Como Construir um Sistema Linux Mínimo do Código Fonte

Greg O'Keefe, gcokeefe@postoffice.utas.edu.au v0.9, Novembro 2000

Essas são instruções para construir um sistema linux mínimo do código fonte. Este documento fazia parte do *From PowerUp to Bash Prompt* <http://www.linuxdoc.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html> mas eu os separei para manter ambos os documentos curtos e objetivos. O sistema que vamos construir aqui é *muito* mínimo, e não é adequado para o trabalho de verdade. Se quiser construir um sistema prático do zero, veja o *Linux From Scratch HOWTO* <http://www.linuxfromscratch.org> de Gerard Beekmans em vez deste.

Contents

1 O Que Você Vai Precisar

Vamos instalar uma distribuição de Linux como a Red Hat em uma partição, e usar ela para construir um novo sistema Linux em outra partição. Vou chamar o sistema que estaremos construindo de "alvo" e o sitema que estaremos usando para construi-lo de "fonte" (não confunda com *código fonte* que também estaremos usando.)

De forma que você vai precisar de uma máquina com duas partições separadas. Você pode usar uma instalação de Linux já existente como seu sistema fonte, mas não recomendaria isto. Se você errar um dos parâmetro de um dos comandos que vamos usar, você poderia acidentalmente instalar bobagens em seus sistema. Isso poderia levar à incompatibilidades e conflitos.

Hardware antigo de PC, a maior parte dos 486's e anteriores, tem uma limitação irritante em suas bios. Eles não podem ler o disco rígido depois dos primeiros 512M. Isso não é muito problemático para o Linux, porque uma vez em funcionamento, ele faz seu próprio io (input/output - entrada/saída), ignorando a bios. Porém para Linux que será carregado por essas máquinas velhas, o kernel tem de residir em algum lugar abaixo dos 512M. Se você tem uma dessas máquinas, vai precisar separar uma partição completamente abaixo dos 512M, para montar como `/boot` em qualquer partição que estiver acima dos 512M.

A última vez que fiz isso, usei um Red Hat 6.1 como sistema fonte. Instalei o sistema base mais:

- cpp
- egcs
- egcs-c++
- patch
- make
- dev86
- ncurses-devel
- glibc-devel
- kernel-headers

Eu também tinha o X-window e o Mozilla, assim eu podia ler a documentação fácilmente, mas isto não é realmente necessário. No momento em que terminei o trabalho, isto usou cerca de 350M de espaço em disco. (parece bastante, gostaria de saber porquê?)

O sistema alvo acabado tomou 650M, mas isto inclui todo o código fonte e os arquivos intermediários da construção. Se o espaço for pouco, você deve fazer um `make clean` depois de cada pacote que for construído. Ainda assim é quantidade que causa um pouco de preocupação.

Finalmente, você precisa do código fonte para o sistema que estaremos construindo. Esses são os "pacotes" que eu discuti nesse documento. Eles podem ser obtidos num CD de código fontes, ou na Internet. Eu darei URL's para os sítios nos EUA e para os espelhos na Autrália.

- MAKEDEV *EUA* <ftp://tsx-11.mit.edu/pub/linux/sources/sbin> outro sítio nos *EUA* <ftp://sunsite.unc.edu/pub/Linux/system/admin> .
- Lilo *EUA* <ftp://brun.dyndns.org/pub/linux//lilo/> ,
  Austrália <ftp://mirror.aarnet.edu.au/pub/metalab/system/boot/lilo/> .
- Kernel Linux, prefira usar um dos espelhos listados na *página web* <http://www.kernel.org> ao *EUA* <ftp://ftp.kernel.org/pub/linux/kernel> porque ele está sempre sobrecarregado. *Australia* <ftp://kernel.mirror.aarnet.edu.au/pub/linux/kernel/>
- A GNU libc e o complemento linuxthreads estão em *EUA* <ftp://ftp.gnu.org/pub/gnu/glibc>
  Austrália <ftp://mirror.aarnet.edu.au/pub/gnu/glibc>
- Complementos da GNU libc. Você também vai precisar dos complementos linuxthreads e libcryot. Se libcrypt não estiver lá é por causa de algumas leis de exportação dos EUA. Você pode obtê-lo em *libcrypt* <ftp://ftp.gwdg.de/pub/linux/glibc> . O complemento linuxthreads está junto com a própria libc.
- GNU ncurses *EUA* <ftp://ftp.gnu.org/gnu/ncurses>
  Austrália <ftp://mirror.aarnet.edu.au/pub/gnu/ncurses> .
- SysVinit *EUA* <ftp://sunsite.unc.edu/pub/Linux/system/daemons/init>
  Austrália <ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/daemons/init> .
- GNU Bash *EUA* <ftp://ftp.gnu.org/gnu/bash>
  Austrália <ftp://mirror.aarnet.edu.au/pub/gnu/bash> .
- GNU sh-utils *EUA* <ftp://ftp.gnu.org/gnu/sh-utils>
  Austrália <ftp://mirror.aarnet.edu.au/pub/gnu/sh-utils> .
- util-linux *Algum outro lugar* <ftp://ftp.win.tue.nl/pub/linux/utils/util-linux/>
  Austrália <ftp://mirror.aarnet.edu.au/pub/linux/metalab/system/misc> Esse pacote contém `agetty` e `login`

Para somá-los, você vai precisar:

- Uma máquina com duas partições separadas de aproximadamente 400M e 700M repectivamente apesar que você poderia se virar com menos.
- Uma distribuição de Linux (um CD da Red Hat, por exemplo) e um meio de instalar (ex: um drive de CDROM).
- Os tarballs de código fonte listados acima.

Presumo que você pode instalar o sistema fonte sozinho, sem nenhuma ajuda minha. Daqui em diante presumo que isto já está feito.

A primeira marca nesse pequeno projeto é tornar o kernel inicializável e mostrar uma mensagem "panic" porque ele não pode encontrar um `init`. Isso significa que teremos que instalar um kernel, e o lilo. Contudo para instalar o lilo agradavelmente, precisaremos dos arquivos de dispositivos no diretório `/dev` do alvo. O lilo precisa deles para fazer o acesso de baixo nível necessário para escrever o setor de boot. MAKEDEV é o roteiro (script) que cria esses arquivos de dispositivos. (Você pode apenas copiá-los do sistema fonte é claro, mas isto é trapaça!) Mas antes de tudo, precisamos de um sistema de arquivos para colocar tudo isso nele.

2 O Sistema de Arquivos

Nosso novo sistema estará em um sistema de arquivos. Portanto, primeiro precisamos criar um sistema de arquivos usando `mke2fs`. Em seguida montá-lo em algum lugar. Sugiro `/mnt/target`. No que segue, assumo que este é ele onde está. Você pode poupar um bocado do seu tempo colocando uma entrada em `/etc/fstab` de forma que quando ligar o sistema será montado automaticamente.

Quando inicializarmos o sistema alvo, o que agora está em `/mnt/target` estará em `/`.

Precisamos uma estrutura de diretórios no alvo. Dê uma olhada no FHS (File Heirarchy Standard - Padrão de Hierarquia de Arquivos, veja 13.2 (Links)) para verificar o que isso deveria ser, ou apenas vá para onde o alvo está montado (`cd`) e cegamente faça:

```
mkdir bin boot dev etc home lib mnt root sbin tmp usr var
cd var; mkdir lock log run spool
cd ../usr; mkdir bin include lib local sbin share src
cd share/; mkdir man; cd man
mkdir man1 man2 man3 ... man9
```

Visto que o FHS e a maior parte dos pacotes discordam sobre onde as páginas de manual deveriam ir, precisamos de um vinculo simbólico:

```
cd ../..; ln -s share/man man
```

3 MAKEDEV

Colocaremos o código fonte no diretório `/usr/src` do alvo. Por exemplo, se o sistema de arquivos do alvo está montado em `/mnt/target` e seus tarballs estão em `/root`, você faria:

```
cd /mnt/target/usr/src
tar -xzvf /root/MAKEDEV-2.5.tar.gz
```

Não seja um completo idiota copiando o tarball para o lugar onde você o vai extrair ;->

Normalmente quando você instala software, você instala ele no sistema que está rodando. Contudo nós não queremos fazer isto, queremos instalar como se `/mnt/target` fosse o sistema de arquivos raiz. Diferentes pacotes tem diferentes meios de permitir isso. Para o MAKEDEV você faz:

```
ROOT=/mnt/target make install
```

Você deve procurar por essas opções nos arquivos README e INSTALL ou fazendo um `./configure -help`.

Dê uma olhada no `Makefile` para ver o que ele faz com a variável `ROOT` que definimos neste comando. Em seguida dê uma olhada na página de manual fazendo `man ./MAKEDEV.man` para ver como isto funciona. Você verá que o jeito de criar nossos arquivos de dispositivos é `cd /mnt/target/dev` e fazer `./MAKEDEV generic`. Faça um `ls` para ver todos os maravilhosos arquivos de dispositivos ele fez para você.

4 Kernel

Em seguida construiremos um kernel. Presumo que você já vez isso antes, portanto vou ser breve. É mais fácil instalar o lilo se o kernel que ele deverá inicializar já estiver ali. Volte ao diretório `usr/src` do alvo e descompacte o kernel linux ali. Entre na árvore de diretórios do linux (`cd linux`) e configure o kernel usando seu método favorito, por exemplo `make menuconfig`. Você pode tornar a vida levemente mais fácil para você configurando um kernel sem módulos. Se você configurar algum módulo, então terá de editar o `Makefile`, encontrar `INSTALL_MOD_PATH` e definir ele para `/mnt/target`.

Agora você pode `make dep`, `make bzImage`, e se você configurou módulos: `make modules`, `make modules_install`. Copie o kernel `arch/i386/boot/bzImage` e o mapa do sistema `System.map` para o diretório `/mnt/target/boot` do alvo, e estamos prontos para instalar o lilo.

5 Lilo

O lilo vem com um ótimo roteiro chamado `QuickInst`. Descompacte o código fonte do lilo no diretório de fontes do alvo, rode este roteiro com o comando `ROOT=/mnt/target ./QuickInst`. Ele vai fazer perguntas à você sobre como quer que o lilo seja instalado.

Lembre-se, já que definimos `ROOT` para a partição alvo, iremos informar os nomes relativos a esta. Portanto, quando ele perguntar qual o kernel inicializar por padrão, responda `/boot/bzImage` *não* `/mnt/target/boot/bzImage`. Encontrei um pequeno erro (bug) no roteiro, ele diz:

```
./QuickInst: /boot/bzImage: no such file
```

mas você pode simplismente ignorá-lo.

Onde devemons instruir ao `QuickInst` colocar o setor de boot? Quando reiniciarmos queremos ter a escolha de inicializar no sistema fonte ou no sistema alvo, ou qualquer outro sistema que tivermos. E queremos que o lilo que estamos construindo agora carregue o kernel de nosso novo sistema. Como conseguiremos alcançar ambos os resultados? Vamos divagar um pouco para ver como o lilo inicializa o DOS em um sistema Linux de dupla inicialização (dual boot). O arquivo `lilo.conf` num sistema destes provavelmente é parecido com esse:

```
prompt
timeout = 50
default = linux

image = /boot/bzImage
        label  = linux
        root   = /dev/hda1
        read-only

other = /dev/hda2
        label = dos
```

Se a máquina for configurada deste modo, então o MBR (Master Boot Record - Registro Mestre de Inicialização) é lido e carregado pela bios, ela carrega o inicializador lilo que fornece uma linha de comandos (prompt). Se você digitar `dos` nele, o lilo carrega o setor de boot de hda2, carregando o DOS.

O que faremos é exatamente a mesma coisa, exceto que o setor de boot em hda2 terá um outro lilo nele - é ele que `QuickInst` vai instalar. Assim o lilo da distribuição Linux vai carregar o lilo que construimos, e ele vai carregar o kernel que construimos. Você verá dois lilos quando reiniciar.

Para encurtar a história, quando o `QuickInst` perguntar onde colocar o setor de boot, diga a ele o dispositivo onde seu sistema de arquivos alvo está, por exemplo `/dev/hda2`.

Agora modifique o `lilo.conf` em seu sistema fonte, de forma que tenha uma linha como esta:

```
other = /dev/hda2
        label = target
```

rode o lilo e estaremos apto para inicializar o sistema alvo pela primeira vez.

6 Glibc

Em seguida queremos instalar o `init`, mas como praticamente todos programas que rodam em Linux, o `init` usa funções de bibliotecas fornecidas pela biblioteca GNU C, glibc. Portanto teremos que instalar esta primeiro.

Glibc é um pacote muito grande e complicado. Ela levou 90 horas para compilar em meu velho 386sx/16 com 8M de RAM. Mas levou apenas 33 minutos em meu Celeron 433 com 64M. Penso que a memória é a principal questão aqui. Se você tem apenas 8M de RAM (ou pior, menos!) prepare-se para uma longa compilação.

A documentação de instalação da glibc recomenda compilar em um diretório separado. Isso permite que você recomesse facilmente, apenas removendo este diretório. Você também pode querer fazer isto para poupar cerca de 265M do seu espaço em disco.

Descompacte o tarball `glibc-2.1.3.tar.gz` (ou qualquer que seja a versão) em `/mnt/target/usr/src` como de costume. Agora, precisamos descompactar os adicionais (add-ons) no diretório da glibc. Então `cd glib-2.1.3`, e descompacte os tarballs `glibc-crypt-2-1-3.tar.gz` e `glibc-linuxthreads-2.1.3.tar.gz` aqui.

Agora podemos criar o diretório para compilação, configurar, construir e instalar a glibc. Estes são os comandos que usei, mas leia a documentação você mesmo para ter certeza que você está fazendo o melhor para sua circunstância. Contudo antes que você o faça, você pode querer dar o comando `df` para ver quanto espaço livre você tem. Você pode dar o comando novamente depois que você construir e instalar a glibc, para ver como ele consome espaço.

```
cd ..
mkdir glibc-build
../glibc-2.1.3/configure --enable-add-ons --prefix=/usr
make
make install_root=/mnt/target install
```

Repare que temos outras formas de dizer ao pacote onde instalar.

7 SysVinit

Compilar e instalar binários do SysVinit é simples e direto. Há uma pequena modificação a ser feita no `Makefile` do subdiretório `src/`. Nas últimas quatro linhas, você precisa por `$(ROOT)` exatamente antes de `/dev/initctl`, desta maneira:

```
@ if [ ! -p /dev/initctl ]; then \
```

se torna:

```
@ if [ ! -p $(ROOT)/dev/initctl ]; then \
```

Esse arquivo de dispositivo `initctl` é um meio de comunição com o init. Por exemplo, a página de manual do `init` diz que este arquivo de dispositivo deveria ser usado no lugar de SIGPWR para fazer o init desligar quando a energia estiver falhando, se estivermos usando um no-break. A modificação que acabamos de fazer garante que isso será feito no sistema alvo, não no fonte.

Logo que isto for feito, no subdiretório `src` simplesmente faça:

```
make
ROOT=/mnt/target make install
```

Também existe um porção de roteiros associados com o `init`. Mas você terá que instalá-los manualmente. Eles são definidos em uma hierarquia sob `debian/etc` na árvore do código fonte do SysVinit. Você pode simplesmente copiá-los direto para o diretório `etc` no sistema alvo, com alguma coisa como: `cd ../debian/etc; cp -r * /mnt/target/etc`. Obviamente você vai querer olhá-los antes de copiá-los!

Tudo está pronto para o kernel alvo carregar o `init` quando reiniciarmos. O problema agora é que os roteiros não irão rodar, porque o `bash` não está lá para interpretá-los. O `init` também tentará rodar os `getty`, mas não há `getty` para ele rodar. Reinicie agora e certifique-se de que não há mais nada errado.

8 Ncurses

A próxima coisa que precisamos é do Bash, mas o bash precisa do ncurses, portanto instalaremos ele primeiro. Ncurses substitui o termcap como forma de manipular telas de texto, mas ele também fornece compatibilidade regressiva suportando as chamadas do termcap. No interesse de ter um sistema limpo, simples e moderno, penso que é melhor desativar o velho método termcap. Você pode enfrentar problemas depois se compilar um aplicativo antigo que usa o termcap. Mas pelo menos você saberá o que está usando o que. Se precisar você pode recompilar o ncurses com suporte ao termcap.

Os comandos que usei são:

```
./configure --prefix=/usr --with-install-prefix=/mnt/target --with-shared --disable-termcap
make
make install
```

9 Bash

Fazer o Bash se instalar no lugar devido me custou bastante leitura, reflexão, tentativa e erro. As opções de configuração que usei são:

```
./configure --prefix=/mnt/target/usr/local --exec-prefix=/mnt/target --with-curses
```

Uma vez compilado e instalado o Bash, você precisa criar um vínculo simbólico (symlink) como este: `cd /mnt/target/bin; ln -s bash sh`. Isso é porque os roteiros normalmente tem a primeira linha como essa:

```
#!/bin/sh
```

Se você não fizer o vínculo simbólico, seus roteiros não serão capazes de rodar, porque estarão procurando por `/bin/sh` não `/bin/bash`.

Você poderia reiniciar novamente nesse ponto se quiser. Você deve notar que agora os roteiros realmente rodam, embora você ainda não pode efetuar login, porque não existem os programas `getty` nem `login`.

10 Util-linux (getty e login)

O pacote util-linux contém `agetty` e `login`. Precisamos de ambos para podermos efetuar login e obter o prompt do bash. Depois de instalado, crie um vínculo simbólico de `agetty` para `getty` no diretório `/sbin` no alvo. O `getty` é um dos programas que presupostos a existir em todos os sistemas Unix, por isso o vínculo é uma idéia melhor que modificar o `inittab` para rodar `agetty`.

Ainda tenho um problema não resolvido com a compilação do util-linux. O pacote também contém o programa `more`, e não fui capaz de persuadir o processo do `make` vincular o `more` contra a biblioteca ncurses 5 no sistema alvo em vez da ncurses 4 no sistema fonte. Vou dar uma olhada mais de perto nisso.

Você também vai precisar de um arquivo `/etc/passwd` no sistema alvo. Esse é onde o programa `login` vai verificar para saber se você está autorizado. Já que esse é apenas um sistema experimental nesse estágio, você pode fazer coisas como definir apenas o usuário root, e não solicitar senha!! Apenas coloque isso no `/etc/passwd` do alvo:

```
root::0:0:root:/root:/bin/bash
```

Os campos são separados por dois pontos, e da esquerda para direita eles são o id do usuário, senha (codificada), número do usuário, número do grupo, nome do usuário, diretório pessoal (home) e o shell padrão.

11 Sh-utils

O último pacote que precisamos é o GNU sh-utils. O único programa que precisamos dele nesse estágio é o `stty`, que é usado no `/etc/init.d/rc` que é usado para mudar os níveis de execução (runlevels), e para entrar no nível de execução inicial. Na verdade eu tenho e uso um pacote que contém apenas o `stty`, mas não me lembro de onde ele veio. É melhor idéia usar o pacote GNU, porque existem outras coisas nele que você vai precisar se aumentar o sistema para fazê-lo usável.

Bom isto é tudo. Agora você deve ter um sistema que vai inicializar e pedir um login. Digite "root" e você deverá ter um shell. Você não será capaz de fazer muita coisa com ele. Não há nem mesmo um comando `ls` aqui para você ver seu trabalho manual. Pressione tab duas vezes para ver os comandos disponíveis. Essa foi a coisa mais satisfatória que eu consegui fazer.

12 Customizando

Pode parecer que fizemos um sistema inútil aqui. Mas na verdade não falta muito para que ele seja útil. Uma das primeiras coisas que você teria que fazer é montar o sistema de arquivos raiz para leitura e gravação. Há um roteiro do pacote SysVinit em `/etc/init.d/mountall.sh` que faz isso, e um `mount -a` de forma que tudo é montado da forma que você especificar em `/etc/fstab`. Coloque um vínculo simbólico para ele com um nome parecido com `S05mountall` no diretório `etc/rc2.d` do alvo.

Você pode descobrir que esse roteiro usará comandos que você ainda não instalou. Nesse caso, encontre o pacote que contém os comandos e instale-o. Veja a seção 13.1 (Dicas Aleatórias) para ajudar em como encontrar pacotes.

Dê uma olhada nos outros roteiros em `/etc/init.d`. A maioria deles será necessário em qualquer sistema sério. Vá colocando um por um, certifique-se de que tudo está funcionando bem antes de colocar mais.

Confira o FHS (veja a seção 13.2 (Links)). Ele tem listas dos comandos que deveriam estar em `/bin` e `/sbin`. Tenha certeza de ter todos esses comandos instalados. Ou melhor ainda, encontre a documentação Posix que especifica isso.

Daí em diante, é realmente apenas uma questão de lançar-se em mais e mais pacotes até tudo que você quer estiver lá. Quanto antes você por as ferramentas de construção como `gcc` e `make` melhor. Uma vez feito isto, você pode usar o próprio sistema alvo para construir, o que é muito menos complicado.

13 Mais Informações

13.1 Dicas Aleatórias

Se você tiver um comando chamado `coisa` num sistema Linux com RPM, e quer uma ajuda para saber de onde obter os fontes, você pode usar o comando:

```
rpm -qif ‘which coisa‘
```

E se você tem o CD de fontes do Red Hat, você pode instalar o código fonte usando:

```
rpm -i /mnt/cdrom/SRPMS/o.que.ele.acabou.de.dizer-1.2.srpm
```

Isso vai colocar o tarball e qualquer patch da Red Hat em `/usr/src/redhat/SOURCES`.

13.2 Links

- Há um mini-comofazer sobre compilar software, o *Software Building mini-HOWTO* <http://www.linuxdoc.org/HOWTO/Software-Building.html> .
- Também há um COMO FAZER sobre construir um sistema Linux do zero. Ele focaliza muito mais em construir um sistema usável em lugar de apenas fazer isto como um exercício de aprendizagem. *The Linux From Scratch HOWTO* <http://www.linuxfromscratch.org>
- *Unix File System Standard* <ftp://tsx-11.mit.edu/pub/linux/docs/linux-standards/fsstnd/> outro *link* <http://www.pathname.com/fhs/> para o Padrão de Sistema de Arquivos do Unix. Ele descreve o que deveria ir, onde e porque em um sistema de arquivos Unix. Ele também tem os requisitos mínimos para o conteúdo de `/bin`, `/sbin` e assim por diante. Essa é uma boa referência se seu objetivo é fazer um sistema mínimo mas completo.

14 Administrivia

14.1 Copyright

14.2 Página Web

A versão mais recente deste documento está em *From Powerup To Bash Prompt* <http://learning.taslug.org.au/power2bash>

14.3 Retorno

Gostaria de ouvir quaisquer comentários, críticas ou sugestões de melhoras que você tenha. Por favor, envie-as para *Greg O'Keefe* <mailto:gcokeefe@postoffice.utas.edu.au>

14.4 Agradecimentos

Os nome de produtos são marcas registradas de seus respectivos proprietários, e por meio disto considerado adequadamente reconhecidos.

Existem algumas pessoas que quero agradecer, por ajudar que isto acontecesse.

Michael Emery

Por me lembrar sobre o Unios

Tim Little

Por várias boas dicas sobre `/etc/passwd`

sPaKr on #linux in efnet

Quem me avisou que o syslogd precisa de `/etc/services`, e me apresentou a frase "rolling your own" para descrever a construção de um sistema a partir do código fonte.

Alex Aitkin

Por me chamar a atenção sobre Vico e seu "verum ipsum factum" (o conhecimento vem pela prática).

Dennis Scott

Por corrigir minha aritimética hexadecimal.

jdd

Por me mostrar alguns erros tipográficos.

14.5 Histórico de modificações

14.5.1 0.8 -> 0.9

- Adicionada uma modificação ao makefile do sysvinit. Essa informação é devido a grande fama do "Linux From Scratch" do Gerard Beekmans.

14.5.2 0.8

- Versão inicial. Separado do "From PowerUp to Bash Prompt".

14.6 TODO (Tarefas a fazer)

- Converter para docbook.