Project Origins

About 10 years ago (2010), a tinylinux proposal: [Work on Tiny Linux Kernel](#) accepted by Embedded Linux Foundation, therefore I have worked on this project for serveral months.

1.5.2 Problems Solved

During the project cycle, several scripts written to verify if the adding tiny features (e.g. [gc-sections](#)) breaks the other kernel features on the main cpu architectures.

These scripts uses qemu-system-ARCH as the cpu/board simulator, basic boot+function tests have been done for ftrace+perf, accordingly, defconfigs, rootfs, test scripts have been prepared, at that time, all of them were simply put in a directory, without a design or holistic consideration.

1.5.3 Project Born

They have slept in my harddisk for several years without any attention, until one day, docker and novnc came to my world, at first, [Linux 0.11 Lab](#) was born, after that, Linux Lab was designed to unify all of the above scripts, defconfigs, rootfs and test scripts.

2. Linux Lab Installation

Linux Lab uses Docker, if have already installed Docker and configured the best mirror site of docker images, it is very easy to install Linux Lab.

If really a Linux newbie or simply don't want to spend time on boring installation, buy the instant Linux Lab Disk:



It supports:

- Capacity
 - From 32G to 512G and even 1T, 2T
- Systems
 - Top6 Linux Distributions and even more based on your requirement
 - Include Ubuntu 18.04-21.04, Deepin 20.2+, Fedora 34+, Mint 20.2+, Ezgo 14.04+, Kali, Manjaro
- Features
 - Boot from any powered-off 64bit X86 Machine, include PC, Laptop and Mac-Book

- Boot from any running Windows, Linux and MacOS and run in parallel with them
- Switch from or to any running Windows, Linux without poweroff
- Support transparent compress, use 128G as ~256G capacity
- Support memory compiling, speedup compiling and save disk erase life
- Where to buy
 - [Taobao shop of TinyLab.org Community](#)

2.1 Hardware and Software Requirement

Linux Lab is a full embedded Linux development system, it needs enough calculation capacity and disk & memory storage space, to avoid potential extension issues, here is the recommended configuration:

Hardware	Requirement	Description
Processor	X86_64, > 1.5GHz	Must choose 64bit X86 while using virtual machine
Disk	>= 50G	System (25G), Docker Images(~5G), Linux Lab (20G)
Memory	>= 4G	Lower than 4G may have many unpredictable exceptions

If often use, please increase disk storage to 100G~200G, memory storage to 8G, cpu cores to 4 and above.

Currently, all of the X86_64 systems support Docker should be able to run Linux Lab, include Windows, Linux and MacOS, all of the popular Linux distributions may have been tried by different users.

Welcome to take a look at [the systems running Linux Lab](#) and share yours, for example:

```
1 $ cd /path/to/cloud-lab
2 $ tools/docker/env
3 System: Ubuntu 16.04.6 LTS
4 Linux: 4.4.0-176-generic
5 Docker: Docker version 18.09.4, build d14af54
```

2.2 Docker Installation

Docker is required by Linux Lab, please install it at first:

- Linux, Mac OSX, Windows 10
 - [Docker CE](#)

- older Windows (include some older Windows 10)

[Docker Toolbox](#); Install Ubuntu via Virtualbox or Vmware Virtual Machine

Before running Linux Lab, please refer to section 6.1.4 and make sure the following command works without sudo and without any issue:

```
1 $ docker run hello-world
```

In China, to use docker service normally, please **must** configure one of chinese docker mirror sites, for example:

- [Aliyun Docker Mirror Documentation](#)
 - For non Univerisity users, require login with freely registered account
- [USTC Docker Mirror Documentation](#)
 - For Univerisity users

More docker related issues, such as download slowly, download timeout and download errors, are cleary documented in the 6.1 section of FAQs.

The other issues, please read the [official docker docs](#).

Notes for Ubuntu Users - [doc/install/ubuntu-docker.md](#)

Notes for Arch Users - [doc/install/arch-docker.md](#)

Notes for Manjaro Users - [doc/install/manjaro-docker.md](#)

Notes for Windows Users:

- Please make sure your Windows version support docker: [Official Docker Documentation](#) and determine Docker Desktop or Docker Toolbox should be used
- Linux Lab only tested with ‘Git Bash’ in Windows, please must use with it
 - After installing [Git For Windows](#), “Git Bash Here” will come out in right-button press menu

2.3 Choose a working directory

Please simply choose one directory in `~/Downloads` OR `~/Documents` OR create a new `~/Develop` directory.

```
1 $ mkdir ~/Develop
2 $ cd ~/Develop
```

For Windows and Mac OSX, to compile Linux normally, please refer to section 5.7.1 and enable building cache:

2.4 Switch to normal user

Before downloading Linux Lab, please **MUST** switch to normal user.

Check who am i, 0 means root, non-zero means normal user:

```
1 $ id -u `whoami`  
2 1000
```

If current user is `root`, switch to a normal one:

```
1 # id -u `whoami`  
2 0  
3 # sudo -su <USER>
```

If no normal user exists, create new:

```
1 $ sudo useradd --create-home --shell /bin/bash --user-group --groups adm,sudo laber  
2 $ sudo passwd laber  
3 $ sudo -su laber  
4 $ whoami  
5 laber
```

2.5 Download the lab

Use Ubuntu system as an example:

Download cloud lab framework with normal user, pull images and checkout linux-lab repository:

```
1 $ git clone https://gitee.com/tinylab/cloud-lab.git  
2 $ cd cloud-lab/
```

If cloned source code with `root` account, please **MUST** switch to normal user and change their owner:

```
1 $ sudo -su <USER>  
2 $ sudo chown -R <USER>:<USER> -R cloud-lab/{*,.git}
```

2.6 Run and login the lab

Launch the lab and login with the user and password printed in the console:

```
1 $ tools/docker/run linux-lab
```

Login with Bash:

```
1 $ tools/docker/bash
```

Re-login the lab via web browser:

```
1 $ tools/docker/webvnc
```

The other login methods:

```
1 $ tools/docker/vnc
2 $ tools/docker/ssh
3 $ tools/docker/webssh
```

Choose one of the methods:

```
1 $ tools/docker/login list # List, choose and record
2 $ tools/docker/login vnc # Choose one directly and record for late login
```

Summary of login methods:

Login Method	Description	Default User	Where
bash	docker bash	ubuntu	localhost
ssh	normal ssh	ubuntu	localhost
vnc	normal vnc	ubuntu	localhost+VNC client
webvnc	web desktop	ubuntu	anywhere via internet
webssh	web ssh	ubuntu	anywhere via internet

Since vnc clients differs from operating systems, we use webvnc by default to make sure auto login vnc for all systems.

If really want to use local vnc clients, please install a vnc client, for example: `vinagre`, then specify it like this:

```
1 $ tools/docker/vnc vinagre
```

If the above command not work normally, based on the information printed above, please configure the vnc client yourself.

Notes: The directly connected ssh and vnc may not always work, please use one of the other three methods instead.

2.7 Update and rerun the lab

Usually, only need to update Linux Lab itself, to get the new boards support or related fixups:

```
1 $ cd /path/to/cloud-lab/labs/linux-lab/
2 $ git checkout master
3 $ git pull
```

Sometimes, need to update Cloud Lab, to fix up potential running issues or getting newer docker image:

```
1 $ cd /path/to/cloud-lab
2 $ git checkout master
3 $ git pull
```

If modified the running environment of Linux Lab locally and want to reuse it in the future, save the container (very slow, not recommend if not necessary):

```
1 $ tools/docker/save linux-lab
2 $ git checkout -- configs/linux-lab/docker/name
```

Then rerun linux lab:

```
1 $ tools/docker/rerun linux-lab
```

2.8 Quickstart: Boot a board

Get into the lab environment, switch directory:

```
1 $ cd /labs/linux-lab
```

Issue the following command to boot the prebuilt kernel and rootfs on the default `vexpress-a9` board:

```
1 $ make boot
```

Login as `root` user without password(password is empty), just input `root` and press Enter:

```
1 Welcome to Linux Lab
2
3 linux-lab login: root
4
5 # uname -a
6 Linux linux-lab 5.1.0 #3 SMP Thu May 30 08:44:37 UTC 2019 armv7l GNU/Linux
7 #
8 # poweroff
9 #
```

Shutdown the board with the `poweroff` command.

****Notes**:** If some boards not support `poweroff`, please press `CTRL+a x`. Of course, open another terminal and issue `kill` or `pkill` command also can quit `qemu`.

3. Linux Lab Kickstart

3.1 Using boards

3.1.1 List available boards

List builtin boards:

```

1 $ make list
2 [ aarch64/raspi3 ]:
3     ARCH      = arm64
4     CPU       ?= cortex-a53
5     LINUX    ?= v5.1
6     ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0
7     ROOTDEV ?= /dev/mmcblk0
8 [ aarch64/virt ]:
9     ARCH      = arm64
10    CPU       ?= cortex-a57
11    LINUX    ?= v5.1
12    ROOTDEV_LIST := /dev/sda /dev/vda /dev/ram0 /dev/nfs
13    ROOTDEV ?= /dev/vda
14 [ arm/mcimx6ul-evk ]:
15    ARCH      = arm
16    CPU       ?= cortex-a9
17    LINUX    ?= v5.4
18    ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0 /dev/nfs
19    ROOTDEV ?= /dev/mmcblk0
20 [ arm/versatilepb ]:
21    ARCH      = arm
22    CPU       ?= arm926t
23    LINUX    ?= v5.1
24    ROOTDEV_LIST := /dev/sda /dev/ram0 /dev/nfs
25    ROOTDEV ?= /dev/ram0
26 [ arm/vexpress-a9 ]:
27    ARCH      = arm
28    CPU       ?= cortex-a9
29    LINUX    ?= v5.1
30    ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0 /dev/nfs
31    ROOTDEV ?= /dev/ram0
32 [ i386/pc ]:
33    ARCH      = x86
34    CPU       ?= qemu32
35    LINUX    ?= v5.1
36    ROOTDEV_LIST ?= /dev/hda /dev/ram0 /dev/nfs
37    ROOTDEV_LIST[LINUX_v2.6.34.9] ?= /dev/sda /dev/ram0 /dev/nfs
38    ROOTDEV ?= /dev/hda
39 [ mips64el/ls2k ]:
40    ARCH      = mips
41    CPU       ?= mips64r2
42    LINUX    ?= loongnix-release-1903
43    LINUX[LINUX_loongnix-release-1903] := 04b98684
44    ROOTDEV_LIST := /dev/sda /dev/ram0 /dev/nfs
45    ROOTDEV ?= /dev/ram0
46 [ mips64el/ls3a7a ]:
47    ARCH      = mips
48    CPU       ?= mips64r2
49    LINUX    ?= loongnix-release-1903
50    LINUX[LINUX_loongnix-release-1903] := 04b98684
51    ROOTDEV_LIST ?= /dev/sda /dev/ram0 /dev/nfs

```

```
52     ROOTDEV ?= /dev/ram0
53 [ mipsel/ls1b ]:
54     ARCH      = mips
55     CPU       ?= mips32r2
56     LINUX    ?= v5.2
57     ROOTDEV_LIST ?= /dev/ram0 /dev/nfs
58     ROOTDEV ?= /dev/ram0
59 [ mipsel/ls232 ]:
60     ARCH      = mips
61     CPU       ?= mips32r2
62     LINUX    ?= v2.6.32-r190726
63     ROOTDEV_LIST := /dev/ram0 /dev/nfs
64     ROOTDEV ?= /dev/ram0
65 [ mipsel/malta ]:
66     ARCH      = mips
67     CPU       ?= mips32r2
68     LINUX    ?= v5.1
69     ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
70     ROOTDEV ?= /dev/ram0
71 [ ppc/g3beige ]:
72     ARCH      = powerpc
73     CPU       ?= generic
74     LINUX    ?= v5.1
75     ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
76     ROOTDEV ?= /dev/ram0
77 [ riscv32/virt ]:
78     ARCH      = riscv
79     CPU       ?= any
80     LINUX    ?= v5.0.13
81     ROOTDEV_LIST := /dev/vda /dev/ram0 /dev/nfs
82     ROOTDEV ?= /dev/vda
83 [ riscv64/virt ]:
84     ARCH      = riscv
85     CPU       ?= any
86     LINUX    ?= v5.1
87     ROOTDEV_LIST := /dev/vda /dev/ram0 /dev/nfs
88     ROOTDEV ?= /dev/vda
89 [ x86_64/pc ]:
90     ARCH      = x86
91     CPU       ?= qemu64
92     LINUX    ?= v5.1
93     ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
94     ROOTDEV_LIST[LINUX_v3.2] := /dev/sda /dev/ram0 /dev/nfs
95     ROOTDEV ?= /dev/ram0
96 [ csky/virt ]:
97     ARCH      = csky
98     CPU       ?= ck810
99     LINUX    ?= v4.9.56
100    ROOTDEV ?= /dev/nfs
```

ARCH, FILTER arguments are supported:

```
1 $ make list ARCH=arm
2 $ make list FILTER=virt
```

and more:

```
1 $ make list-board      # only ARCH
2 $ make list-short     # ARCH and LINUX
3 $ make list-base      # no plugin
```

```
4 $ make list-plugin      # only plugin
5 $ make list-full       # everything
6 $ make list-real       # real hardware boards
7 $ make list-virt       # only virtual boards
```

3.1.2 Choosing a board

3.1.2.1 Real board

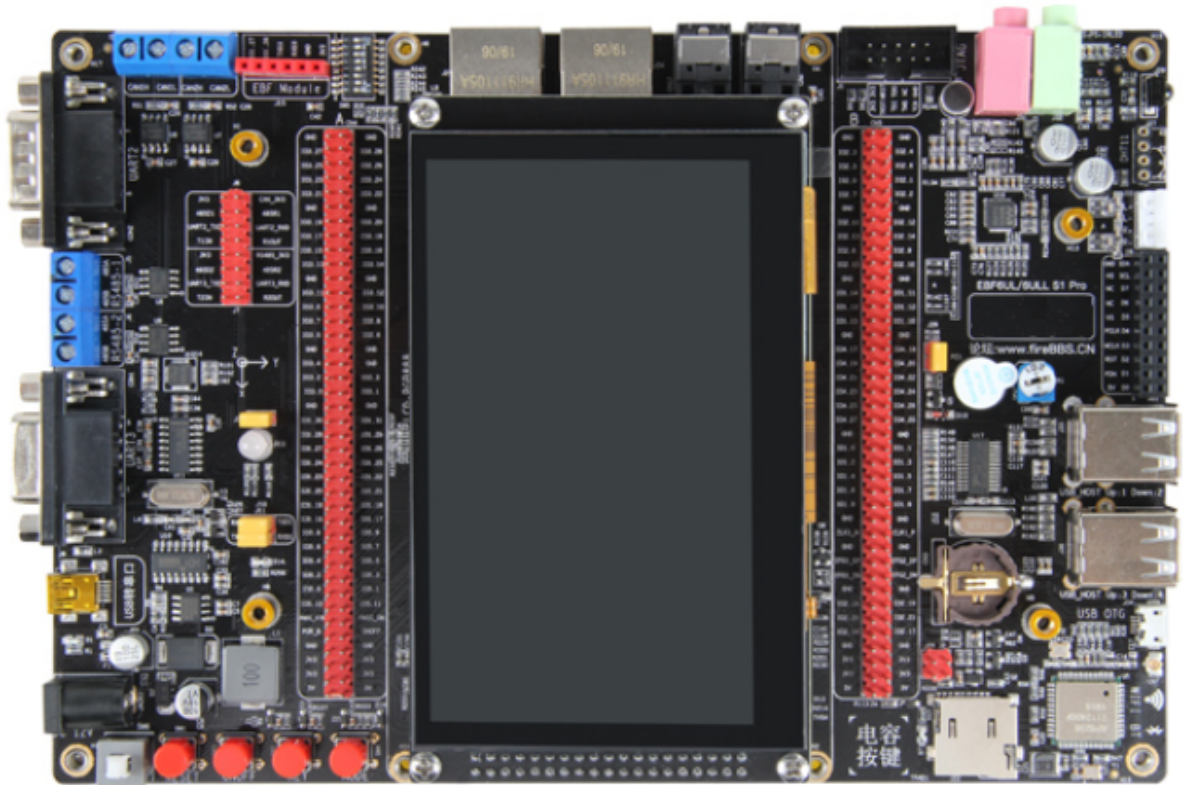
From version v0.6, to support learn external devices, Linux Lab adds real hardware board support, to use such boards, please buy them and connect them to your develop host correctly.

Only list real boards:

```
1 $ make list-real
2 [ arm/ebf-imx6ull ]:
3   ARCH      = arm
4   CPU       ?= cortex-a9
5   LINUX     ?= v4.19.35
6   ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0 /dev/nfs
7   ROOTDEV   ?= /dev/mmcblk0
```

Because real hardware boards differs from each other, so, board specific document are recommended, for example: `boards/arm/ebf-imx6ull/README.md`.

All supported real hardware boards and the related Linux Lab Disk will be put in [TinyLab.org's Taobao Shop](https://tinylab.org/taobao), after bought them, please contact with wechat: `tinylab` and join in the development group.



3.1.2.2 Virtual board

By default, the default virtual board: `vexpress-a9` is used, we can configure, build and boot for a specific board with `BOARD`, for example:

```
1 $ make BOARD=malta
2 $ make boot
```

If several boards have the same name, please specify the architecture to distinguish:

```
1 $ make BOARD=mipsel/malta
```

Currently, such boards have the same name:

```
1 $ make list FILTER=virt
2 [ aarch64/virt ]:
3     ARCH      = arm64
4     CPU       ?= cortex-a57
5     LINUX     ?= v5.1
6     ROOTDEV_LIST := /dev/sda /dev/vda /dev/ram0 /dev/nfs
7     ROOTDEV   ?= /dev/vda
8 [ riscv32/virt ]:
9     ARCH      = riscv
10    CPU       ?= any
11    LINUX     ?= v5.0.13
12    ROOTDEV_LIST := /dev/vda /dev/ram0 /dev/nfs
13    ROOTDEV   ?= /dev/vda
14 [ riscv64/virt ]:
15    ARCH      = riscv
16    CPU       ?= any
17    LINUX     ?= v5.1
```

```
18     ROOTDEV_LIST := /dev/vda /dev/ram0 /dev/nfs
19     ROOTDEV ?= /dev/vda
20
21 $ make list FILTER=/pc
22 [ i386/pc ]:
23     ARCH      = x86
24     CPU       ?= qemu32
25     LINUX     ?= v5.1
26     ROOTDEV_LIST ?= /dev/hda /dev/ram0 /dev/nfs
27     ROOTDEV_LIST[LINUX_v2.6.34.9] ?= /dev/sda /dev/ram0 /dev/nfs
28     ROOTDEV ?= /dev/hda
29 [ x86_64/pc ]:
30     ARCH      = x86
31     CPU       ?= qemu64
32     LINUX     ?= v5.1
33     ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
34     ROOTDEV_LIST[LINUX_v3.2] := /dev/sda /dev/ram0 /dev/nfs
35     ROOTDEV ?= /dev/ram0
```

Use them like this:

```
1 $ make BOARD=x86_64/pc
2 $ make BOARD=riscv64/virt
```

If using `board`, it only works on-the-fly, the setting will not be saved, this is helpful to run multiple boards at the same and not to disrupt each other:

```
1 $ make board=malta boot
```

This allows to run multi boards in different terminals or background at the same time.

Check the board specific configuration:

```
1 $ cat boards/arm/vexpress-a9/Makefile
```

3.1.3 Using as plugins

The ‘Plugin’ feature is supported by Linux Lab, to allow boards being added and maintained in standalone git repositories. Standalone repository is very important to ensure Linux Lab itself not grow up big and big while more and more boards being added in.

Book examples or the boards with a whole new cpu architecture benefit from such feature a lot, for book examples may use many boards and a new cpu architecture may need require lots of new packages (such as cross toolchains and the architecture specific qemu system tool).

Here maintains the available plugins:

- [C-Sky Linux](#)
- [Loongson Linux](#)

The Loongson plugin has been merged into v5.0.

3.1.4 Configure boards

Every board has its own configuration, some can be changed on demand, for example, memory size, linux version, buildroot version, qemu version and the other external devices, such as serial port, network devices and so on.

The configure method is very simple, just edit it by referring to current values (`boards/<BOARD>/Makefile`), this command open local configuration (`boards/<BOARD>/labconfig`) via vim:

```
1 $ make edit
```

But please don't make a big change once, we often only need to tune linux version, this command is better for such case:

```
1 $ make list-linux
2 v4.12 v4.5.5 v5.0.10 [v5.1]
3 $ make config LINUX=v5.0.10
4 $ make list-linux
5 v4.12 v4.5.5 [v5.0.10] v5.1
```

If want to upstream your local changes, please use `board-edit` and `board-config`, otherwise, `edit` and `config` are preferable, for they will avoid conflicts while pulling remote updates.

3.2 Build in one command

v0.3+ version add target dependency by default, so, if want to compile a kernel, just run:

```
1 $ make kernel-build
2
3 Or
4
5 $ make build kernel
```

It will do everything required, of course, we still be able to run the targets explicitly.

And futher, with the timestamping support, finished targets will not be run again during the late operations, if still want, just clean the stamp and run it again:

```
1 $ make cleanstamp kernel-build
2 $ make kernel-build
3
4 Or
5
6 $ make force-kernel-build
```

To clean all of the stamp files:

```
1 $ make cleanstamp kernel
```

This function also support uboot, root and qemu.

3.3 Detailed Operations

3.3.1 Downloading

Download board specific package and the kernel, buildroot source code:

```
1 $ make source APP=bsp,kernel,root,uboot
2 Or
3 $ make source APP=all
4 Or
5 $ make source all
```

Download one by one:

```
1 $ make bsp-source
2 $ make kernel-source
3 $ make root-source
4 $ make uboot-source
5
6 Or
7
8 $ make source bsp
9 $ make source kernel
10 $ make source root
11 $ make source uboot
```

After v0.5, the source code are downloaded in `src/`, before, they are saved in the root directory of Linux Lab.

Notes: the source code will be downloaded to `build/src` when `CACHE_SRC` OR `ONESHOT` is configured to 1, please save or backup the data inside manually, otherwise, they will be lost after system poweroff.

3.3.2 Checking out

Checkout the target version of kernel and builroot:

```
1 $ make checkout APP=kernel,root
```

Checkout them one by one:

```
1 $ make kernel-checkout
2 $ make root-checkout
3
4 Or
5
6 $ make checkout kernel
7 $ make checkout root
```

If checkout not work due to local changes, save changes and run to get a clean environment:

```
1 $ make kernel-cleanup
2 $ make root-cleanup
3
4 Or
5
6 $ make cleanup kernel
7 $ make cleanup root
```

The same to qemu and uboot.

3.3.3 Patching

Apply available patches in `boards/<BOARD>/bsp/patch/linux` and `src/patch/linux/:`

```
1 $ make kernel-patch
2
3 Or
4
5 $ make patch kernel
```

3.3.4 Configuration

3.3.4.1 Default Configuration

Configure kernel and buildroot with defconfig:

```
1 $ make defconfig APP=kernel,root
```

Configure one by one, by default, use the defconfig in `boards/<BOARD>/bsp/:`

```
1 $ make kernel-defconfig
2 $ make root-defconfig
3
4 Or
5
6 $ make defconfig kernel
7 $ make defconfig root
```

Configure with specified defconfig:

```
1 $ make B=raspi3
2 $ make kernel-defconfig bcmrpi3_defconfig
3 $ make root-defconfig raspberrypi3_64_defconfig
```

If only defconfig name specified, search `boards/` at first, and then the default configs path of buildroot, u-boot and linux-stable respectively: `src/buildroot/configs`, `src/u-boot/configs`, `src/linux-stable/arch//configs`.

3.3.4.2 Manual Configuration

```
1 $ make kernel-menuconfig
2 $ make root-menuconfig
3
4 Or
5
6 $ make menuconfig kernel
7 $ make menuconfig root
```

3.3.4.3 Old default configuration

```
1 $ make kernel-olddefconfig
2 $ make root-olddefconfig
3 $ make uboot-olddefconfig
4
5 Or
6
7 $ make olddefconfig kernel
8 $ make olddefconfig root
9 $ make olddefconfig uboot
```

3.3.5 Building

Build kernel and buildroot together:

```
1 $ make build APP=kernel,root
```

Build them one by one:

```
1 $ make kernel-build # make kernel
2 $ make root-build # make root
3
4 Or
5
6 $ make build kernel
7 $ make build root
```

After v0.5, the building result are stored in `build/`, before they are put in `output/`.

3.3.6 Saving

Save all of the configs and rootfs/kernel/dtb images:

```
1 $ make save APP=kernel,root
2 $ make saveconfig APP=kernel,root
```

Save configs and images to `boards/<BOARD>/bsp/`:

```
1 $ make kernel-saveconfig
2 $ make root-saveconfig
3 $ make root-save
4 $ make kernel-save
```

```
5
6 Or
7
8 $ make saveconfig kernel
9 $ make saveconfig root
10 $ make save kernel
11 $ make save root
```

3.3.7 Booting

Boot with serial port (nographic) by default, exit with CTRL+a x, poweroff, reboot OR pkill qemu (See [poweroff hang](#)):

```
1 $ make boot
```

Boot with graphic (Exit with CTRL+ALT+2 quit):

```
1 $ make b=pc boot G=1 LINUX=v5.1 BUILDROOT=2019.11
2 $ make b=versatilepb boot G=1 LINUX=v5.1 BUILDROOT=2016.05
3 $ make b=g3beige boot G=1 LINUX=v5.1 BUILDROOT=2016.05
4 $ make b=malta boot G=1 LINUX=v2.6.36 BUILDROOT=2016.05
5 $ make b=vexpress-a9 boot G=1 LINUX=v4.6.7 BUILDROOT=2016.05 // LINUX=v3.18.39 works too
```

Note:

- real graphic boot require LCD and keyboard drivers, the above boards work well, with linux v5.1, raspib3 and malta has tty0 console but without keyboard input.
- new buildroot config files set tty console to serial with (BR2_TARGET_GENERIC_GETTY_PORT="ttyAMA0"), to enable console with G=1, please change the getty line in /etc/inittab, for example, replace ttyAMA0 with console, we can also simply switch to the serial console via the Qemu 'View' menu.

vexpress-a9 and virt has no LCD support by default, but for the latest qemu, it is able to boot with G=1 and switch to serial console via the 'View' menu, this can not be used to test LCD and keyboard drivers. xopts specify the eXtra qemu options.

```
1 $ make b=vexpress-a9 CONSOLE=ttyAMA0 boot G=1 LINUX=v5.1
2 $ make b=raspib3 CONSOLE=ttyAMA0 XOPTS="-serial vc -serial vc" boot G=1 LINUX=v5.1
```

Boot with curses graphic (friendly to bash/ssh login, not work for all boards, exit with ESC+2 quit OR ALT+2 quit):

```
1 $ make b=pc boot G=2 LINUX=v4.6.7
```

Boot with PreBuilt Kernel, Dtb and Rootfs:

```
1 $ make boot kernel=old dtb=old root=old
```

Boot with new kernel, dtb and rootfs if exists:

```
1 $ make boot kernel=new dtb=new root=new
```

Boot with new kernel and uboot, build them if not exists:

```
1 $ make boot BUILD=kernel,uboot
```

Boot without Uboot (only versatilepb and vexpress-a9 boards tested):

```
1 $ make boot U=0
```

Boot with different rootfs (depends on board, check /dev/ after boot):

```
1 $ make boot ROOTDEV=ram // support by all boards, basic boot method
2 $ make boot ROOTDEV=nfs // depends on network driver, only raspi3 not work
3 $ make boot ROOTDEV=sda
4 $ make boot ROOTDEV=mmcblk0
5 $ make boot ROOTDEV=vda // virtio based block device
```

Boot with extra kernel command line (XKCLI = eXtra Kernel Command Line):

```
1 $ make boot ROOTDEV=nfs XKCLI="init=/bin/bash"
```

List supported options:

```
1 $ make list ROOTDEV
2 $ make list BOOTDEV
3 $ make list CCORI
4 $ make list NETDEV
5 $ make list LINUX
6 $ make list UBOOT
7 $ make list QEMU
```

And more <xxx>-list are also supported with list <xxx>, for example:

```
1 $ make list features
2 $ make list modules
3 $ make list gcc
```

4. Linux Lab Advance

4.1 Using Linux Kernel

4.1.1 non-interactive configuration

A tool named `scripts/config` in linux kernel is helpful to get/set the kernel config options non-interactively, based on it, both of `kernel-getconfig` and `kernel-setconfig` are added to tune the kernel options, with them, we can simply “enable/disable/setstr/setval/getstate” of a kernel option or many at the same time:

Get state of a kernel module:

```
1 $ make kernel-getconfig m=minix_fs
2 Getting kernel config: MINIX_FS ...
3
4 build/aarch64/linux-v5.1-virt/.config:CONFIG_MINIX_FS=m
```

Enable a kernel module:

```
1 $ make kernel-setconfig m=minix_fs
2 Setting kernel config: m=minix_fs ...
3
4 build/aarch64/linux-v5.1-virt/.config:CONFIG_MINIX_FS=m
5
6 Enable new kernel config: minix_fs ...
```

More control commands of `kernel-setconfig` including `y`, `n`, `c`, `o`, `s`, `v`:

Option	Description
<code>y</code>	build the modules in kernel or enable another kernel options.
<code>c</code>	build the modules as pluginable modules, just like <code>m</code> .
<code>o</code>	build the modules as pluginable modules, just like <code>m</code> .
<code>n</code>	disable a kernel option.
<code>s</code>	<code>RTC_SYSTOHC_DEVICE="rtc0"</code> , set the rtc device to <code>rtc0</code>
<code>v</code>	<code>PANIC_TIMEOUT=5</code> , set the kernel panic timeout to 5 secs.

Operates many options in one command line:

```
1 $ make kernel-setconfig m=tun,minix_fs y=ikconfig v=panic_timeout=5 s=DEFAULT_HOSTNAME=
   linux-lab n=debug_info
2 $ make kernel-getconfig o=tun,minix,ikconfig,panic_timeout,hostname
```

4.1.2 using kernel modules

Build all internal kernel modules:

```
1 $ make modules
2 $ make modules-install
3 $ make root-rebuild // not need for nfs boot
4 $ make boot
```

List available modules in `src/modules/`, `boards/<BOARD>/bsp/modules/`:

```
1 $ make modules-list
```

If `m` argument specified, list available modules in `src/modules/`, `boards/<BOARD>/bsp/modules/` and `src/linux-stable/`:

```
1 $ make modules-list m=hello
2     1 m=hello ; M=$PWD/src/modules/hello
3 $ make modules-list m=tun,minix
4     1 c=TUN ; m=tun ; M=drivers/net
5     2 c=MINIX_FS ; m=minix ; M=fs/minix
```

Enable one kernel module:

```
1 $ make kernel-getconfig m=minix_fs
2 Getting kernel config: MINIX_FS ...
3
4 build/aarch64/linux-v5.1-virt/.config:CONFIG_MINIX_FS=m
5
6 $ make kernel-setconfig m=minix_fs
7 Setting kernel config: m=minix_fs ...
8
9 build/aarch64/linux-v5.1-virt/.config:CONFIG_MINIX_FS=m
10
11 Enable new kernel config: minix_fs ...
```

Build one kernel module (e.g. `minix.ko`):

```
1 $ make modules M=fs/minix/
2 Or
3 $ make modules m=minix
```

Install and clean the module:

```
1 $ make modules-install M=fs/minix/
2 $ make modules-clean M=fs/minix/
```

More flexible usage:

```
1 $ make kernel-setconfig m=tun
2 $ make kernel x=tun.ko M=drivers/net
3 $ make kernel x=drivers/net/tun.ko
4 $ make do kernel drivers/net/tun.ko
```

Build external kernel modules (the same as internal modules):

```
1 $ make modules m=hello
2 Or
3 $ make kernel x=$PWD/modules/hello/hello.ko
```

4.1.3 using kernel features

4.1.3.1 list supported kernel features

Kernel features are abstracted in 'src/feature/linux/', including their configurations patchset, it can be used to manage both of the out-of-mainline and in-mainline features.

```
1 $ make feature-list
2 [ /labs/linux-lab/src/feature/linux ]:
3 + 9pnet
4 + core
5   - debug
6   - module
7 + ftrace
8   - v2.6.36
9     * env.g3beige
10    * env.malta
11    * env.pc
12    * env.versatilepb
13   - v2.6.37
14    * env.g3beige
15 + gcs
16   - v2.6.36
17     * env.g3beige
18     * env.malta
19     * env.pc
20     * env.versatilepb
21 + kft
22   - v2.6.36
23     * env.malta
24     * env.pc
25 + uksm
26   - v2.6.38
```

Verified boards and linux versions are recorded there, so, it should work without any issue if the environment not changed.

4.1.3.2 using kernel modules

For example, to enable kernel modules support, simply do:

```
1 // only upper case 'FEATURE' will be saved
2 $ make feature FEATURE=module
3 $ make kernel-olddefconfig
4 $ make kernel
```

4.1.3.3 using rust feature

Use x86_64/pc as an example :

```
1 $ make BOARD=x86_64/pc
```

switch to v5.13 Linux, because the current latest rust patchset can only be applied to v5.13:

```
1 $ make config LINUX=v5.13
```

Compile the kernel, and test it with one of the simplest module - rust_print:

```
1 // clean up everything for a whole new test
2 $ make kernel-cleanall
3
4 // this 'f' variable will not be saved for standalone make targets
5 $ make test f=rust m=rust_print
```

4.1.3.4 using kft feature

For kft feature in v2.6.36 for malta board:

```
1 $ make cleanall b=malta
2 $ make test b=malta f=kft LINUX=v2.6.36
```

4.1.3.5 using rt feature

Linux officially provide RT Preemption support, but many patches are outside of mainline kernel, to use it:

```
1 $ make feature-list f=rt
2 $ make test b=i386/pc f=rt LINUX=v5.2
```

4.1.3.6 persist or clear feature setting

Clear feature setting (reset feature saved in .labconfig):

```
1 $ make feature FEATURE=rust
2 $ make feature FEATURE=
```

The above function is the same as ‘make config’.

4.1.4 Create new development branch

If want to use a new development branch, please follow such steps:

At first, Get into `src/linux-stable` or another directory specified with `KERNEL_SRC`, checkout a development branch from a specific version:

```
1 $ cd src/linux-stable
2 $ git checkout -b linux-v5.1-dev v5.1
```

And then, clone the necessary configurations and directories for our new branch.

```
1 $ make kernel-clone LINUX=v5.1 LINUX_NEW=linux-v5.1-dev
```

The v5.1 must be the already supported version, if not, please use the near one in supported list, for example, i386/pc board support such versions:

```
1 $ make b=i386/pc list linux
2 v2.6.10 v2.6.11.12 v2.6.12.6 v2.6.21.5 v2.6.24.7 v2.6.34.9 v2.6.35.14 v2.6.36 v4.6.7 [v5
  .1] v5.2
```

If want to develop v2.6.38, please try to clone one from v2.6.36:

```
1 $ cd src/linux-stable
2 $ git checkout -b linux-v2.6.38-dev v2.6.38
3 $ make kernel-clone LINUX=v2.6.36 LINUX_NEW=linux-v2.6.38-dev
```

In development, please commit asap, and also, please use such commands carefully to avoid destroy your important changes:

- kernel-checkout, checkout a specified kernel version, may override your changes
- kernel-cleanup, clean up git repository, may remove your changes
- kernel-clean, clean building history
- kernel-cleanall, clean both of the building history and the source code changes

4.1.5 Use standalone git repository

v0.8 starts to add `KERNEL_FORK`, allows to configure the third party Linux source code repository, has added openEuler and wsl2, both of them support `x86_64/pc` and the former support `aarch64/virt` too.

For example, to compile wsl2 kernel, switch `KERNEL_FORK` to wsl2 directly:

```
1 $ make BOARD=x86_64/pc
2 $ make config KERNEL_FORK=wsl2
3 $ make kernel
```

To configure the wsl2 kernel version, configure it as following:

```
1 $ make edit
2 LINUX[KERNEL_FORK_wsl2] := linux-msft-wsl-5.10.74.3
```

The value should be one of the available tag in `git tag` list.

4.2 Using Uboot Bootloader

Choose one of the tested boards: `versatilepb` and `vexpress-a9`.

```
1 $ make BOARD=vexpress-a9
```


Download Uboot:

```
1 $ make uboot-source
```

Checkout the specified version:

```
1 $ make uboot-checkout
```

Patching with necessary changes, `BOOTDEV` and `ROOTDEV` available, use `flash` by default.

```
1 $ make uboot-patch
```

Use `tftp`, `sdcard` or `flash` explicitly, should run `make uboot-checkout` before a new `uboot-patch`:

```
1 $ make uboot-patch BOOTDEV=tftp
2 $ make uboot-patch BOOTDEV=sdcard
3 $ make uboot-patch BOOTDEV=flash
```

`BOOTDEV` is used to specify where to store and load the images for uboot, `ROOTDEV` is used to tell kernel where to load the rootfs.

Configure:

```
1 $ make uboot-defconfig
2 $ make uboot-menuconfig
```

Building:

```
1 $ make uboot
```

Boot with `BOOTDEV` and `ROOTDEV`, use `flash` by default:

```
1 $ make boot U=1
```

Use `tftp`, `sdcard` or `flash` explicitly:

```
1 $ make boot U=1 BOOTDEV=tftp
2 $ make boot U=1 BOOTDEV=sdcard
3 $ make boot U=1 BOOTDEV=flash
```

We can also change `ROOTDEV` during boot, for example:

```
1 $ make boot U=1 BOOTDEV=flash ROOTDEV=nfs
```

Clean images if want to update ramdisk, dtb and uImage:

```
1 $ make uboot-images-clean
2 $ make uboot-clean
```

Save uboot images and configs:

```
1 $ make uboot-save
2 $ make uboot-saveconfig
```

4.3 Using Qemu Emulator

Builtin qemu may not work with the newest linux kernel, so, we need compile and add external prebuilt qemu, this has been tested on vexpress-a9 and virt board.

At first, build qemu-system-ARCH:

```
1 $ make B=vexpress-a9
2 $ make qemu
3 $ make qemu-save
```

qemu-ARCH-static and qemu-system-ARCH can not be compiled together. to build qemu-ARCH-static, please enable `QEMU_US=1` in board specific Makefile and rebuild it.

If QEMU and QTOOL specified, the one in bsp submodule will be used in advance of one installed in system, but the first used is the one just compiled if exists.

While porting to newer kernel, Linux 5.0 hangs during boot on qemu 2.5, after compiling a newer qemu 2.12.0, no hang exists. please take notice of such issue in the future kernel upgrade.

If already download qemu and its submodules and don't want to upadte the submodules, just skip it:

```
1 $ make qemu git_module_status=0
```

4.4 Using Toolchains

The pace of Linux mainline is very fast, builtin toolchains can not keep up, to reduce the maintaining pressure, external toolchain feature is added. for example, ARM64/virt, CCVER and CCPATH has been added for it.

List available prebuilt toolchains:

```
1 $ make gcc-list
```

Download, decompress and enable the external toolchain:

```
1 $ make gcc
```

Switch compiler version if exists, for example:

```
1 $ make gcc-switch CCORI=internal GCC=4.7
2
3 $ make gcc-switch CCORI=linaro
```

If not external toolchain there, the builtin will be used back.

If no builtin toolchain exists, please must use this external toolchain feature, currently, aarch64, arm, riscv, mipsel, ppc, i386, x86_64 support such feature.

GCC version can be configured in board specific Makefile for Linux, Uboot, Qemu and Root, for example:

```
1 GCC[LINUX_v2.6.11.12] = 4.4
```

With this configuration, GCC will be switched automatically during defconfig and compiling of the specified Linux v2.6.11.12.

To build host tools, host gcc should be configured too (please specify b=i386/pc explicitly):

```
1 $ make gcc-list b=i386/pc
2 $ make gcc-switch CCOR1=internal GCC=4.8 b=i386/pc
```

4.5 Using Rootfs

Builtin rootfs is minimal, is not enough for complex application development, which requires modern Linux distributions.

Such a type of rootfs has been introduced and has been released as docker image, ubuntu 18.04 is added for arm32v7 at first, more later.

Run it via docker directly:

```
1 $ docker run -it tinylab/arm32v7-ubuntu
```

Extract it out and run in Linux Lab:

```
1 (host)$ sudo apt-get install -y qemu-user-static
```

ARM32/vexpress-a9 (user: root, password: root):

```
1 (host)$ tools/root/docker/extract.sh tinylab/arm32v7-ubuntu arm
2 (lab )$ make boot b=arm/vexpress-a9 U=0 V=1 MEM=1024M ROOTDEV=nfs ROOTFS=$PWD/prebuilt/
    fullroot/tmp/tinylab-arm32v7-ubuntu
```

ARM64/raspi3 (user: root, password: root):

```
1 (host)$ tools/root/docker/extract.sh tinylab/arm64v8-ubuntu arm
2 (lab )$ make boot b=aarch64/virt V=1 ROOTDEV=nfs ROOTFS=$PWD/prebuilt/fullroot/tmp/
    tinylab-arm64v8-ubuntu
```

More rootfs from docker can be found:

```
1 $ docker search arm64 | egrep "ubuntu|debian"
2 arm64v8/ubuntu   Ubuntu is a Debian-based Linux operating system  25
3 arm64v8/debian   Debian is a Linux distribution that's composed  20
```

4.6 Debugging Linux and Uboot

4.6.1 Debugging Linux

Compile the kernel with debugging options:

```
1 $ make feature FEATURE=debug
2 $ make kernel-olddefconfig
3 $ make kernel
```

Compile with one thread:

```
1 $ make kernel JOBS=1
```

And then debug it directly:

```
1 $ make debug
```

If login via `vnc` or `webvnc`, It will open a new terminal, load the scripts from `.gdb/kernel.default`, run `gdb` automatically.

But if login with `bash`, `ssh` or `webssh`, please read the prompt and run this command again to start debugging:

```
1 $ make debug
```

To customize kernel `gdbinit` script, simply copy one and edit it manually:

```
1 $ cp .gdb/kernel.default .gdb/kernel.user
```

It equals to:

```
1 $ make debug linux
```

to automate debug testing:

```
1 $ make test-debug linux
```

find out the code line of a kernel panic address:

```
1 $ make kernel-calltrace func+offset/length
```

if the debug port has been used, please try to find out who used the port and kill it:

```
1 $ sudo netstat -tlnp | grep 1234
2 tcp        0      0 0.0.0.0:1234          0.0.0.0:*            LISTEN      3943/qemu
   -xxx
3 $ sudo kill -9 3943
```

4.6.2 Debugging Uboot

To debug uboot with `.gdb/uboot.default`:

```
1 $ make debug uboot
```

If login with `vnc` or `webvnc`, the above command will open a terminal and start debugging automatically.

But if login with `bash`, `ssh` or `webssh`, please read the prompt and run this command again to start real debugging:

```
1 $ make debug uboot
```

To automate uboot debug testing:

```
1 $ make test-debug uboot
```

The same to kernel `gdbinit` script, customize one for uboot:

```
1 $ cp .gdb/uboot.default .gdb/uboot.user
```

4.7 Test Automation

Use `aarch64/virt` as the demo board here.

```
1 $ make BOARD=virt
```

Prepare for testing, install necessary files/scripts in `src/system/`:

```
1 $ make rootdir
2 $ make root-rebuild
```

Simply boot and poweroff (See [poweroff hang](#)):

```
1 $ make test
```

Don't poweroff after testing:

```
1 $ make test TEST_FINISH=echo
```

Run guest test case:

```
1 $ make test TEST_CASE=/tools/ftrace/trace.sh
```

Run guest test cases (`COMMAND_LINE_SIZE` must be big enough, e.g. 4096, see `cmdline_size` feature below):

```
1 $ make test TEST_BEGIN=date TEST_END=date TEST_CASE='ls /,echo hello world'
```

Reboot the guest system for several times:

```
1 $ make test TEST_REBOOT=2
```

NOTE: reboot may 1) hang, 2) continue; 3) timeout killed, TEST_TIMEOUT=30;
4) timeout continue, TIMEOUT_CONTINUE=1

Test a feature of a specified linux version on a specified board(`cmdline_size` feature is for increase `COMMAND_LINE_SIZE` to 4096):

```
1 $ make test f=kft LINUX=v2.6.36 b=malta TEST_PREPARE=board-init,kernel-cleanup
```

NOTE: `board-init` and `kernel-cleanup` make sure test run automatically, but `kernel-cleanup` is not safe, please save your code before use it!!

Test a kernel module:

```
1 $ make test m=hello
```

Test multiple kernel modules:

```
1 $ make test m=exception,hello
```

Test modules with specified ROOTDEV, `nfs` boot is used by default, but some boards may not support network:

```
1 $ make test m=hello,exception TEST_RD=ram0
```

Run test cases while testing kernel modules (test cases run between `insmod` and `rmmod`):

```
1 $ make test m=exception TEST_BEGIN=date TEST_END=date TEST_CASE='ls /root,echo hello world' TEST_PREPARE=board-init,kernel-cleanup f=cmdline_size
```

Run test cases while testing internal kernel modules:

```
1 $ make kernel-setconfig y=debug_fs
2 $ make test m=lkdtm TEST_BEGIN='mount -t debugfs debugfs /mnt' TEST_CASE='echo EXCEPTION ">" /mnt/provoke-crash/DIRECT'
```

Run test cases while testing internal kernel modules, pass kernel arguments:

```
1 $ make test m=lkdtm lkdtm_args='cpoint_name=DIRECT cpoint_type=EXCEPTION'
```

Run test without feature-init (save time if not necessary):

```
1 $ make test m=lkdtm lkdtm_args='cpoint_name=DIRECT cpoint_type=EXCEPTION' TEST_INIT=0
2 Or
3 $ make raw-test m=lkdtm lkdtm_args='cpoint_name=DIRECT cpoint_type=EXCEPTION'
```

Run test with module and the module's necessary dependencies (check with `make kernel-menuconfig`):

```
1 $ make test m=lkdtm y=runtime_testing_menu,debug_fs lkdtm_args='cpoint_name=DIRECT cpoint_type=EXCEPTION' LINUX=v5.1 TEST_PREPARE=kernel-cleanup
```

Run test without feature-init, boot-init, boot-finish and no TEST_PREPARE:

```
1 $ make boot-test m=lkdtm lkdtm_args='cpoint_name=DIRECT cpoint_type=EXCEPTION'
```

Test a kernel module and make some targets before testing:

```
1 $ make test m=exception TEST=kernel-checkout,kernel-patch,kernel-defconfig
```

Test everything in one command (from download to poweroff, see [poweroff hang](#)):

```
1 $ make test TEST=kernel,root TEST_PREPARE=board-init,kernel-cleanup,root-cleanup
```

Test everything in one command (with uboot while support, e.g. vexpress-a9):

```
1 $ make test TEST=kernel,root,uboot TEST_PREPARE=board-init,kernel-cleanup,root-cleanup,
  uboot-cleanup
```

Test kernel hang during boot, allow to specify a timeout, timeout must happen while system hang:

```
1 $ make test TEST_TIMEOUT=30s
```

Test kernel debug:

```
1 $ make test DEBUG=1
```

Notes: The above tests may fail on some boards with some Linux versions, please upgrade the kernel versions if necessary.

4.8 File Sharing

To transfer files between Qemu Board and Host, three methods are supported by default:

4.8.1 Install files to rootfs

Simply put the files with a relative path in `src/system/`, install and rebuild the rootfs:

```
1 $ mkdir src/system/root/
2 $ touch src/system/root/new_file
3 $ make root-rebuild
4 $ make boot
```

4.8.2 Share with NFS

Boot the board with `ROOTDEV=nfs`:

```
1 $ make boot ROOTDEV=nfs
```

Host:

```
1 $ make env-dump VAR=ROOTDIR
2 ROOTDIR="/labs/linux-lab/boards/<BOARD>/bsp/root/<BUILDRoot_VERSION>/rootfs"
```

4.8.3 Transfer via tftp

Using tftp server of host from the Qemu board with the `tftp` command.

Host:

```
1 $ ifconfig br0
2 inet addr:172.17.0.3 Bcast:172.17.255.255 Mask:255.255.0.0
3 $ cd tftpboot/
4 $ ls tftpboot
5 kft.patch kft.log
```

Qemu Board:

```
1 $ ls
2 kft_data.log
3 $ tftp -g -r kft.patch 172.17.0.3
4 $ tftp -p -r kft.log -l kft_data.log 172.17.0.3
```

Note: while put file from Qemu board to host, must create an empty file in host firstly. Buggy?

4.8.4 Share with 9p virtio

To enable 9p virtio for a new board, please refer to [qemu 9p setup](#). qemu must be compiled with `--enable-virtfs`, and kernel must enable the necessary options.

Reconfigure the kernel with:

```
1 CONFIG_NET_9P=y
2 CONFIG_NET_9P_VIRTIO=y
3 CONFIG_NET_9P_DEBUG=y (Optional)
4 CONFIG_9P_FS=y
5 CONFIG_9P_FS_POSIX_ACL=y
6 CONFIG_PCI=y
7 CONFIG_VIRTIO_PCI=y
8 CONFIG_PCI_HOST_GENERIC=y (only needed for the QEMU Arm 'virt' board)
```

If using `-virtfs` OR `-device virtio-9p-pci` option for qemu, must enable the above PCI related options, otherwise will not work:

```
1 9pnet_virtio: no channels available for device hostshare
2 mount: mounting hostshare on /hostshare failed: No such file or directory
```

`-device virtio-9p-device` requires less kernel options.

To enable the above options, please simply type:


```
1 $ make feature FEATURE=9pnet
2 $ make kernel-olddefconfig
```

Docker host:

```
1 $ modprobe 9pnet_virtio
2 $ lsmod | grep 9p
3 9pnet_virtio          17519  0
4 9pnet                 72068  1 9pnet_virtio
```

Host:

```
1 $ make BOARD=virt
2
3 $ make root-rebuild
4
5 $ touch hostshare/test      # Create a file in host
6
7 $ make boot U=0 ROOTDEV=ram0 PBR=1 SHARE=1
8
9 $ make boot SHARE=1 SHARE_DIR=src/modules # for external modules development
10
11 $ make boot SHARE=1 SHARE_DIR=build/aarch64/linux-v5.1-virt/ # for internal modules
    learning
12
13 $ make boot SHARE=1 SHARE_DIR=src/examples # for c/assembly learning
```

Qemu Board:

```
1 $ ls /hostshare/          # Access the file in guest
2 test
3 $ touch /hostshare/guest-test # Create a file in guest
```

Verified boards with Linux v5.1:

boards	Status
aarch64/virt	virtio-9p-device (virtio-9p-pci breaks nfsroot)
arm/vexpress-a9	only work with virtio-9p-device and without uboot booting
arm/versatilepb	only work with virtio-9p-pci
x86_64/pc	only work with virtio-9p-pci
i386/pc	only work with virtio-9p-pci
riscv64/virt	work with virtio-9p-pci and virtio-9p-dev
riscv32/virt	work with virtio-9p-pci and virtio-9p-dev

4.9 Learning Assembly

Linux Lab has added many assembly examples in `src/examples/assembly`:

```
1 $ cd src/examples/assembly
2 $ ls
3 aarch64 arm mips64el mipsel powerpc powerpc64 README.md x86 x86_64
4 $ make -s -C aarch64/
5 Hello, ARM64!
```

4.10 Learning C

4.10.1 Host build and Run

Use hello as example:

```
1 $ cd src/examples/c/hello
2 $ make
3 gcc -fno-stack-protector -fomit-frame-pointer -fno-asynchronous-unwind-tables -fno-pie -
   no-pie -m32 -Wall -Werror -g -o hello hello.c
4 Hello, World!
```

4.10.2 Cross build and Run

Use X32 (Code for x86-64, int/long/pointer to 32bits), ARM, MIPS, PPC and RISC-V as example:

```
1 $ sudo apt-get update -y
2
3 $ sudo apt-get install -y libc6-x32 libc6-dev-x32 libx32gcc-8-dev
4 $ gcc -mx32 -o hello hello.c
5 $ ./hello
6 Hello, World!
7
8 $ sudo apt-get install -y libc6-dev-armel-cross libc6-armel-cross
9 $ arm-linux-gnueabi-gcc -o hello hello.c
10 $ qemu-arm -L /usr/arm-linux-gnueabi/ ./hello
11 Hello, World!
12
13 $ sudo apt-get install -y libc6-dev-mipsel-cross libc6-mipsel-cross
14 $ mipsel-linux-gnu-gcc -o hello hello.c
15 $ qemu-mipsel -L /usr/mipsel-linux-gnu/ ./hello
16 Hello, World!
17
18 $ sudo apt-get install -y libc6-dev-powerpc-cross libc6-powerpc-cross
19 // Must use -static for Linux Lab v0.6, otherwise, there will be segmentation fault
20 $ powerpc-linux-gnu-gcc -static -o hello hello.c
21 $ qemu-ppc -L /usr/powerpc-linux-gnu/ ./hello
22 Hello, World!
23
24 $ sudo apt-get install -y libc6-dev-riscv64-cross libc6-dev-riscv64-cross
25 $ riscv64-linux-gnu-gcc -o hello hello.c
26 $ qemu-riscv64 -L /usr/riscv64-linux-gnu/ ./hello
27 Hello, World!
```

Above run through `qemu-user`, to run on target boards, please copy the binaries to target boards' rootfs with help from section 4.8.1.

The main packages are `libc6-dev`, `libc6` OR `libgcc`, but x32 is an exception, it is `libx32gcc`. please list them via `apt-cache search`.

4.11 Running any make goals

Linux Lab allows to access Makefile goals easily via the `do` target, for example:

```
1 $ make do kernel help
2 $ make do kernel menuconfig
3
4 $ make do root help
5 $ make do root busybox-menuconfig
6
7 $ make do uboot help
8 $ make do uboot menuconfig
```

The `do` goal allows to run sub-make goals of kernel, root and uboot directly without entering into their own building directory.

4.12 More Usage

Read more:

- Why
 - [Why Using Linux Lab V1.0 \(In Chinese\)](#)
 - [Why Using Linux Lab V2.0 \(In Chinese\)](#)
- User Manual
 - [Linux Lab v0.9 User Manual](#)
 - [Linux Lab v0.8 User Manual](#)
 - [Linux Lab Loongson Manual V0.2](#)
- Linux Lab Videos
 - [CCTALK](#)
 - [Bilibili](#)
 - [Zhihu](#)
- Video Courses use Linux Lab as experiment environment
 - [The Perspective Linux ELF](#)
 - [《Rust Language Quickstart》](#)
 - [《Software Reverse Engineering Quickstart》](#)
 - [《Linux Kernel Livepatch Introduction》](#)

- The books or courses Linux Lab supported or plan to support
 - [books or courses list](#)
- The boards Linux Lab supported or plan to support
 - [ARM IMX6ULL](#)
 - RISC-V-64 D1
- The hardwares developed by Linux Lab community
 - [Linux Lab Disk](#), pre-installed Linux Lab disk
 - * Support Ubuntu 18.04-21.04, Deepin 20.2+, Fedora 34+, Mint 20.2+, Ezgo 14.04+, Kali, Manjaro
 - [Pocket Linux Disk](#), pre-installed Linux distribution disk
 - * Support Ubuntu 18.04-21.04, Deepin 20.2+, Fedora 34+, Mint 20.2+, Ezgo 14.04+, Kali, Manjaro

5. Linux Lab Development

This introduces how to add a new board for Linux Lab.

5.1 Choose a board supported by qemu

list the boards, use arm as an example:

```
1 $ qemu-system-arm -M ?
```

5.2 Create the board directory

Use `vexpress-a9` as an example:

```
1 $ mkdir boards/arm/vexpress-a9/
```

5.3 Clone a Makefile from an existing board

Use `versatilepb` as an example:

```
1 $ cp boards/arm/versatilepb/Makefile boards/arm/vexpress-a9/Makefile
```

5.4 Configure the variables from scratch

Comment everything, add minimal ones and then others.

Please refer to `doc/qemu/qemu-doc.html` or the online one <https://www.qemu.org/docs/master/>.

5.5 At the same time, prepare the configs

We need to prepare the configs for linux, buildroot and even uboot.

Buildroot has provided many examples about buildroot and kernel configuration:

```
1 buildroot: src/buildroot/configs/qemu_ARCH_BOARD_defconfig
2 kernel: src/buildroot/board/qemu/ARCH-BOARD/linux-VERSION.config
```

Uboot has also provided many default configs:

```
1 uboot: src/u-boot/configs/vexpress_ca9x4_defconfig
```

Kernel itself also:

```
1 kernel: src/linux-stable/arch/arm/configs/vexpress_defconfig
```

Linux Lab itself also provide many working configs too, the `xxx-clone` target is a good helper to utilize existing configs:

```
1 $ make list kernel
2 v4.12 v5.0.10 v5.1
3 $ make kernel-clone LINUX=v5.1 LINUX_NEW=v5.4
4 $ make kernel-menuconfig
5 $ make kernel-saveconfig
6
7 $ make list root
8 2016.05 2019.02.2
9 $ make root-clone BUILDROOT=2019.02.2 BUILDROOT_NEW=2019.11
10 $ make root-menuconfig
11 $ make root-saveconfig
```

Edit the configs and Makefile until they match our requirements.

```
1 $ make kernel-menuconfig
2 $ make root-menuconfig
3 $ make board-edit
```

The configuration must be put in `boards/<BOARD>/` and named with necessary version info, use `raspi3` as an example:

```
1 $ make kernel-saveconfig
2 $ make root-saveconfig
3 $ ls boards/aarch64/raspi3/bsp/configs/
4 buildroot_2019.02.2_defconfig linux_v5.1_defconfig
```

2019.02.2 is the buildroot version, v5.1 is the kernel version, both of these variables should be configured in `boards/<BOARD>/Makefile`.

More usage about the `xxx-clone` commands:

```
1 $ make qemu-clone QEMU=<old_version> QEMU_NEW=<new_version>
2 $ make uboot-clone UBOOT=<old_version> UBOOT_NEW=<new_version>
3 $ make kernel-clone LINUX=<old_version> LINUX_NEW=<new_version>
4 $ make root-clone BUILDROOT=<old_version> BUILDROOT_NEW=<new_version>
```

5.6 Choose the versions of kernel, rootfs and uboot

Please use `tag` instead of `branch`, use `kernel` as an example:

```
1 $ cd src/linux-stable
2 $ git tag
3 ...
4 v5.0
5 ...
6 v5.1
7 ..
8 v5.1.1
9 v5.1.5
10 ...
```

If want v5.1 kernel, just put a line “LINUX = v5.1” in boards/<BOARD>/Makefile.

Or clone a kernel config from the old one or the official defconfig:

```
1 $ make kernel-clone LINUX_NEW=v5.3 LINUX=v5.1
2
3 Or
4
5 $ make B=i386/pc
6 $ pushd linux-stable && git checkout v5.4 && popd
7 $ make kernel-clone LINUX_NEW=v5.4 KCFG=i386_defconfig
```

If no tag existed, a virtual tag name with the real commit number can be configured as following:

```
1 LINUX = v2.6.11.12
2 LINUX[LINUX_v2.6.11.12] = 8e63197f
```

Linux version specific ROOTFS are also supported:

```
1 ROOTFS[LINUX_v2.6.12.6] ?= $(BSP_ROOT)/$(BUILDR00T)/rootfs32.cpio.gz
```

5.7 Configure, build and boot them

Use kernel as an example:

```
1 $ make kernel-defconfig
2 $ make kernel-menuconfig
3 $ make kernel
4 $ make boot
```

The same to rootfs, uboot and even qemu.

5.7.1 Speed up compiling and save disk life

Notes: This operation may lose data, please take care!

This feature aims to create a ram based temporary filesystem as the ‘build’ directory, to store the building data, **If not backup them, they will be lost after shutting down the machine.**

Create temporary building cache:

```
1 $ make build cache
```

Check the status of building cache:

```
1 $ make build status
```

Use the cache for building speedup:

```
1 $ time make kernel
```

Backup the cache to a persistent file (If the building file are important to you):

```
1 $ make build backup
```

Stop the building cache, revert back to use the build directory on the disk:

```
1 $ make build uncache
```

Use the backup as the build directory:

```
1 $ sudo mount /path/to/backup-file /labs/linux-lab/build/
```

5.8 Save the images and configs

```
1 $ make root-save
2 $ make kernel-save
3 $ make uboot-save
4
5 $ make root-saveconfig
6 $ make kernel-saveconfig
7 $ make uboot-saveconfig
```

5.9 Upload everything

At last, upload the images, defconfigs, patchset to board specific bsp submodule repository.

Firstly, get the remote bsp repository address as following:

```
1 $ git remote show origin
2 * remote origin
3   Fetch URL: https://gitee.com/tinylab/qemu-aarch64-raspi3/
4   Push URL: https://gitee.com/tinylab/qemu-aarch64-raspi3/
5   HEAD branch: master
6   Remote branch:
7     master tracked
8   Local branch configured for 'git pull':
9     master merges with remote master
10  Local ref configured for 'git push':
11  master pushes to master (local out of date)
```

Then, fork this repository from gitee.com, upload your changes, and send your pull request.

6. FAQs

6.1 Docker Issues

6.1.1 Speed up docker images downloading

To optimize docker images download speed, please refer to section 6.1.6.

6.1.2 Docker network conflicts with LAN

We assume the docker network is 10.66.0.0/16, if not, we'd better change it as following:

```
1 $ sudo vim /etc/default/docker
2 DOCKER_OPTS="$DOCKER_OPTS --bip=10.66.0.10/16"
3
4 $ sudo vim /lib/systemd/system/docker.service
5 ExecStart=/usr/bin/dockerd -H fd:// --bip=10.66.0.10/16
```

Please restart docker service and lab container to make this change works:

```
1 $ sudo service docker restart
2 $ tools/docker/rerun linux-lab
```

If lab network still not work, please try another private network address and eventually to avoid conflicts with LAN address.

6.1.3 Why not allow running Linux Lab in local host

The full function of Linux Lab depends on the full docker environment managed by [Cloud Lab](#), so, please really never try and therefore please don't complain about why there are lots of packages missing failures and even the other weird issues.

Linux Lab is designed to use pre-installed environment with the docker technology and save our life by avoiding the packages installation issues in different systems, so, Linux Lab would never support local host using even in the future.

6.1.4 Run tools without sudo

To use the tools under `tools` without `sudo`, please make sure add your account to the docker group and reboot your system to take effect:

```
1 $ sudo usermod -aG docker <USER>
2 $ newgrp docker
```

Notes: Currently, root user is not allowed to run Linux Lab.

6.1.5 Network not work

If ping not work, please check one by one:

- DNS issue

if ping 8.8.8.8 work, please check `/etc/resolv.conf` and make sure it is the same as your host configuration.

- IP issue

if ping not work, please refer to [network conflict issue](#) and change the ip range of docker containers.

6.1.6 Client.Timeout exceeded while waiting headers

This means must configure one of the following docker mirror sites:

- [Aliyun Docker Mirror Documentation](#)
- [USTC Docker Mirror Documentation](#)

Potential methods of configuration in Ubuntu, depends on docker and ubuntu versions:

`/etc/default/docker:`

```
1 echo "DOCKER_OPTS=\"\${DOCKER_OPTS} --registry-mirror=<your accelerate address>\""
```

`/lib/systemd/system/docker.service:`

```
1 ExecStart=/usr/bin/dockerd -H fd:// --bip=10.66.0.10/16 --registry-mirror=<your  
accelerate address>
```

`/etc/docker/daemon.json:`

```
1 {  
2   "registry-mirrors": ["<your accelerate address>"]  
3 }
```

Please restart docker service after change the accelerate address:

```
1 $ sudo service docker restart
```

For the other Linux systems, Windows and MacOS System, please refer to [Aliyun Mirror Speedup Document](#).

IF still slow, please check if the mirror site is configured normally and without typos:

```
1 $ docker info | grep -A1 -i Mirrors  
2 Registry Mirrors:  
3 https://XXXXX.mirror.aliyuncs.com/
```

6.1.7 Restart Linux Lab after host system shutdown or reboot

If want to restore the installed softwares and related configurations, please save the container manually:

```
1 $ tools/docker/save linux-lab
```

After host system (include virtual machine) shutdown or reboot, you can restart the lab via the “Linux Lab” icon on the desktop, or just like before, issue this command:

```
1 $ tools/docker/run linux-lab
```

Current implementation doesn't support the direct 'docker start' command, please learn it.

If the above methods still not restart the lab, please refer to the methods mentioned in the 6.3.9 section.

If resume from a suspended host system, the lab will restore automatically, no need to do anything to restart it, just use one of the 4 login methods mentioned in the 2.4 section, for example, start a web browser to connect it:

```
1 $ tools/docker/webvnc
```

6.1.8 the following directives are specified both as a flag and in the configuration file

If getting such error:

```
1 unable to configure the Docker daemon with file /etc/docker/daemon.json: the
2 following directives are specified both as a flag and in the configuration
3 file: registry-mirrors: (from flag: [https://docker.mirrors.ustc.edu.cn/], from
4 file: [https://xxx.mirror.aliyuncs.com])
```

Means both `/etc/docker/daemon.json` and `/etc/default/docker` configured `registry-mirrors`, please comment the late one and restart docker:

```
1 $ sudo service docker restart
```

6.1.9 pathspec FETCH_HEAD did not match any file known to git

If get such error while running `make boot`, it means network issue, please refer to section 6.1.5.

```
1 Could not resolve host: gitee.com
2 error: pathspec 'FETCH_HEAD' did not match any file(s) known to git
```

6.1.10 Docker not work in Ubuntu 20.04

If docker not work in Ubuntu 20.04, please use `doc/install/daemon.json` and clean up the arguments of `dockerd`, learn more from [docker daemon](#):

```
1 $ sudo cat /etc/systemd/system/docker.service.d/docker.conf
2 [Service]
3 ExecStart=
4 ExecStart=/usr/bin/dockerd
5
6 $ sudo cp /etc/docker/daemon.json /etc/docker/daemon.json.bak
7 $ sudo cp doc/install/daemon.json /etc/docker/
8 $ sudo service docker restart
```

Please make sure using the best `registry-mirrors` for better download speed.

6.1.11 Error creating aufs mount

If not work with failure like “error creating aufs mount to ... invalid arguments”, that means the storage driver used by docker is not supported by current system, please choose another one from [this page](#), and configure it in `/etc/docker/daemon.json`, for example:

```
1 $ sudo vim /etc/docker/daemon.json
2 {
3   "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"],
4   "storage-driver": "devicemapper"
5 }
```

This issue is related to kernel version, the same system may upgrade kernel version and therefore support different storage driver.

6.2 Qemu Issues

6.2.1 Why kvm speeding up is disabled

kvm only supports both of `qemu-system-i386` and `qemu-system-x86_64` currently, and it also requires the cpu and bios support, otherwise, you may get this error log:

```
1 modprobe: ERROR: could not insert 'kvm_intel': Operation not supported
```

Check cpu virtualization support, if nothing output, then, cpu not support virtualization:

```
1 $ cat /proc/cpuinfo | egrep --color=always "vmx|svm"
```

If cpu supports, we also need to make sure it is enabled in bios features, simply reboot your computer, press ‘Delete’ to enter bios, please make sure the ‘Intel virtualization technology’ feature is ‘enabled’.

6.2.2 Poweroff hang

Both of the `poweroff` and `reboot` commands not work on these boards currently (LINUX=v5.1):

- mipsel/malta (exclude LINUX=v2.6.36)
- mipsel/ls232
- mipsel/ls1b
- mips64el/ls2k
- mips64el/ls3a7a
- aarch64/raspi3
- arm/versatilepb

System will directly hang there while running `poweroff` or `reboot`, to exit qemu, please pressing `CTRL+a x` or using `pkill qemu`.

To test such boards automatically, please make sure setting `TEST_TIMEOUT`, e.g. `make test TEST_TIMEOUT=50`.

Welcome to fix up them.

6.2.3 How to exit qemu

Where	How
Serial Port Console	<code>CTRL+a x</code>
Curses based Graphic	<code>ESC+2 quit</code> Or <code>ALT+2 quit</code>
X based Graphic	<code>CTRL+ALT+2 quit</code>
Generic Methods	<code>poweroff, reboot, kill, pkill</code>

6.2.4 Boot with missing sdl2 libraries failure

That's because the docker image is not updated, just enter into cloud-lab and rerun the lab (please must not use `tools/docker/restart` here for it not using the new docker image):

```
1 $ tools/docker/rerun linux-lab
```

6.3 Environment Issues

6.3.1 NFS/tftpboot not work

If nfs or tftpboot not work, please run `modprobe nfsd` in host side and restart the net services via `/configs/tools/restart-net-servers.sh` and please make sure not use `tools/docker/trun`.

6.3.2 How to switch windows in vim

`CTRL+w` is used in both of browser and vim, to switch from one window to another, please USE `CTRL+Left` OR `CTRL+Right` key instead, Linux Lab has remapped `CTRL+Right` to `CTRL+w` and `CTRL+Left` to `CTRL+p`.

6.3.3 How to delete typo in shell command line

Long keypress not work in novnc client currently, so, long `Delete` not work, please use `alt+delete` OR `alt+backspace` instead, more tips:

Function	Vim	Bash
begin/end	<code>~/\$</code>	<code>Ctrl + a/e</code>
forward/backward	<code>w/b</code>	<code>Ctrl + Home/end</code>
cut one word backward	<code>db</code>	<code>Alt + Delete/backspace</code>
cut one word forward	<code>dw</code>	<code>Alt + d</code>
cut all to begin	<code>d^</code>	<code>Ctrl + u</code>
cut all to end	<code>d\$</code>	<code>Ctrl + k</code>
paste all cutted	<code>p</code>	<code>Ctrl + y</code>

6.3.4 Language input switch shortcuts

In order to switch English/Chinese input method, please use `CTRL+s` shortcuts, it is used instead of `CTRL+space` to avoid conflicts with local system.

6.3.5 How to tune the screen size

There are tow methods to tune the screen size, one is auto scaling by noVNC, another is pre-setting during launching.

The first one is setting noVNC before connecting.

```
1 * Press the left sidebar of noVNC web page
2 * Disconnect
3 * Enable 'Auto Scaling Mode' via 'Settings -> Scaling Mode: -> Local Scaling -> Apply'
4 * Connect
```

The second one is setting `SCREEN_SIZE` while running Linux Lab.

The screen size of lab is captured by `xrandr`, if not work, please check and set your own, for example:

Get available screen size values:

```
1 $ xrandr --current
2 Screen 0: minimum 1 x 1, current 1916 x 891, maximum 16384 x 16384
3 Virtual1 connected primary 1916x891+0+0 (normal left inverted right x axis y axis) 0mm x
   0mm
4   1916x891      60.00*+
5   2560x1600     59.99
6   1920x1440     60.00
7   1856x1392     60.00
8   1792x1344     60.00
9   1920x1200     59.88
10  1600x1200     60.00
11  1680x1050     59.95
12  1400x1050     59.98
13  1280x1024     60.02
14  1440x900      59.89
15  1280x960      60.00
16  1360x768      60.02
17  1280x800      59.81
18  1152x864      75.00
19  1280x768      59.87
20  1024x768      60.00
21  800x600       60.32
22  640x480       59.94
```

Update remote screen size:

```
1 $ cd /path/to/cloud-lab
2 $ tools/docker/resize 1280x1024 # Specifiy anyone above
3 $ tools/docker/resize          # If no argument, Sync with host system
```

If want fullscreen, follow these steps:

1. If using virtual machine, fullscreen virtual machine at fist
2. Run `tools/docker/resize` to resize remote lab screen size
3. Enter into WebVNC Interface, Click the FullScreen button at the left sidebar

6.3.6 How to work in fullscreen mode

Open the left sidebar, press the 'Fullscreen' button.

6.3.7 How to record video

- Enable recording

Open the left sidebar, press the ‘Settings’ button, config ‘File/Title/Author/Category/Tags/Description’ and enable the ‘Record Screen’ option.

- Start recording

Press the ‘Connect’ button.

- Stop recording

Press the ‘Disconnect’ button.

- Replay recorded video

Press the ‘Play’ button.

- Share it

Videos are stored in ‘cloud-lab/recordings’, share it with help from showdesk.io.

6.3.8 Linux Lab not response

The VNC connection may hang for some unknown reasons and therefore Linux Lab may not response sometimes, to restore it, please press the flush button of web browser or re-connect after explicitly disconnect.

6.3.9 VNC login with failures

If VNC login return “Disconnect timeout”, wait a while and press the left ‘Connect’ button again, otherwise, check as following:

At first, check the containers’ status (Up: Ok, Exit: Bad):

```
1 $ docker ps -a
2 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
3 19a61ba075b5 tinylab/linux-lab "/tools/lab/run" 4 days ago Up 4 days 22/tcp, 5900/tcp
   linux-lab-21575
4 75dae89984c9 tinylab/cloud-ubuntu-web "/startup.sh" 8 days ago Up 8 days ...443/tcp
   cloud-ubuntu-web
```

If the status is ‘Exit’, that means container may be shutdown or may never up, run it again to resume for the shutdown case:

```
1 $ tools/docker/run linux-lab
```

Otherwise, check the running logs:

```
1 $ tools/docker/logs linux-lab
```


If normal, that means the login account and password may have been invalid for some exceptions, please regenerate new account and password with the coming steps:

Note: The `clean` command will remove some containers and data, please do necessary backup before run it, for example, save the container:

```
1 $ tools/docker/save linux-lab
```

VNC login fails while using mismatched password, to fix up such issue, please clean up all and rerun it:

```
1 $ tools/docker/clean linux-lab
2 $ tools/docker/rerun linux-lab
```

If the above command not work, please try this one (**It will clean more data, please do necessary backup**)

```
1 $ tools/docker/clean-all
2 $ tools/docker/rerun linux-lab
```

6.3.10 Ubuntu Snap Issues

Users report many snap issues, please use `apt-get` instead:

- users can not be added to docker group and break non-root operation.
- snap service exhausts the `/dev/loop` devices and break mount operation.

6.3.11 How to exit fullscreen mode of vnc clients

The easiest method is kill the VNC server in Linux Lab:

```
1 $ sudo pkill x11vnc
```

6.4 Lab Issues

6.4.1 No working init found

This means the `rootfs.ext2` image may be broken, please remove it and try `make boot` again, for example:

```
1 $ rm boards/aarch64/raspi3/bsp/root/2019.02.2/rootfs.ext2
2 $ make boot
```

`make boot` command can create this image automatically.

6.4.2 linux/compiler-gcc7.h: No such file or directory

This means using a newer gcc than the one linux kernel version supported, the solution is switching to an older gcc version via `make gcc-switch`, use i386/pc board as an example:

```
1 $ make gcc-list
2 $ make gcc-switch CCORR=internal GCC=4.4
```

6.4.3 linux-lab/configs: Permission denied

This may happen at `make boot` while the repository is cloned with `root` user, please simply update the owner of `cloud-lab/` directory:

```
1 $ cd /path/to/cloud-lab
2 $ sudo chown <USER>:<USER> -R ./
3 $ tools/docker/rerun linux-lab
```

Notes: To make a consistent working environment, Linux Lab only support using as general user: ‘ubuntu’.

6.4.4 scripts/Makefile.headersinst: Missing UAPI file

This means MAC OSX not use Case sensitive filesystem, create one using `hdiutil` or Disk Utility yourself:

```
1 $ hdiutil create -type SPARSE -size 60g -fs "Case-sensitive Journaled HFS+" -volname
   labspace labspace.dmg
2 $ hdiutil attach -mountpoint ~/Develop/labspace -nobrowse labspace.dmg.sparseimage
3 $ cd ~/Develop/labspace
```

6.4.5 unable to create file: net/netfilter/xt_dscp.c

This means Windows not enable filesystem’s case sensitive feature, just enable it:

```
1 $ cd /path/to/cloud-lab
2 $ fsutil file SetCaseSensitiveInfo ./ enable
```

6.4.6 how to run as root

By default, no password required to run as root with:

```
1 $ sudo -s
```

Notes: Please don’t use the ‘su’ command.

6.4.7 not in supported list

Such information means the specified value is not supported currently:

```
1 $ make boot ROOTDEV=vda
2 ERR: /dev/vda not in supported ROOTDEV list: /dev/sda /dev/ram0 /dev/nfs, update may help
   : 'make bsp B=mips64el/ls3a7a'. Stop.
3
4 $ make boot LINUX=v5.8
5 Makefile:594: *** ERR: v5.8 not in supported LINUX list: loongnix-release-1903 v5.7,
   clone one please: 'make kernel-clone KERNEL_NEW=v5.8'. Stop.
6
7 $ make boot QEMU=loongson-v1.1
8 Makefile:606: *** ERR: loongson-v1.1 not in supported QEMU list: loongson-v1.0, clone one
   please: 'make qemu-clone QEMU_NEW=loongson-v1.1'.
```

There are two main types:

- One is the specified version is not there or has not been verified
 - Please clone one and verify it with the usage of `xxx-clone` from section 5.
- Another is the specified value is invalid or simply not verified
 - For example, the above `vda` is not added in the `ROOTDEV_LIST`
 - This board may not support such type of device or just nobody verify and add it
 - This differs from board and kernel version

6.4.8 is not a valid rootfs directory

If using prebuilt filesystem, this error means the rootfs dir, ramdisk or harddisk creating procedure has been interrupted by `CTRL+C` or similar operations and it means the filesystem is not complete. If no important changes in BSP repository, reset it may help:

```
1 $ make bsp-cleanup
```

If using external filesystem, please make sure the filesystem architecture follows the Linux standards.

7. Contact and Sponsor

Contact Us

Welcome to join our discuss group:

- Wechat: **tinylab**
- Email: `contact /AT tinylab /dot/ org`

Give me a star

Welcome to mark our web site, star our git repositories:

- Wechat Group: Tinylab-Org
- Web site: <https://tinylab.org>
 - Created near 10+ years
 - Focus on Linux kernel and embedded Linux system
- Git Repositories
 - Gitee: <https://gitee.com/tinylab>
 - Github: <https://github.com/tinyclub>

Buy our products

- Store: <https://shop155917374.taobao.com>
 - The taobao store of TinyLab.org community, mainly sell products around our open source projects
 - The products include Linux Lab Disk, Pocket Linux Disk and the Linux Lab boards
 - Welcome to buy some based on your requirement, you can search ‘Linux Lab’ in taobao application to find us
- Circle: <https://t.zsxq.com/uB2vJyF>
 - The VIP knowledge channel of TinyLab.org community
 - Create 3+ years, about 1000+ shares and 20+ Linux professionals
- Courses: <https://m.cctalk.com/inst/sh8qtdag>
 - TinyLab School – The video channel of TinyLab.org community
 - Video Live, Video Courses are shared by active Linux professionals from TinyLab.org community

Sponsor

Sponsor via wechat



联系我们



捐赠项目

Figure 3: contact-sponsor

Sponsor list

- 2021
 - [Lazyparser](#)
 - * HelloGCC and HelloLLVM founder
 - * 5000RMB
 - [Summer 2021](#)
 - * Such projects are sponsored by Summer 2021: Rust for Linux, openEuler Kernel for aarch64/virt and x86_64/pc
 - T-head
 - * 1 D1 board
 - Allwinner
 - * 3 D1 boards
- 2020
 - [Loongson](#)
 - * The famous Chinese Loongson CPU designer and manufacturer
 - * Such boards are sponsored by Loongson: mips64el/ls2k, mips64el/ls3a7a, mipsel/ls1b, mipsel/ls232
 - [Summer 2020](#)
 - * Such projects are sponsored by Summer 2020: Linux Lab docker image upgrade from Ubuntu 14.04 to Ubuntu 20.04

– Embedfire

* 6 imx6ull boards