Linux From Scratch (LFS)

Linux From Scratch

"Linux From Scratch (LFS) is a project that provides you with step-by-step instructions for building your own custom Linux system, entirely from source code."

— *https://linuxfromscratch.org*

Why LFS?

- For learning purposes only

- Started in 1999
- Latest update Sep 2022

- Teaches what “Linux” is
- Teaches about Kernel vs use

- It will not be
  - Up-to-date
  - Secure
  - Fast
  - Fully featured (e.g., no package manager or desktop)
  - Maintainable
- It will be
  - Fun
  - Informative
  - Tedious

Cross-Compiling

"A **cross compiler** is a compiler capable of creating executable code for a platform other than the one on which the compiler is running."

— Wikipedia

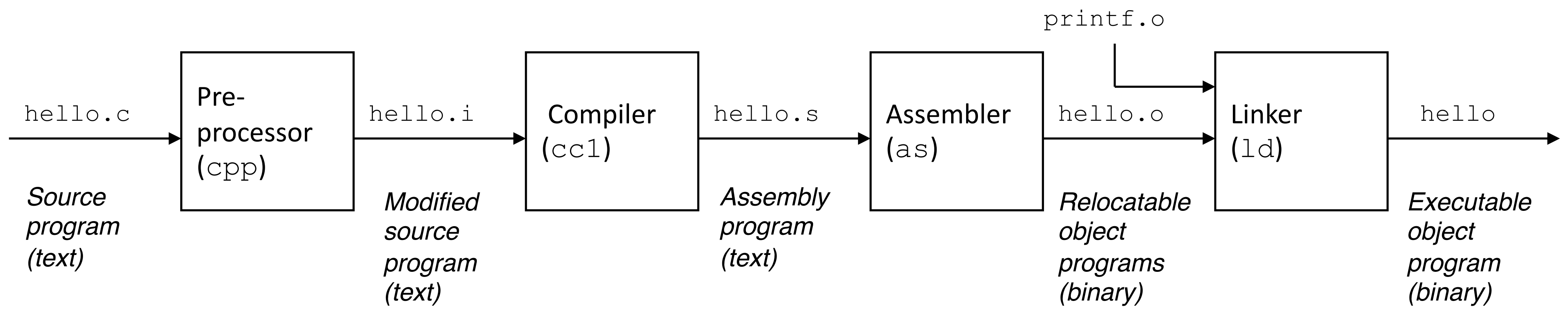
For example

- Compile an Android App using Windows
- Compile a MacOS App using Linux
- Compile an Xbox Game using Unreal Engine on Windows or Linux
- Compile a Web Assembly App using Zig on MacOS

Do you want to compile on your phone?

Cross-Compiling

Cross-Compiling is surprisingly complicated

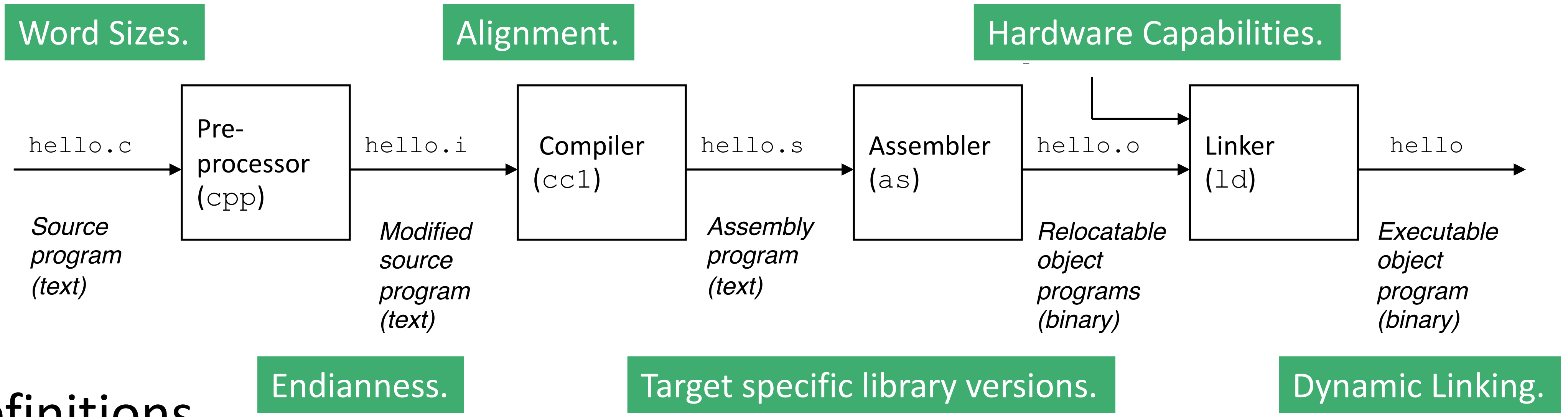


Definitions

- Host: computer running the compiler
- Target: computer on which the application will run

Cross-Compiling

Cross-Compiling is surprisingly complicated



Definitions

- Host: computer running the compiler
- Target: computer on which the application will run

Frustratingly, you normally need a compiler toolchain built for the target.

Circular Dependencies

- We are not inserting punch cards into a reader anymore
- The cross-compiling issues discussed in the previous slide are due to circular dependencies

- We need a binary (driver) to get input from the keyboard, but where does that binary come from?

- It must be compiled for the system. But how do we compile it if we can’t even type!?

What’s Needed to Cross-Compile?

To cross compile, we need

1. An existing kernel (OS)
2. A C library (`fwrite`, `printf`, `socket`, etc.)
3. Command line utitlies (bash, tar, scp, ssh, etc.)
4. A compiler toolchain
    1. Compiler (gcc)
    2. Linker (ld)
    3. Build system (make)
    4. And some other extras

Boot Process

Main (Mother) Board

1. Power-on
2. BIOS copied to RAM
3. Run BIOS firmware
4. Copy MBR from HD to RAM
5. Run MBR (bootloader)
6. Copy OS kernel to RAM
7. Run kernel (services/daemons)
8. Mount root partitions

Creating a “Bootable” Operating System

1. Needs to work with existing BIOS
2. Create a partitioned hard drive (e.g., USB or fake QEMU drive)
    1. Drive needs to include a marked boot partition
    2. Drive needs to include a root partition
3. Setup a build machine with the utilities needed to compile for a new target system (install stuff on your laptop or in QEMU)
4. Cross-compile all software needed and store in boot and root partitions
5. Copy data into newly partitioned hard drive
6. Boot new target machine with hard drive installed

Linux From Scratch

- I don’t want to brick a spare laptop (or have you all do so either)
- We are going to create our own Linux using QEMU

Rough process

1. Create a fake build system with QEMU
2. Use the fake build system to build a bootable LFS image
3. Boot the LFS image using QEMU

Host, Builder, and Target

Linux From Scratch Methodology

1. Prepare the build environment (safely in QEMU build env for us)
2. Build cross-compiling toolchain (safely in QEMU build env for us)
3. Build LFS system
4. Configure the new LFS System

Linux From Scratch Methodology

1. Prepare the build environment (safely in QEMU build env for us)
2. Build cross-compiling toolchain (safely in QEMU build env for us)
3. Build LFS system
4. Configure the new LFS System

Linux From Scratch Methodology

1. Prepare the build environment (safely in QEMU build env for us)
    1. Install (as root) tools needed to create the cross-compiler
    2. Format and partition the LFS boot medium (e.g., hard drive)
    3. Mount the LFS partitions
    4. Create directory structure in new partitions
    5. Create a new user for safety and sandboxing
    6. Setup the build user environment
2. Build cross-compiling toolchain (safely in QEMU build env for us)
3. Build LFS system
4. Configure the new LFS System

Linux From Scratch Methodology

1. Prepare the build environment (safely in QEMU build env for us)
2. Build cross-compiling toolchain (safely in QEMU build env for us)
    1. Compile cross-toolchain (binutils, gcc, linux headers, glibc, libstdc++)
    2. Cross compile helper tools (bash, make, tar, binutils, gcc)
        - Handles circular dependencies
    3. Create an environment isolated from the builder host (Fedora)
    4. Finish compiling and installing cross-toolchain in isolated chroot
3. Build LFS system
4. Configure the new LFS System

There is a difference between development packages and the resulting LFS packages.

Linux From Scratch Methodology

1. Prepare the build environment (safely in QEMU build env for us)
2. Build cross-compiling toolchain (safely in QEMU build env for us)
3. Build LFS system
    1. Install cross-compiled packages into LFS file system
4. Configure the new LFS System

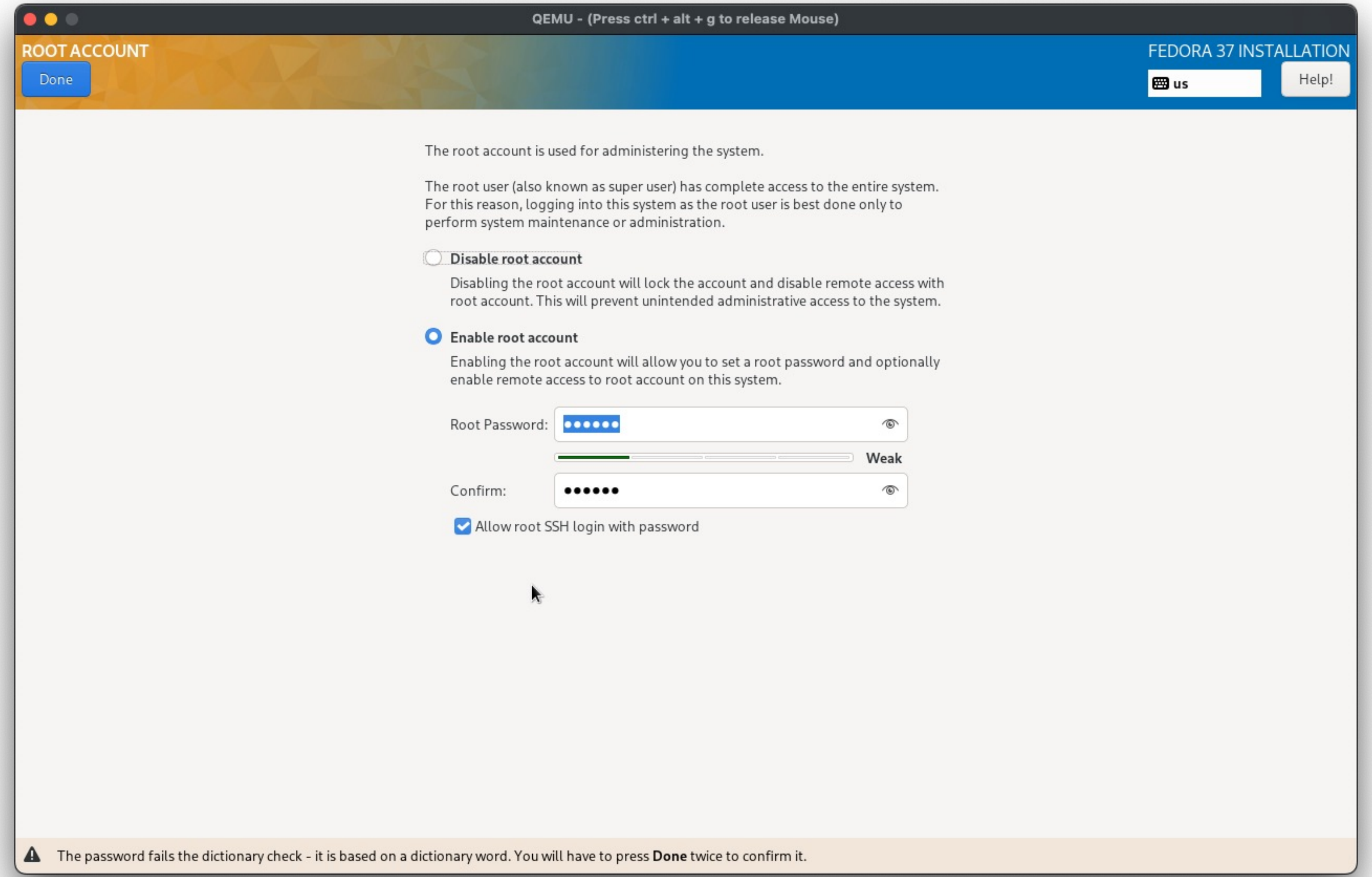
Linux From Scratch Methodology

1. Prepare the build environment (safely in QEMU build env for us)
2. Build cross-compiling toolchain (safely in QEMU build env for us)
3. Build LFS system
4. Configure the new LFS System
    1. Make the system "bootable"

LFS Process Diagram

Builder Environment

```
# Create a disk image for the builder environment
❯ qemu-img create -f qcow2 builder.img 32G
Formatting 'builder.img', fmt=qcow2 cluster_size=65536 extended_l2=off
compression_type=zlib size=34359738368 lazy_refcounts=off refcount_bits=16

# Install Fedora 37 in builder environment (settings are specific to my mac)
❯ qemu-system-x86_64                               \
    -cpu host,-pdpe1gb                             \
    -m 8G                                          \
    -smp 4                                         \
    -accel hvf                                     \
    -usb -device usb-tablet                        \
    -vga virtio                                    \
    -display default,show-cursor=on                \
    -audiodev coreaudio,id=coreaudio               \
    -device ich9-intel-hda                         \
    -device hda-output,audiodev=coreaudio \
    -drive file=builder.img,if=virtio              \
    -boot d -cdrom Fedora-Server-dvd-x86_64-37-1.7.iso
```



The root account is used for administering the system.

The root user (also known as super user) has complete access to the entire system. For this reason, logging into this system as the root user is best done only to perform system maintenance or administration.

Disable root account
Disabling the root account will lock the account and disable remote access with root account. This will prevent unintended administrative access to the system.

[✓ Enable root account]
Enabling the root account will allow you to set a root password and optionally enable remote access to root account on this system.

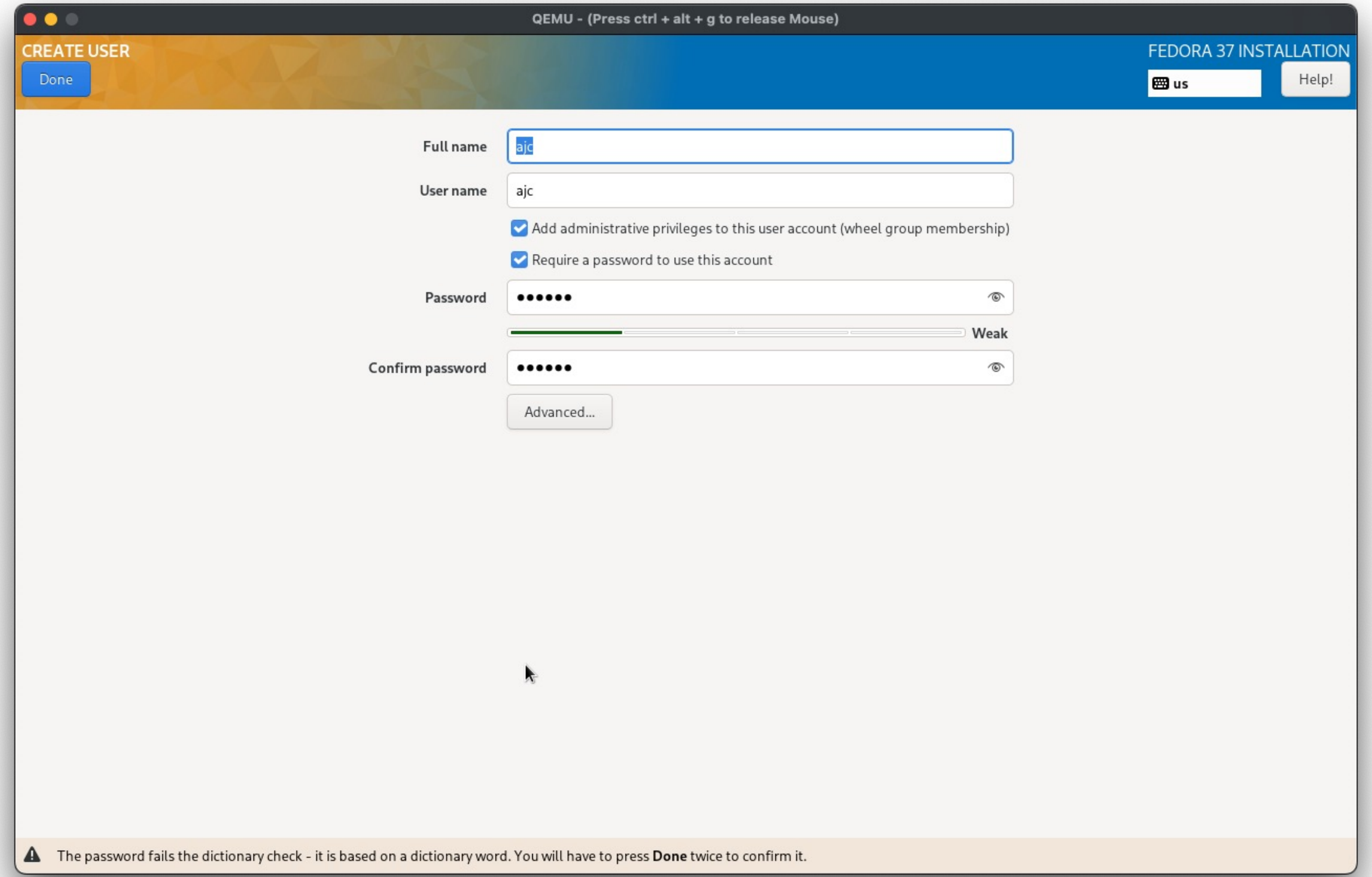
Root Password: ●●●●●●

Weak

Confirm: ●●●●●●

[✓] Allow root SSH login with password

The password fails the dictionary check - it is based on a dictionary word. You will have to press **Done** twice to confirm it.



Full name: ajc

User name: ajc

☑ Add administrative privileges to this user account (wheel group membership)

☑ Require a password to use this account

Password: ●●●●●●

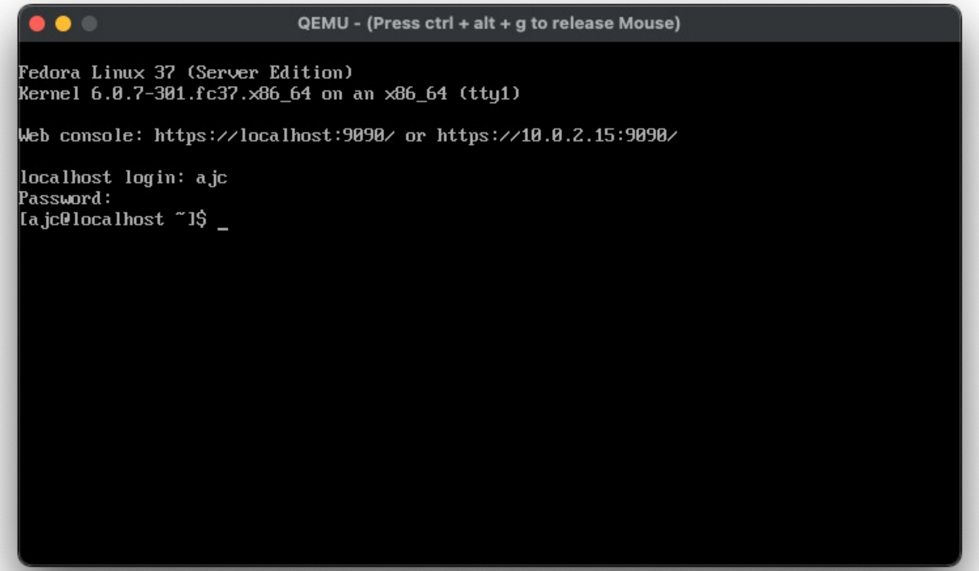
Weak

Confirm password: ●●●●●●

The password fails the dictionary check - it is based on a dictionary word. You will have to press **Done** twice to confirm it.

Boot Builder Environment

```
❯ qemu-system-x86_64                              \
    -cpu host,-pdpe1gb                            \
    -m 8G                                         \
    -smp 4                                        \
    -accel hvf                                    \
    -usb -device usb-tablet                       \
    -vga virtio                                   \
    -display default,show-cursor=on               \
    -audiodev coreaudio,id=coreaudio              \
    -device ich9-intel-hda                        \
    -device hda-output,audiodev=coreaudio \
    -drive file=builder.img,if=virtio
```



```
Fedora Linux 37 (Server Edition)
Kernel 6.0.7-301.fc37.x86_64 on an x86_64 (tty1)

Web console: https://localhost:9090/ or https://10.0.2.15:9090/

localhost login: ajc
Password:
[ajc@localhost ~]$ _
```

```
# Host environment
❯ qemu-system-x86_64…
```

```
# Builder Environment
sudo dnf update -y
```

Update Build Environment

```
# Run the following inside Builder Environment

# Update default software packages
sudo dnf update -y

# Shutdown (so that it can be restarted)
sudo shutdown now –h

# Restart builder and start ssh daemon
sudo systemctl enable --now sshd
sudo shutdown now -h
```

LFS Drive

- We need to create a “bootable” hard drive for our LFS OS

```
❯ qemu-img create -f qcow2 lfs.img 16G
Formatting 'lfs.img', fmt=qcow2 cluster_size=65536 extended_l2=off
compression_type=zlib size=17179869184 lazy_refcounts=off refcount_bits=16
```

Start Builder and SSH In

```
❯ qemu-system-x86_64 -cpu host,-pdpe1gb -m 8G -smp 4 -accel hvf -usb -
device usb-tablet -vga virtio -display default,show-cursor=on -audiodev
coreaudio,id=coreaudio -device ich9-intel-hda -device hda-
output,audiodev=coreaudio -drive file=builder.img,if=virtio -drive
file=lfs.img,if=virtio -display none -device virtio-net,netdev=vmnic -
netdev user,id=vmnic,hostfwd=tcp::1234-:22
```

```
❯ ssh -p 1234 ajc@localhost
ajc@localhost's password:
```

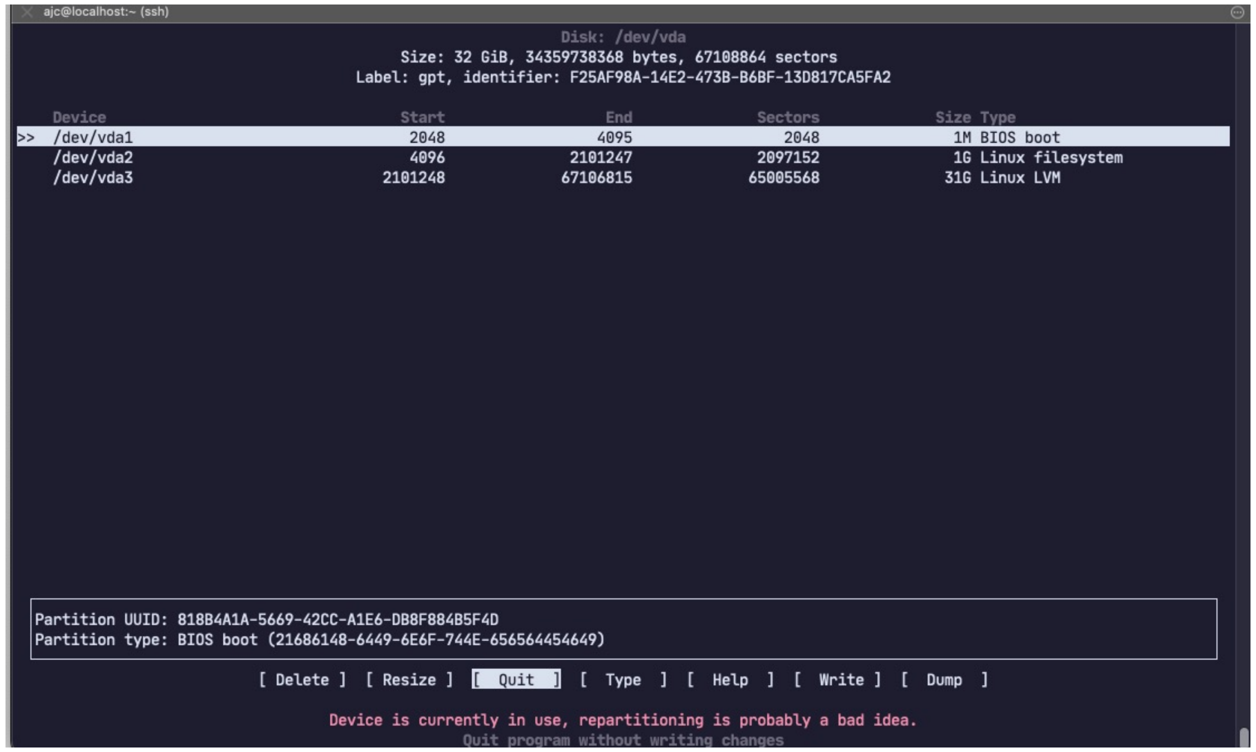
```
Web console: https://localhost:9090/ or https://10.0.2.15:9090/

Last login: Tue Dec  6 13:52:21 2022 from 10.0.2.2
[ajc@localhost ~]$ hostname
localhost.localdomain
[ajc@localhost ~]$
```

Update Builder

```
# Switch to root user and go to /root
su
cd /root

# Install needed dependencies
dnf groupinstall "C Development Tools and Libraries" -y
dnf groupinstall "Development Tools" -y
dnf install texinfo -y

# Run version-check.sh (Section 2.2)
bash version-check.sh
```

Format LFS Drive

```
[root@localhost ~]# lsblk
NAME             MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sr0               11:0    1 1024M  0 rom
zram0            251:0    0  7.8G  0 disk [SWAP]
vda              252:0    0   32G  0 disk
├─vda1           252:1    0    1M  0 part
├─vda2           252:2    0    1G  0 part /boot
└─vda3           252:3    0   31G  0 part
  └─fedora-root  253:0    0   15G  0 lvm  /
vdb              252:16   0   16G  0 disk
[root@localhost ~]# cfdisk /dev/vdb

Syncing disks.
[root@localhost ~]# lsblk
NAME             MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sr0               11:0    1 1024M  0 rom
zram0            251:0    0  7.8G  0 disk [SWAP]
vda              252:0    0   32G  0 disk
├─vda1           252:1    0    1M  0 part
├─vda2           252:2    0    1G  0 part /boot
└─vda3           252:3    0   31G  0 part
  └─fedora-root  253:0    0   15G  0 lvm  /
vdb              252:16   0   16G  0 disk
├─vdb1           252:17   0  512M  0 part
├─vdb2           252:18   0  256M  0 part
└─vdb3           252:19   0 15.2G  0 part
```



```
                              Disk: /dev/vda
            Size: 32 GiB, 34359738368 bytes, 67108864 sectors
        Label: gpt, identifier: F25AF98A-14E2-473B-B6BF-13D817CA5FA2

    Device                 Start        End      Sectors    Size Type
>>  /dev/vda1               2048       4095         2048      1M BIOS boot
    /dev/vda2               4096    2101247      2097152      1G Linux filesystem
    /dev/vda3            2101248   67106815     65005568     31G Linux LVM

 Partition UUID: 818B4A1A-5669-42CC-A1E6-DB8F884B5F4D
 Partition type: BIOS boot (21686148-6449-6E6F-744E-656564454649)

      [ Delete ]  [ Resize ]  [  Quit  ]  [  Type  ]  [  Help  ]  [ Write ]  [  Dump  ]

           Device is currently in use, repartitioning is probably a bad idea.
                       Quit program without writing changes
```

Set the File System Formats

```
# File system types (Section 2.5)
mkfs -v -t ext4 /dev/vdb2
mkfs -v -t ext4 /dev/vdb3
mkswap /dev/vdb1

# Setting $LFS (add to .bashrc) (Section 2.6)
export LFS=/mnt/lfs

# Mount partitions (Section 2.7)
mkdir -pv $LFS
mount -v -t ext4 /dev/vdb3 $LFS
restorecon -R /mnt/lfs

mkdir -pv $LFS/boot
mount -v -t ext4 /dev/vdb2 $LFS/boot

/sbin/swapon -v /dev/vdb1
```

Partition and Mounted File System

Download Build and LFS Sources

```
# Download Sources (Section 3.1)
mkdir -v $LFS/sources
chmod -v a+wt $LFS/sources

wget https://www.linuxfromscratch.org/lfs/view/stable/wget-list-sysv
wget --input-file=wget-list-sysv --continue --directory-prefix=$LFS/sources

wget https://github.com/libexpat/libexpat/releases/download/R_2_4_8/expat-2.4.8.tar.xz
wget https://www.zlib.net/fossils/zlib-1.2.12.tar.gz

wget https://www.linuxfromscratch.org/lfs/view/stable/md5sums
cp md5sums $LFS/sources
pushd $LFS/sources
      md5sum -c md5sums
popd
```

Create (Partial) Linux Directory Structure

```
# Create directory layout (Section 4.2)
mkdir -pv $LFS/{etc,var} $LFS/usr/{bin,lib,sbin}

for i in bin lib sbin; do
  ln -sv usr/$i $LFS/$i
done

case $(uname -m) in
  x86_64) mkdir -pv $LFS/lib64 ;;
esac

mkdir -pv $LFS/tools
```

Create LFS User for Isolation/Sandboxing

```
# Create LFS user (Section 4.3)
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
passwd lfs

chown -v lfs $LFS/{usr{,/*},lib,var,etc,bin,sbin,tools}

case $(uname -m) in
  x86_64) chown -v lfs $LFS/lib64 ;;
esac

su - lfs
```

Make and Install binutils

```
cd $LFS/sources

tar xf binutils-2.39.tar.xz
cd binutils-2.39

mkdir -v build
cd       build

../configure --prefix=$LFS/tools \
             --with-sysroot=$LFS \
             --target=$LFS_TGT   \
             --disable-nls       \
             --enable-gprofng=no \
             --disable-werror

make
make install
```

Make and Install gcc

```
cd $LFS/sources
tar xf gcc-12.2.0.tar.xz
cd gcc-12.2.0
…

mkdir -v build
cd       build

../configure                  \
    --target=$LFS_TGT         \
    --prefix=$LFS/tools       \
    --with-glibc-version=2.36 \
…

make
make install
```

Make and Install…

- Linux API Headers
- Glibc
- Libstd++

- These (along with binutils and gcc) are all that is needed for cross-compiling

Cross Compile Helper Utilities

- They all follow this process
  1. `cd $LFS/sources`
  2. `tar xf UTILITY`
  3. `cd UTILITY`
  4. Call `configure` with needed settings
  5. Call `make`
  6. Call `make install` with `DESTDIR=$LFS`

- This includes: m4, ncurses, bash, coreutils, diffutils, file, findutils, gawk, grep, gzip, make, patch , sed, tar, xz, binutils, gcc

Making LFS Bootable