Sessão 9: Criação de sistemas Linux customizados

Nesta sessão iremos aprender como criar uma distribuição Linux sob medida usando o *live-build*, um sistema de construção de distribuições Debian sob medida.

1) Topologia desta sessão

Criaremos apenas uma nova máquina nesta sessão, a saber:

- `live`, sistema para construção de imagens Linux customizadas usando o sistema *live-build*. Endereço IP 10.0.42.11/24.

1. Como de costume, vamos à criação dos registros DNS. Acesse a máquina `ns1` como o usuário `root`:

```
# hostname ; whoami
ns1
root
```

Edite o arquivo de zonas `/etc/nsd/zones/intnet.zone`, inserindo entradas A para a máquinas indicadas no começo desta atividade. **Não se esqueça** de incrementar o valor do serial no topo do arquivo!

```
# nano /etc/nsd/zones/intnet.zone
(...)
```

```
# grep live /etc/nsd/zones/intnet.zone
live    IN    A                10.0.42.11
```

Faça o mesmo para o arquivo de zona reversa:

```
# nano /etc/nsd/zones/10.0.42.zone
```

```
# grep live /etc/nsd/zones/10.0.42.zone
11      IN   PTR               live.intnet.
```

Assine o arquivo de zonas usando o *script* criado anteriormente:

```
# bash /root/scripts/signzone-intnet.sh
reconfig start, read /etc/nsd/nsd.conf
ok
ok
ok
ok removed 6 rrsets, 2 messages and 0 key entries
```

Verifique a criação das entradas usando o comando `dig`:

```
# dig live.intnet +short
10.0.42.11
```

```
# dig -x 10.0.42.11 +short
live.intnet.
```

2) Criação da VM de build

1. Vamos criar a VM `live` e utilizá-la para criar imagens Linux customizadas. Clone a máquina `debian-template` para uma de nome `live`, com uma única interface de rede conectada à DMZ. O IP da máquina será 10.0.42.11/24.

   Concluída a clonagem, na console principal do Virtualbox, acesse o menu *Settings*. Em *System* > *Motherboard* > *Base Memory*, aloque ao menos 4 GB de RAM para essa VM. Adicionalmente, vá para *System* > *Processor* e aumente o número de processadores disponíveis para a máquina, tanto quanto possível: se você tiver à disposição uma máquina *quad-core*, por exemplo, aloque dois processadores; caso sejam oito *cores*, aloque quatro CPUs, e assim sucessivamente.

   Essas alterações são necessárias pois o processo de *build* de uma nova distribuição é bastante intensivo computacionalmente, e quanto mais processamento tivermos à disposição, mais cedo concluiremos os passos.

   Em *Settings* > *Storage* > *Controller: SATA*, adicione um novo disco à VM. Escolha o formato VDI, alocação dinâmica de espaço, nome da unidade `live-build` e 20 GB de tamanho. Iremos usar este espaço para fazer o download e instalação dos pacotes das distribuições customizadas em um *chroot* dedicado.

   Finalmente, clique em *OK* e ligue a VM. Logue como `root` e use o script `/root/scripts/changehost.sh` para fazer a configuração automática, como de costume.

```
# hostname ; whoami
debian-template
root
```

```
# bash ~/scripts/changehost.sh -h live -i 10.0.42.11 -g 10.0.42.1
Signing ssh_host_ecdsa_key.pub key...
Signing ssh_host_ed25519_key.pub key...
Signing ssh_host_rsa_key.pub key...
Configuring host key trust...
Configuring user key trust...
All done!
```

2. Aplique o *baseline* de segurança à máquina `live`, repetindo o que fizemos no passo (2), atividade (2) da sessão 7:

```
$ hostname ; whoami
client
ansible
```

```
$ sed -i '/\[srv\]/a live' ~/ansible/hosts
```

```
$ ansible-playbook -i ~/ansible/hosts -l live -Ke ansible_become_method=su
~/ansible/srv.yml ; ansible-playbook -i ~/ansible/hosts -l live ~/ansible/srv.yml
SUDO password:

(...)

PLAY RECAP
*********************************************************************************
************************

live                       : ok=10   changed=8    unreachable=0    failed=0
```

3) Construindo uma distribuição mínima

1. Acesse a máquina `live` como o usuário `root`:

```
# hostname ; whoami
docker1
root
```

2. Vamos preparar o disco para uso. Descubra sob qual nome ele foi detectado:

```
# dmesg | grep 'GiB'
[    1.507032] sd 2:0:0:0: [sda] 16777216 512-byte logical blocks: (8.59 GB/8.00
GiB)
[    1.507324] sd 3:0:0:0: [sdb] 41943040 512-byte logical blocks: (21.5 GB/20.0
GiB)
```

Perfeito, o nome do disco é `/dev/sdb`. Use o `fdisk` para formatá-lo — crie uma única partição do tipo LVM ocupando a totalidade do espaço.

```
# fdisk /dev/sdb

Bem-vindo ao fdisk (util-linux 2.29.2).
As alterações permanecerão apenas na memória, até que você decida gravá-las.
Tenha cuidado antes de usar o comando de gravação.

A unidade não contém uma tabela de partição conhecida.
Criado um novo rótulo de disco DOS com o identificador de disco 0x19d210eb.
```

```
Comando (m para ajuda): o
Criado um novo rótulo de disco DOS com o identificador de disco 0xdaead0ba.
```

```
Comando (m para ajuda): n
Tipo da partição
   p   primária (0 primárias, 0 estendidas, 4 livre)
   e   estendida (recipiente para partições lógicas)
Selecione (padrão p): p
Número da partição (1-4, padrão 1): 1
Primeiro setor (2048-41943039, padrão 2048):
Último setor, +setores ou +tamanho{K,M,G,T,P} (2048-41943039, padrão 41943039):

Criada uma nova partição 1 do tipo "Linux" e de tamanho 20 GiB.
```

```
Comando (m para ajuda): t
Selecionou a partição 1
Tipo de partição (digite L para listar todos os tipos): 8e
O tipo da partição "Linux" foi alterado para "Linux LVM".
```

```
Comando (m para ajuda): w
A tabela de partição foi alterada.
Chamando ioctl() para reler tabela de partição.
Sincronizando discos.
```

Inicialize o disco para o sistema LVM:

```
# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created.
```

Crie novos VG/LV para o disco:

```
# vgcreate vg-live /dev/sdb1
  Volume group "vg-live" successfully created
```

```
# lvcreate -l +100%FREE -n lv-live vg-live
  Logical volume "lv-live" created.
```

Formate-o sob o sistema de arquivos `ext4`:

```
# mkfs.ext4 /dev/mapper/vg--live-lv--live
mke2fs 1.43.4 (31-Jan-2017)
Creating filesystem with 5241856 4k blocks and 1310720 inodes
Filesystem UUID: 0716d2c6-cd6a-43a0-ab04-83075886e790
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

Configure a montagem automática via `/etc/fstab` no diretório `/live`, e monte o disco:

```
# mkdir /live
```

```
# echo '/dev/mapper/vg--live-lv--live /live ext4 defaults 0 2' >> /etc/fstab
```

```
# mount -a
```

```
# df -h | grep live
/dev/mapper/vg--live-lv--live   20G   45M   19G   1% /live
```

3. Instale a ferramenta que iremos utilizar para construir imagens customizadas, o *live-build*:

```
# apt-get install -y live-build
```

4. Vamos fazer uma imagem mínima para experimentar com o sistema do *live-build*. Crie o diretório novo `/live/basic`, e entre nele:

```
# mkdir /live/basic ; cd /live/basic
```

O comando `lb config` irá criar uma estrutura de diretórios padrão para o *live-build* operar:

```
# lb config
```

Antes de disparar o *build*, vamos apenas customizar o conjunto de repositórios usados para buscar os pacotes de instalação para uma opção mais veloz:

```
# sed -i 's|ftp.debian.org|ftp.br.debian.org|g' /live/basic/config/bootstrap
```

```
# sed -i 's|ftp.debian.org|ftp.br.debian.org|g' /live/basic/config/build
```

Desabilite também a instalação de *firmwares* diversos para o kernel Linux — como iremos rodar a distribuição exclusivamente em um ambiente virtualizado, não há porque atrasar o *build* e aumentar o tamanho da imagens com *drivers* que não utilizaremos no sistema finalizado.

```
# sed -i 's|^\(LB_FIRMWARE_[A-Z]*=\).*|\1\"false\"|' /live/basic/config/binary
```

Finalmente, inicie o *build* do sistema customizado, e aguarde sua construção. Esse passo pode ser relativamente demorado, então tenha paciência.

```
# lb build
[2018-11-17 23:51:27] lb build
P: live-build 1:20170213
P: Building config tree for a debian/stretch/amd64 system

(...)

Reading package lists... Done
Building dependency tree
Reading state information... Done
[2018-11-17 23:57:01] lb source
```

Para ilustrar o tempo médio a esperar, note que o *build* acima levou cerca de seis minutos para concluir. Esse tempo pode variar para mais, ou para menos, dependendo da velocidade do processador, memória e disco da máquina física, bem como o volume de recursos alocados à

VM `live`.

5. O que esse comando produziu? Vejamos:

```
# ls *.iso
live-image-amd64.hybrid.iso
```

```
# du -sm live-image-amd64.hybrid.iso
216     live-image-amd64.hybrid.iso
```

Foi criada uma imagem ISO de 216 MB, com um sistema *live* perfeitamente bootável, que testaremos a seguir.

6. Copie a imagem ISO produzida no passo (4) acima para sua máquina física. Há vários métodos para se atingir esse objetivo — um dos meus favoritos é usar o comando `scp` no Cygwin, que permite cópia direta da VM para a máquina física, sem a necessidade de instalação de qualquer software adicional. Caso não tenha o Cygwin instalado, considere usar o programa WinSCP, uma ferramenta gráfica que provê funcionalidade semelhante.

   No exemplo abaixo, copiaremos via `scp` + Cygwin a ISO para a Área de Trabalho do usuário `fbs`, na máquina física Windows:

```
$ scp aluno@10.0.42.11:/live/basic/live-image-amd64.hybrid.iso
/cygdrive/c/Users/fbs/Desktop/
aluno@10.0.42.11's password:
live-image-amd64.hybrid.iso
100%  216MB  59.5MB/s   00:03
```

7. Na console principal do Virtualbox, crie uma nova máquina virtual. Como nome, define `iso-test`, *Type* `Linux` e *Version* `Debian (64-bit)`. Mantenha o valor padrão de memória RAM, 1024 GB. Quando perguntado sobre o disco rígida da VM, escolha a opção *Do not add a virtual hard disk* e clique em *Create*, confirmando sua escolha.

   Selecione a nova VM e acesse o menu *Settings* > *Storage* > *Controller: IDE*. Selecione o *drive* de CD vazio e, em *Attributes*, escolha o arquivo de disco ótico virtual que copiamos no passo anterior, a imagem `live-image-amd64.hybrid.iso`.

   Clique em *OK* e ligue a máquina virtual. Após a tela de BIOS do Virtualbox, você verá o *bootloader* do sistema customizado que construímos, como mostrado abaixo:

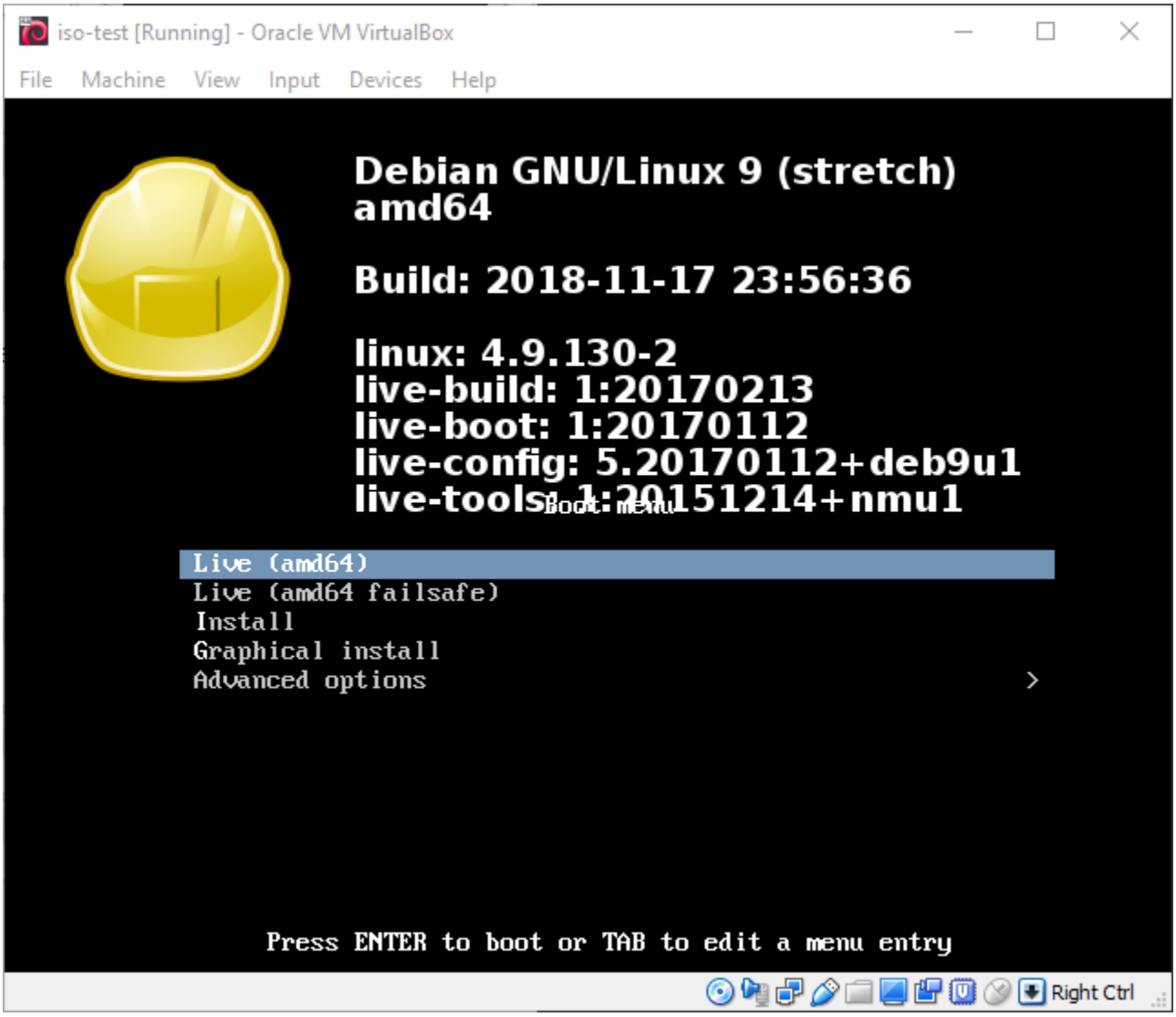

Figura 1. Bootloader do sistema live

Selecione a primeira opção, e prossiga com o *boot*. Brevemente surgirá um *shell* para interação com o sistema — estamos logados com o usuário `user`:

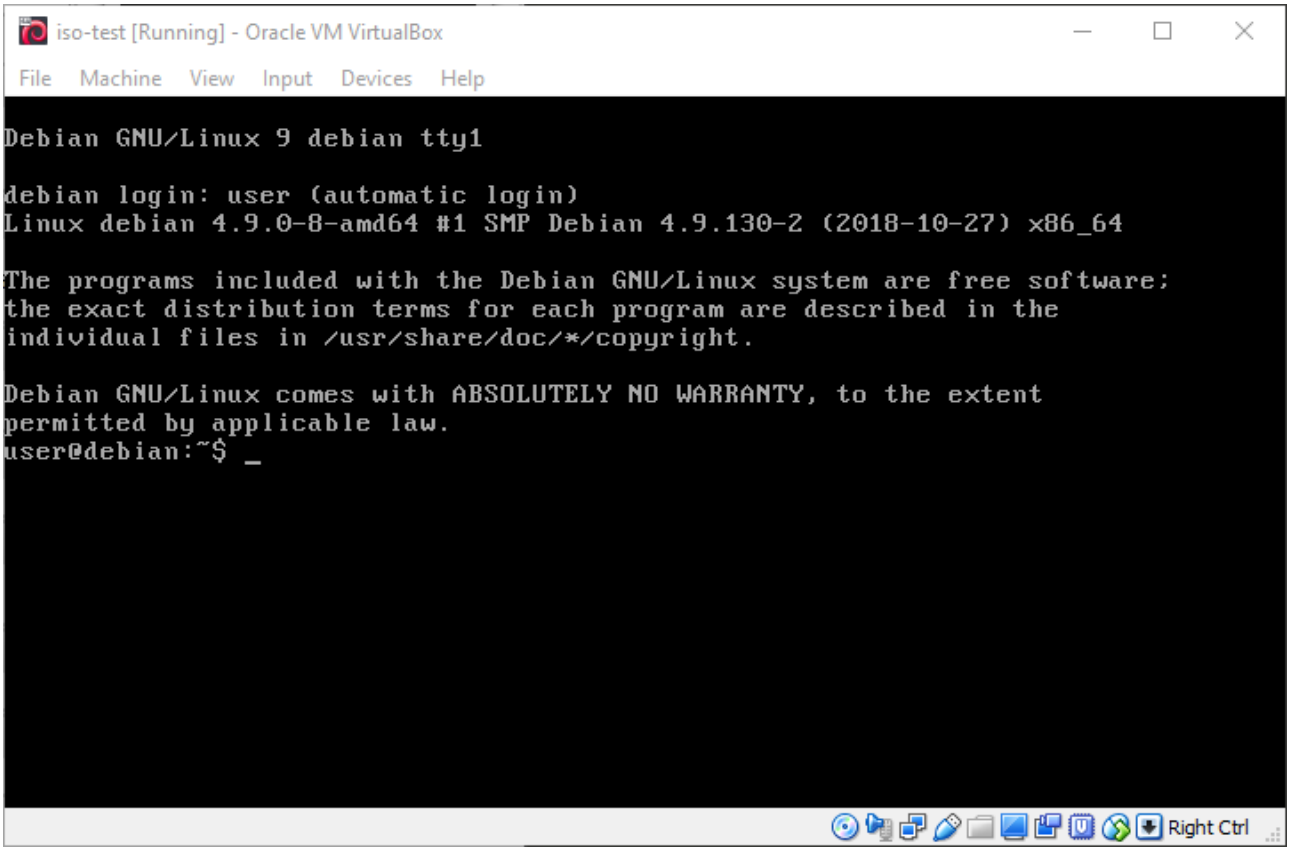

Figura 2. Shell do sistema live

Temos em mãos um sistema plenamente funcional, operando em memória. Para escalar privilégio para `root`, execute `sudo -i`. Para colocar em perspectiva o quão enxuto é o sistema que estamos trabalhando, observe a memória utilizada:

```
# free -m
              total        used        free      shared  buff/cache   available
Mem:            996          26         872           7          97         847
Swap:             0           0           0
```

Apenas 26 MB de RAM! Conte o número de pacotes instalados:

```
# dpkg -l | grep '^ii' | wc -l
198
```

Compare com a máquina `live`, por exemplo, que é derivada de um sistema bastante minimalista que construímos durante a primeira sessão deste curso: ela possui 360 pacotes instalados.

```
# hostname ; dpkg -l | grep '^ii' | wc -l
live
360
```

8. Como o sistema customizado que fizemos está perfeitamente funcional, vamos instalar o cliente OpenSSH:

```
# apt-get install -y openssh-client
```

O APT consegue instalar pacotes mesmo operando em memória, como podemos ver. Produza uma lista dos pacotes instalados e copie-a para a máquina `live`:

```
# dpkg -l | grep '^ii' > /tmp/basic-packages.txt
```

```
# scp /tmp/basic-packages.txt aluno@10.0.42.11:~
aluno@10.0.42.11's password:
basic-packages.txt                                100%   24KB  20.4MB/s   00:00
```

9. Apesar de interessante, nosso sistema customizado ainda não faz muita coisa. Desligue-o, e vamos tentar incrementá-lo.

```
# halt -p
```

4) Utilizando um repositório local de pacotes

1. Volte à máquina `live`, como o usuário `root`. Em seguida, verifique o tamanho do diretório `/live/basic`.

```
# hostname ; whoami
live
root
```

```
# du -sh /live/basic/
1,6G    /live/basic/
```

Note que mesmo para produzir um sistema tão básico quando o que fizemos na atividade (3), o sistema de *build* ocupou 1,6 **gigabytes** de espaço em disco! Além da necessidade de criar um sistema de *bootstrap* e um *chroot* que contém o sistema customizado, é também necessário armazenar todos os pacotes `.deb` que foram baixados durante o *build*.

2. Para limpar os arquivos de trabalho do *live-build*, utilize o comando `lb clean`. Para remover a totalidade dos arquivos gerados pelo *live-build*, como diretórios de *cache*, *chroot*, binários e fontes, use a opção `--purge`.

```
# cd /live/basic ; lb clean --purge
```

Volte a verificar o tamanho da pasta `/live/basic`:

```
# du -sh /live/basic/
208K    /live/basic/
```

3. Se quisermos produzir um novo sistema, como fizemos antes, basta rodar o comando `lb build` novamente. Note, porém, que todos os pacotes `.deb` de instalação do *bootstrap* e sistema-base serão baixados novamente — como a construção de sistemas customizados normalmente envolve uma boa quantidade de tentativa e erro, a quantidade de dados a serem baixados a cada *build* irá rapidamente se tornar a principal fonte de atraso no processo.

   Para suplantar esse problema, podemos construir um repositório local de pacotes, usando por exemplo a ferramenta Aptly. Ao contrário de um espelho total do repositório de pacotes do Debian, que é gigantesco (estimado em 326 GB quando da escrita desta atividade, ref. https://www.debian.org/mirror/size), o Aptly permite que façamos um repositório local bastante enxuto, contendo apenas os pacotes necessários à instalação do sistema-base e suas dependências.

   Copie o diretório `/live/basic` para `/live/aptly-basic`, e entre dentro do novo diretório.

```
# cp -a /live/basic /live/aptly-basic ; cd /live/aptly-basic
```

   Agora, crie o arquivo novo `/live/aptly-basic/makebuild.sh`, com o seguinte conteúdo:

```
 1 #!/bin/bash
 2
 3 ABS_PATH=`readlink -f $0 | sed 's/\/[^\/]*$//'`
 4
 5 if uname -r | egrep '686-pae$' &> /dev/null; then
 6   LB_DISK="${ABS_PATH}/live-image-i386.img"
 7   RAW_DISK="${ABS_PATH}/live-image-i386.raw"
 8   LINUX_HEADERS="linux-headers-686-pae"
 9   REPO_ARCH="i386"
10 else
11   LB_DISK="${ABS_PATH}-amd64.img"
12   RAW_DISK="${ABS_PATH}/live-image-amd64.raw"
13   LINUX_HEADERS="linux-headers-amd64"
14   REPO_ARCH="amd64"
15 fi
16
17 DEPSOK="${ABS_PATH}/.depsok"
18
19 MIRROR_DIR="${ABS_PATH}/aptly"
20 PKGL_LB="${MIRROR_DIR}/lb_packages.list"
21 PKGL_DIR="${ABS_PATH}/config/package-lists"
22 PKGL_STR="$( cat ${PKGL_LB} ${PKGL_DIR}/* | grep -v '^#' | sed '/^$/d' | sort |
paste -s -d'|' | sed 's/|/ | /g' )" FILTER="Priority (required) | Priority
(important) | Priority (standard) | ${PKGL_STR}"
23
```

```
OPTS="(update | img | clean | purge)"

# - - - - -


err() {
  echo
  echo "  [*] Error: $1"
}


usage() {
  echo
  echo
"------------------------------------------------------------------------------
------"
  echo
  echo "  Usage: $0 -o $OPTS"
  echo
  echo "  Run 'update' first if executing for the first time. Must be connected
to the Internet."
  echo "  Run 'img' to build ISO/HDD file. Inbetween builds, run 'clean' or
'purge'."
  echo
  echo
"------------------------------------------------------------------------------
------"
  echo
  exit 1
}


setmirror() {
  cat ${MIRROR_DIR}/aptly.conf | sed "s|\(^ *\"rootDir\": \).*|\1\"${
MIRROR_DIR}\",|" > /root/.aptly.conf
}


imgbuild() {
  setmirror

  # check if repo key is in place
  if ! [ -f ${ABS_PATH}/config/archives/aptly.key ]; then
    gpg --export --armor > ${ABS_PATH}/config/archives/aptly.key
  fi

  # create and publish mirror snapshots
  aptly snapshot create stretch-main-spei from mirror stretch-main
  aptly snapshot create stretch-updates-spei from mirror stretch-updates
  aptly snapshot create stretch-security-spei from mirror stretch-security

```

```
  aptly snapshot merge -latest stretch-final-spei stretch-main-spei stretch-updates-spei stretch-security-spei
  aptly publish snapshot -distribution=stretch stretch-final-spei

  # ensure mirror is not yet running, then run it
  kill $( pgrep -f "aptly serve" ) 2> /dev/null
  aptly serve &

  # run build
  lb build

  # stop mirror and wipe snapshots
  kill $( pgrep -f "aptly serve" ) 2> /dev/null

  aptly publish drop stretch

  aptly snapshot drop stretch-final-spei
  aptly snapshot drop stretch-security-spei
  aptly snapshot drop stretch-updates-spei
  aptly snapshot drop stretch-main-spei
}


update() {
  if ! [ -d /root/.gnupg ]; then
    rngd -r /dev/urandom
    gpg --gen-key --batch ${MIRROR_DIR}/genkey.unattended
    gpg --no-default-keyring --keyring /usr/share/keyrings/debian-archive-keyring.gpg --export | gpg --no-default-keyring --keyring trustedkeys.gpg --import
    killall rngd
  fi

  setmirror

  mirrors="$( aptly mirror list | grep '^ * ' | sed 's/.*\[\([A-Za-z-]*\).*/\1/' )"

  echo "$mirrors" | grep stretch-main      &> /dev/null && aptly mirror drop stretch-main
  echo "$mirrors" | grep stretch-updates   &> /dev/null && aptly mirror drop stretch-updates
  echo "$mirrors" | grep stretch-security  &> /dev/null && aptly mirror drop stretch-security

  aptly mirror create -architectures=${REPO_ARCH} -filter="$FILTER" -filter-with-deps stretch-main http://ftp.br.debian.org/debian/ stretch main contrib non-free
  aptly mirror create -architectures=${REPO_ARCH} -filter="$FILTER" -filter-with-deps stretch-updates http://ftp.br.debian.org/debian/ stretch-updates main contrib non-free
  aptly mirror create -architectures=${REPO_ARCH} -filter="$FILTER" -filter
```

```
-with-deps stretch-security http://security.debian.org/debian-security/
stretch/updates main contrib non-free
109
110   aptly mirror update stretch-main
111   aptly mirror update stretch-updates
112   aptly mirror update stretch-security
113 }
114
115
116 clean () {
117   rm -rf ${LB_DISK}
118   [ -n "$1" ] && lb clean --purge || lb clean
119 }
120
121
122 deps() {
123   # add necessary repository sections & update
124   sed -i 's/\(main\) *$/\1 contrib non-free/' /etc/apt/sources.list
125   apt-get update
126
127   apt-get -y install --no-install-recommends ${LINUX_HEADERS} live-build mbr
netcat-traditional syslinux aptly rng-tools dirmngr
128 }
129
130
131 # - - - - -
132
133
134 if [ $( id -u ) -ne 0 ]; then
135   err "$0 must be run as root. Aborting..."
136   exit 1
137 fi
138
139 if ! uname -r | egrep '686-pae$|amd64$' &> /dev/null; then
140   err "Must run on '686-pae' or 'amd64' kernel archs. Aborting..."
141   exit 1
142 fi
143
144 while getopts ":o:" opt; do
145   case "$opt" in
146     o)
147       option=${OPTARG}
148       ;;
149     *)
150       usage
151       ;;
152   esac
153 done
154
155 [ -z $option ] && { err "No option specified, aborting."; usage; }
156
```

```
157 # check deps
158 if [ ! -f ${DEPSOK} ]; then
159   if [ $( which lb ) ] && [ $( which install-mbr ) ] && [ $( which nc ) ] && [
$( which syslinux ) ] && [ $( which aptly ) ] && [ $( which rngd ) ] && [ $( which
dirmngr ) ]; then
160     echo "All dependencies met. Continuing..."
161   else
162     echo "Missing dependencies. Installing..."
163     deps
164   fi
165
166   touch ${DEPSOK}
167 fi
168
169 case ${option} in
170   "img")
171     mirrors="$( aptly mirror list | grep '^ * ' | sed 's/.*\[\([A-Za-z-
]*\).*/\1/' )"
172     ! echo "$mirrors" | grep stretch-main &> /dev/null && { err "No 'stretch-
main' mirror detected, run '$0 -o update' first."; usage; }
173     ! echo "$mirrors" | grep stretch-updates &> /dev/null && { err "No
'stretch-updates' mirror detected, run '$0 -oupdate' first."; usage; }
174     ! echo "$mirrors" | grep stretch-security &> /dev/null && { err "No
'stretch-security' mirror detected, run '$0 -o update' first."; usage; }
175
176     imgbuild
177     ;;
178   "update")
179     update
180     ;;
181   "clean")
182     clean
183     ;;
184   "purge")
185     clean all
186     ;;
187   *)
188     usage
189     ;;
190 esac
```

O *script* acima é relativamente complexo, então convidamos o aluno a estudá-lo atentamente. Em linhas gerais, o objetivo é automatizar o uso e criação de repositórios locais usando o Aptly antes de iniciar o *build* de um sistema customizado. Também há a checagem de dependências dos pacotes necessários ao funcionamento do *live-build* e do Aptly, conjuntamente.

4. Vamos agora gerar a lista de pacotes que o Aptly deve manter localmente para acelerar a construção do sistema customizado. Crie um diretório de nome `aptly` dentro da pasta atual:

```
# mkdir /live/aptly-basic/aptly
```

Dentro dele, crie o arquivo novo `/live/aptly-basic/aptly/aptly.conf` com o seguinte conteúdo:

```
{
  "rootDir": "/live/aptly-basic/aptly",
  "downloadConcurrency": 4,
  "downloadSpeedLimit": 0,
  "architectures": [],
  "dependencyFollowSuggests": false,
  "dependencyFollowRecommends": false,
  "dependencyFollowAllVariants": false,
  "dependencyFollowSource": false,
  "dependencyVerboseResolve": false,
  "gpgDisableSign": false,
  "gpgDisableVerify": false,
  "gpgProvider": "gpg",
  "downloadSourcePackages": false,
  "skipLegacyPool": true,
  "ppaDistributorID": "ubuntu",
  "ppaCodename": "",
  "skipContentsPublishing": false,
  "FileSystemPublishEndpoints": {},
  "S3PublishEndpoints": {},
  "SwiftPublishEndpoints": {}
}
```

Basicamente, no arquivo acima definimos a raiz do repositório local que será criado, bem como quais pacotes serão baixados pelo Aptly (se apenas dependências básicas, ou também pacotes recomendados/sugeridos).

5. Agora, crie o arquivo novo `/live/aptly-basic/aptly/genkey.unattended` com o seguinte conteúdo:

```
    Key-Type: default
    Subkey-Type: default
    Name-Real: ESR
    Name-Email: suporte@esr.rnp.br
    Expire-Date: 0
    %no-protection
    %commit
    %echo done
```

Todos os pacotes mantidos no repositório local do Aptly serão assinados com um par de chaves criado sob demanda — as informações de geração das chaves são definidas no arquivo acima.

6. Crie o arquivo novo `/live/aptly-basic/aptly/lb_packages.list` com o seguinte conteúdo:

```
```

```
adduser
apt
apt-utils
base-files
base-passwd
bash
bsdmainutils
bsdutils
busybox
coreutils
cpio
cron
dash
dbus
dctrl-tools
debconf
debconf-i18n
debian-archive-keyring
debianutils
diffutils
dmidecode
dmsetup
dosfstools
dpkg
e2fslibs
e2fsprogs
extlinux
findutils
firmware-linux-free
gcc-6-base
gnupg
gnupg-agent
gpgv
grep
grub-common
grub-efi-amd64-bin
grub-efi-ia32-bin
gzip
hdmi2usb-fx2-firmware
hostname
ifupdown
init
initramfs-tools
initramfs-tools-core
init-system-helpers
iproute2
iptables
iputils-ping
irqbalance
isc-dhcp-client
isc-dhcp-common
```

```
isolinux
ixo-usb-jtag
keyboard-configuration
klibc-utils
kmod
krb5-locales
libacl1
libapparmor1
libapt-inst2.0
libapt-pkg5.0
libassuan0
libattr1
libaudit1
libaudit-common
libblkid1
libbsd0
libbz2-1.0
libc6
libcap2
libcap-ng0
libc-bin
libc-l10n
libcomerr2
libcryptsetup4
libdb5.3
libdbus-1-3
libdebconfclient0
libdevmapper1.02.1
libdns-export162
libedit2
libelf1
libestr0
libexpat1
libfastjson4
libfdisk1
libffi6
libgcc1
libgcrypt20
libgdbm3
libglib2.0-0
libglib2.0-data
libgmp10
libgnutls30
libgpg-error0
libgssapi-krb5-2
libhogweed4
libicu57
libidn11
libidn2-0
libip4tc0
libip6tc0
```

```
libiptc0
libisc-export160
libk5crypto3
libkeyutils1
libklibc
libkmod2
libkrb5-3
libkrb5support0
libksba8
liblocale-gettext-perl
liblogging-stdlog0
liblognorm5
liblz4-1
liblzma5
libmnl0
libmount1
libncurses5
libncursesw5
libnetfilter-conntrack3
libnettle6
libnewt0.52
libnfnetlink0
libnpth0
libnuma1
libp11-kit0
libpam0g
libpam-modules
libpam-modules-bin
libpam-runtime
libpcre3
libpipeline1
libpopt0
libprocps6
libpsl5
libreadline7
librsvg2-bin
libseccomp2
libselinux1
libsemanage1
libsemanage-common
libsepol1
libslang2
libsmartcols1
libsqlite3-0
libss2
libssl1.0.2
libssl1.1
libstdc++6
libsystemd0
libtasn1-6
libtext-charwidth-perl
```

```
libtext-iconv-perl
libtext-wrapi18n-perl
libtinfo5
libudev1
libunistring0
libustr-1.0-1
libuuid1
libx11-6
libx11-data
libxapian30
libxau6
libxcb1
libxdmcp6
libxext6
libxml2
libxmuu1
libxtables12
linux-base
linux-image-4.9.0-8-amd64
linux-image-amd64
live-boot
live-boot-doc
live-boot-initramfs-tools
live-config
live-config-doc
live-config-systemd
live-tools
locales
login
logrotate
lsb-base
mawk
mount
multiarch-support
nano
ncurses-base
ncurses-bin
netbase
openssh-client
parted
passwd
perl-base
pinentry-curses
procps
readline-common
rsync
rsyslog
sed
sensible-utils
sgml-base
shared-mime-info
```

```
squashfs-tools
sudo
syslinux
syslinux-common
systemd
systemd-sysv
sysvinit-utils
tar
tasksel
tasksel-data
tzdata
udev
user-setup
util-linux
uuid-runtime
vim-common
vim-tiny
wget
whiptail
xauth
xdg-user-dirs
xml-core
xorriso
xxd
zlib1g
zsync
```

A lista acima foi construída a partir da lista de pacotes instalados automaticamente no sistema-base (que copiamos no passo 8 da atividade anterior), bem como através de tentativa-e-erro durante *builds* consecutivos usando o repositório local. Caso algum pacote essencial esteja faltando, o *build* irá falhar e reclamar que o pacote não está disponível no repositório local — nesse caso, adicionamos o pacote à lista acima e repetimos o *build*, até que não ocorram mais erros.

Se você estiver se perguntando: sim, produzir a lista acima levou UM BOM número de tentativas.

7. Temos que trocar os repositórios a serem usados durante o *build*: ao invés de usar os repositórios `ftp.br.debian.org` e `security.debian.org`, iremos usar o Aptly local, escutando em `127.0.0.1:8080`:

```
# sed -i 's|http://ftp\.br\.debian\.org/debian/|http://127\.0\.0\.1:8080/|'
/live/aptly-basic/config/bootstrap
```

```
# sed -i 's|http://security\.debian\.org/|http://127\.0\.0\.1:8080/|' /live/aptly-
basic/config/bootstrap
```

```
# sed -i 's|http://httpredir\.debian\.org/debian/|http://127\.0\.0\.1:8080/|'
/live/aptly-basic/config/bootstrap
```

```
# sed -i 's|http://ftp\.br\.debian\.org/debian/|http://127\.0\.0\.1:8080/|'
/live/aptly-basic/config/build
```

Uma vez que todos os repositórios serão concatenados em um único, gerenciado localmente pelo Aptly, não precisamos incluir as seções `security` ou `updates` no *chroot*:

```
# sed -i 's|^\(LB_SECURITY=\).*|\1\"false\"|' /live/aptly-basic/config/chroot
```

```
# sed -i 's|^\(LB_UPDATES=\).*|\1\"false\"|' /live/aptly-basic/config/chroot
```

Finalmente, vamos desabilitar a instalação de pacotes recomendados, bem como a checagem de confiança da chave de assinatura dos pacotes no repositório local (já que iremos usar uma chave auto-assinada):

```
# sed -i 's|^\(LB_APT_RECOMMENDS=\).*|\1\"false"|' /live/aptly-basic/config/common
```

```
# sed -i 's|^\(LB_APT_SECURE=\).*|\1\"false"|' /live/aptly-basic/config/common
```

```
# sed -i 's|^\(LB_APT_SOURCE_ARCHIVES=\).*|\1\"false"|' /live/aptly-
basic/config/common
```

```
# sed -i 's|^\(APT_OPTIONS.*\)\"|\1 -o Acquire::ForceIPv4=true\"|' /live/aptly-
basic/config/common
```

```
# sed -i 's|^\(DEBOOTSTRAP_OPTIONS.*\)\"|\1--no-check-gpg\"|' /live/aptly-
basic/config/common
```

8. Algumas das coisas que mais ocupam espaço em instalações minimalistas — além de bibliotecas e *drivers* essenciais — são artefatos como páginas de manual, documentação e *locales* (traduções de *strings* para diferentes linguagens). Não precisamos de nada disso em nosso sistema!

   Crie o arquivo novo `/live/aptly-basic/config/hooks/normal/0450-stripped.hook.chroot` com o seguinte conteúdo:

```
#!/bin/sh

set -e

# remover pacotes desnecessarios
for PACKAGE in apt-utils aptitude man-db manpages info wget dselect
do
    if ! apt-get remove --purge --yes "${PACKAGE}"
    then
        echo "WARNING: ${PACKAGE} isn't installed"
    fi
done

# limpar a cache apt por completo
apt-get autoremove --yes || true
apt-get clean
find /var/cache/apt/ -type f -exec rm -f {} \;
find /var/lib/apt/lists/ -type f -exec rm -f {} \;

# remover arquivos temporarios
find . -name *~ -print0 | xargs -0 rm -f

# remover locales ! en/pt-br
find /usr/share/locale -maxdepth 1 -type d -not -regex
'.*\(locale\|en\|pt_BR\)$' | xargs rm -rf

# remover paginas de manual e documentacao
rm -rf /usr/share/groff/*
rm -rf /usr/share/doc/*
rm -rf /usr/share/man/*
rm -rf /usr/share/info/*
rm -rf /usr/share/lintian/*
rm -rf /usr/share/linda/*
rm -rf /var/cache/man/*

# truncar logs
for FILE in $(find /var/log/ -type f)
do
    : > ${FILE}
done
```

O *script* acima será executado ao final do passo de *chroot*, e irá remover boa parte dos arquivos que não são integralmente necessários ao funcionamento do sistema, reduzindo o tamanho da imagem final consideravelmente.

9. Ufa! Chega de configurações — vamos atualizar o repositório local:

```
# cd /live/aptly-basic/ ; bash makebuild.sh -o update
```

O *script* irá detectar dependências faltantes e instalá-las, e posteriormente irá baixar todos os pacotes que mapeamos no passo (6) desta atividade para o repositório APT local.

10. Uma vez concluído o download, rode novamente o *build* do sistema customizado:

```
# cd /live/aptly-basic/ ; date > buildtime ; bash makebuild.sh -o img  ; date >>
buildtime
```

O Aptly irá publicar um repositório local com todos os pacotes que baixamos no passo anterior, e logo a seguir o comando `lb build` será invocado. Note como a velocidade de obtenção dos pacotes é significativamente superior, desta vez.

Note que criamos um arquivo `/live/aptly-basic/buildtime` para registrar o tempo de *build*, desta vez. Confira seu conteúdo:

```
# cat /live/aptly-basic/buildtime
dom nov 18 01:24:35 -02 2018
dom nov 18 01:26:56 -02 2018
```

No mesmo sistema que produziu o *build* da atividade (3), note que o tempo caiu de cerca de seis minutos para, agora, cerca de 2 minutos e 20 segundos. Significativo, não? E outra vantagem — se precisarmos refazer o *build*, todos os pacotes já estão na *cache* local, e não precisam ser baixados novamente!

11. Vamos ver se nossas otimizações de espaço surtiram efeito no tamanho da imagem:

```
# du -sh /live/aptly-basic/live-image-amd64.hybrid.iso
129M    /live/aptly-basic/live-image-amd64.hybrid.iso
```

De 216 MB na imagem anterior, temos agora uma imagem equivalente de tamanho igual a 129 MB, apenas com a remoção de documentação, *locales* e outros arquivos acessórios. Uma redução de 40%!

5) Construindo uma imagem mais... divertida?

É bem verdade que apesar de extremamente enxuto, nosso sistema customizado não faz nada... interessante, até aqui. Para incrementar suas funcionalidades, vamos produzir um sistema que possua um ambiente gráfico e um navegador.

1. Entre na pasta `/live/aptly-basic` e limpe os arquivos de trabalho do *build* anterior. Vamos usá-la como base para nossa próxima imagem.

```
# cd /live/aptly-basic/ ; bash makebuild.sh -o purge
[2018-11-18 01:35:53] lb clean --purge
P: Cleaning chroot
```

Note que o diretório ainda é significativamente grande, em razão da *cache* de pacotes do Aptly:

```
# du -sh /live/aptly-basic/
263M    /live/aptly-basic/
```

2. Copie o diretório `/live/aptly-basic` para um novo `/live/aptly-x`, e entre nesse diretório:

```
# cp -a /live/aptly-basic /live/aptly-x ; cd /live/aptly-x
```

3. Podemos customizar a lista de pacotes instalados em uma imagem do *live-build* criando arquivos de pacotes no diretório `config/package-lists`. Crie o arquivo novo `/live/aptly-x/config/package-lists/my.list.chroot` com o seguinte conteúdo:

```
 1 feh
 2 firefox-esr
 3 fluxbox
 4 initramfs-tools
 5 keyboard-configuration
 6 live-tools
 7 locales
 8 nodm
 9 openssl
10 rsync
11 sudo
12 task-desktop
13 user-setup
14 uuid-runtime
15 xterm
16 x11-xserver-utils
```

No arquivo acima informamos que, além dos pacotes básicos do sistema *live*, instalaremos também o sistema gráfico X.Org, o gerenciador de janelas Fluxbox e o navegador web Mozilla Firefox, dentre outros pacotes.

4. É também possível customizar quais arquivos estão presentes no *build* final, inserindo-os em uma raiz alternativa no diretório `config/includes.chroot`. Suponha que queiramos que o usuário `user`, ao fazer login no sistema *live*, tenha lançado para si o gerenciador de janelas Fluxbox.

   Crie o caminho de diretórios apropriado:

```
# mkdir -p /live/aptly-x/config/includes.chroot/home/user/
```

   Agora, crie o arquivo `.xinitrc` com a configuração adequada:

```
# echo "startfluxbox" > /live/aptly-x/config/includes.chroot/home/user/.xinitrc
```

Quando iniciarmos o sistema *live*, os diretórios e arquivos que criamos acima já estarão presentes na distribuição, e o gerenciado de login `nodm` se encarregará de iniciar o Fluxbox automaticamente. Esse é um excelente método para distribuir arquivos e configurações em sistemas especialistas, como um servidor web embarcado, por exemplo.

5. Tudo pronto? Vamos atualizar a lista de pacotes do repositório local, já que fizemos várias adições novas no passo (3):

```
# cd /live/aptly-x ; bash makebuild.sh -o update
```

Os pacotes faltantes e suas dependências serão baixados para o *mirror* Aptly local, como esperado.

6. Faça o *build* do sistema customizado:

```
# cd /live/aptly-x/ ; date > buildtime ; bash makebuild.sh -o img  ; date >>
buildtime
```

Vamos ver qual foi o tempo de *build* para esse sistema, um pouco mais complexo que o anterior:

```
# cat /live/aptly-x/buildtime
dom nov 18 01:51:13 -02 2018
dom nov 18 01:55:12 -02 2018
```

Cerca de quatro minutos, muito bom. E quanto ao tamanho?

```
# du -sh /live/aptly-x/live-image-amd64.hybrid.iso
262M    /live/aptly-x/live-image-amd64.hybrid.iso
```

Com 262 MB, o tamanho é superior ao que tínhamos obtido com o sistema básico anterior, mas ainda é certamente muito inferior ao que poderíamos esperar de uma distribuição Linux de propósito geral — especialmente ao considerar que essa imagem inclui sistema gráfico, ambiente de janelas e o navegador Mozilla Firefox.

7. Vamos aos testes. Copie a imagem para sua máquina física usando o comando `scp` ou o programa WinSCP:

```
$ scp aluno@10.0.42.11:/live/aptly-x/live-image-amd64.hybrid.iso
/cygdrive/c/Users/fbs/Desktop/
aluno@10.0.42.11's password:
live-image-amd64.hybrid.iso
100%  262MB  59.6MB/s   00:04
```

Inicie a VM `iso-test` na console principal do Virtualbox. Não é necessário reconfigurá-la, já que o caminho do CD de *boot* aponta para o mesmo arquivo que sobrescrevemos na cópia acima.

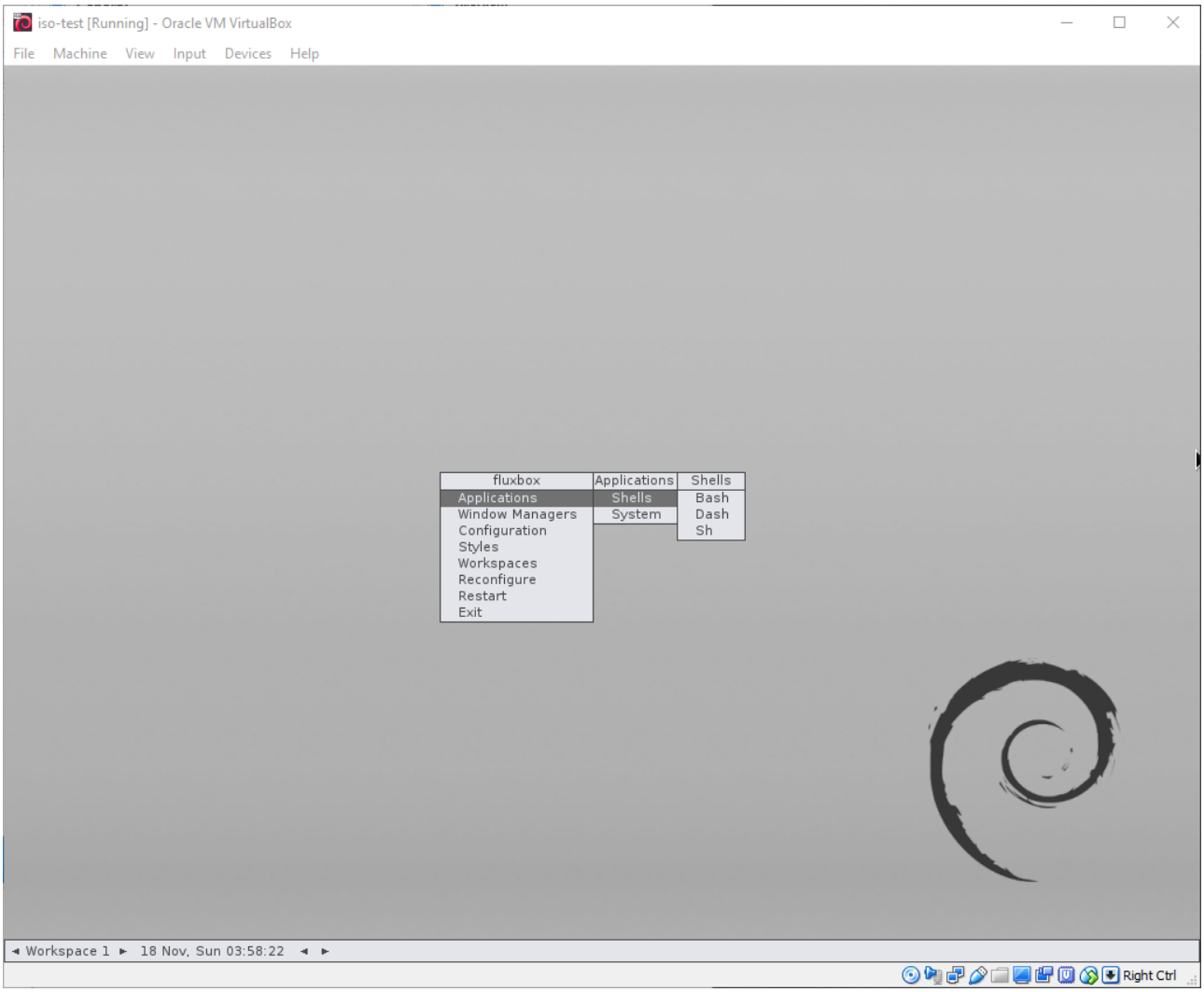

Figura 3. Fluxbox iniciado no sistema live

Legal, não é mesmo? Que tal navegar na Internet? Lance o `xterm` com o atalho `ALT + F1`, e invoque o Mozilla Firefox com o comando `firefox &`:

Figura 4. Navegação no Firefox com o sistema live

Temos aberto acima o site da Escola Superior de Redes, rodando dentro de um sistema customizado com tamanho inferior a 300MB, e que fizemos em alguns poucos passos.

8. Daqui pra frente, seu limite passa a ser sua imaginação:

   - Como poderíamos tornar a imagem acima ainda mais leve e eficiente? Há outros programas e navegadores que seriam mais apropriados para esse tipo de uso?
   - Para quais outras aplicações seria interessante produzir sistemas especialistas como o que fizemos aqui?
   - Que outras opções de segurança e *lockdown* poderíamos ativar em nosso sistema customizado para torná-lo ainda mais seguro?

9. Encerradas as nossas atividades com a máquina `live`, recomenda-se que o aluno a mantenha desligada a partir desta sessão. A grande quantidade de recursos demandada por esse sistema para operar com sucesso a torna um peso muito grande na execução das atividades das próximas sessões.