# BusyBox 简化嵌入式 Linux 系统

BusyBox 是很多标准 Linux® 工具的一个单个可执行实现。BusyBox 包含了一些简单的工具，例如 cat 和 echo，还包含了一些更大、更复杂的工具，例如 grep、find、mount 以及 telnet（不过它的选项比传统的版本要少）；有些人将 BusyBox 称为 Linux 工具里的瑞士军刀。本文将探索 BusyBox 的目标，它是如何工作的，以及为什么它对于内存有限的环境来说是如此重要。

BusyBox 的诞生

BusyBox 最初是由 Bruce Perens 在 1996 年为 Debian GNU/Linux 安装盘编写的。其目标是在一张软盘上创建一个可引导的 GNU/Linux 系统，这可以用作安装盘和急救盘。一张软盘可以保存大约 1.4-1.7MB 的内容，因此这里没有多少空间留给 Linux 内核以及相关的用户应用程序使用。

BusyBox 揭露了这样一个事实：很多标准 Linux 工具都可以共享很多共同的元素。例如，很多基于文件的工具（比如 `grep` 和 `find` ）都需要在目录中搜索文件的代码。当这些工具被合并到一个可执行程序中时，它们就可以共享这些相同的元素，这样 可以产生更小的可执行程序。实际上，BusyBox 可以将大约 3.5MB 的工具包装成大约 200KB 大小。这就为可引导的磁盘和使用 Linux 的嵌入式设备提供了更多功能。我们可以对 2.4 和 2.6 版本的 Linux 内核使用 BusyBox。

**BusyBox 许可证**
BusyBox 是按照 GNU General Public License（GPL）许可证发行的。这意味着如果我们在一个项目中使用 BusyBox，就必须遵守这个许可证。我们可以在 BusyBox Web 站点（请参看本文后面 参 考资料 一节的内容）上看到这个许可证的内容。
BusyBox 团队似乎正忙于监视违反这个许可证的情况。实际上，他们维护了一个 “Hall of Shame” 页面来说明违反者的情况。

BusyBox 是如何工作的？

为了让一个可执行程序看起来就像是很多可执行程序一样，BusyBox 为传递给 C 的 main 函数的参数开发了一个很少使用的特性。回想一下 C 语言的 main 函数的定义如下：

**POSIX 环境**
尽管 BusyBox 的目标 是提供一个相当完整的 POSIX（可移植操作系统接口）环境，这是一个期望，而不是一种需求。这些工具虽然并不完整，但是它们提供了我们期望的主要功能。

### 清单 1. C 的 main 函数

```
int main( int argc, char *argv[] )
```

在这个定义中，`argc` 是传递进来的参数的个数（参数数量），而 `argv` 是一个字符串数组，代表从命令行传递进来的参数（参数向量）。`argv` 的索引 0 是从命令行调用的程序名。

清单 2 给出的这个简单 C 程序展示了 BusyBox 的调用。它只简单地打印 `argv` 向量的内容。

### 清单 2. BusyBox 使用 `argv[0]` 来确定调用哪个应用程序

```
// test.c



#include <stdio.h>
int main( int argc, char *argv[] )
{
  int i;
  for (i = 0 ; i < argc ; i++) {
    printf("argv[%d] = %s/n", i, argv[i]);
  }
  return 0;
}
```

调用这个程序会显示所调用的第一个参数是该程序的名字。我们可以对这个可执行程序重新进行命名，此时再调用就会得到该程序的新名字。另外，我们可以 创建一个到可执行程序的符号链接，在执行这个符号链接时，就可以看到这个符号链接的名字。

### 清单 3. 在使用新命令更新 BusyBox 之后的命令测试

```
$ gcc -Wall -o test test.c
$ ./test arg1 arg2
argv[0] = ./test
argv[1] = arg1
argv[2] = arg2
$ mv test newtest

$ ./newtest arg1

argv[0] = ./newtest
argv[1] = arg1

$ ln -s newtest linktest

$ ./linktest arg
argv[0] = ./linktest
argv[1] = arg
```

BusyBox 使用了符号链接以便使一个可执行程序看起来像很多程序一样。对于 BusyBox 中包含的每个工具来说，都会这样创建一个符号链接，这样就可以使用这些符号链接来调用 BusyBox 了。BusyBox 然后可以通过 `argv[0]` 来调用内部工具。

配置并编译 BusyBox

我们可以从 BusyBox 的 Web 站点上下载最新版本的 BusyBox（请参看 参 考资料 一节的内容）。与大部分开放源码程序一样，它是以一个压缩的 tarball 形式发布的，我们可以使用清单 4 给出的命令将其转换成源代码树。（如果我们下载的版本不是 1.1.1，那就请在这个命令中使用适当的版本号以及特定于这个版本号的命令。）

### 清单 4. 展开 BusyBox

```
$ tar xvfz busybox-1.1.1.tar.gz
```

结果会生成一个目录，名为 busybox-1.1.1，其中包含了 BusyBox 的源代码。要编译默认的配置（其中包含了几乎所有的内容，并禁用了调试功能），请使用 `defconfig` make 目标：

### 清单 5. 编译默认的 BusyBox 配置

```
$ cd busybox-1.1.1

$ make defconfig

$ make

$
```

结果是一个相当大的 BusyBox 映像，不过这只是开始使用它的最简单的方法。我们可以直接调用这个新映像，这会产生一个简单的 Help 页面，里面包括当前配置的命令。要对这个映像进行测试，我们也可以对一个命令调用 BusyBox 来执行，如清单 6 所示。

**BusyBox 源代码树**
BusyBox 的源代码树组织得很好。这些工具都基于它们的用途进行了分类，并存储在单独的子目录中。例如，网络工具和守护进程（如 `httpd`、`ifconfig` 等）都在 ./networking 目录中；标准的模块工具（包括 `insmod`、`rmmod` 和 `lsmod` ）都在 ./modutils 目录中；编辑器（例如 `vi` 和流编辑器，如 `awk` 和 `sed` ）都在 ./editors 目录中。makefile 配置、编译和安装所使用的各个文档都在这个目录树的根目录中。

### 清单 6. 展示 BusyBox 命令的执行和 BusyBox 中的 ash shell

```
$ ./busybox pwd

/usr/local/src/busybox-1.1.1


$ ./busybox ash

/usr/local/src/busybox-1.1.1 $ pwd

/usr/local/src/busybox-1.1.1


/usr/local/src/busybox-1.1.1 $ exit

$
```

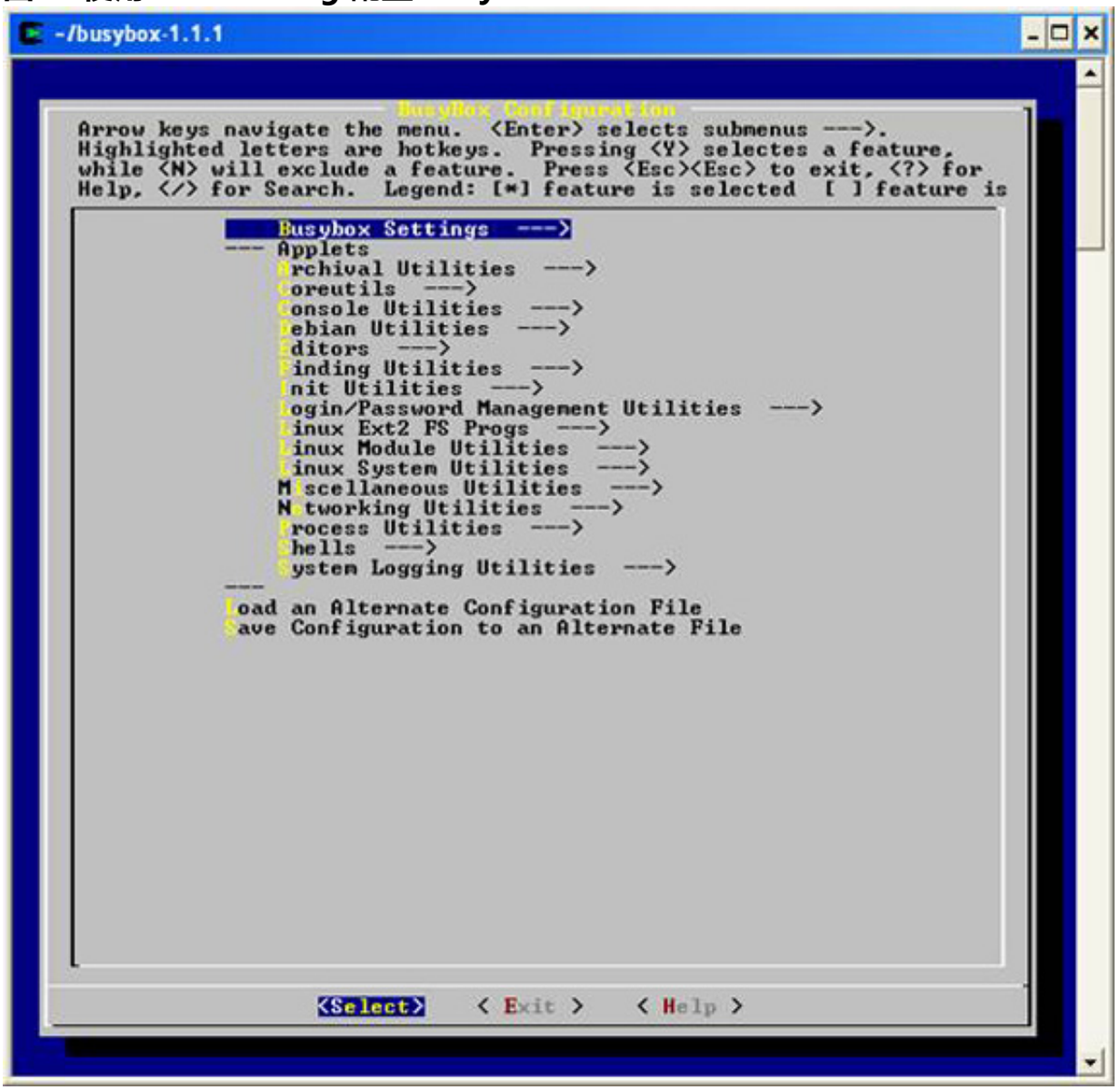
在这个例子中，我们调用了 `pwd` （打印工作目录）命令，使用 BusyBox 进入了 `ash` shell，并在 `ash` 中调用了 `pwd` 。

手工配置

如果您正在构建一个具有特殊需求的嵌入式设备，那就可以手工使用 `menuconfig` make 目标来配置 BusyBox 的内容。如果您熟悉 Linux 内核的编译过程，就会注意到 `menuconfig` 与配置 Linux 内核的内容所使用的目标相同。实际上，它们都采用了相同的基于 ncurses 的应用程序。

使用手工配置，我们可以指定在最终的 BusyBox 映像中包含的命令。我们也可以对 BusyBox 环境进行配置，例如包括对 NSA（美国国家安全代理）的安全增强 Linux（SELinux），指定要使用的编译器（用来在嵌入式环境中进行交叉编译）以及 BusyBox 应该静态编译还是动态编译。图 1 给出了 `menuconfig` 的主界面。在这里我们应该可以看到可以为 BusyBox 配置的不同类型的应用程序（applet）。

**图 1. 使用 menuconfig 配置 BusyBox**




要手工配置 BusyBox，请使用下面的命令：

**清单 7. 手工配置 BusyBox**

**多体系结构支持**
可以简单地为 BusyBox 指定交叉编译器意味着我们可以为很多体系结构编译 BusyBox。要为您的

```
$ make menuconfig

$ make

$
```

目标体系结构编译 BusyBox，我们需要一个交叉编译器和一个已经为特定目标体系结构编译好的 C 库（uClibc 或 glibc）。

这为我们提供了可以调用的 BusyBox 的二进制文件。下一个步骤是围绕 BusyBox 构建一个环境，包括将标准 Linux 命令重定向到 BusyBox 二进制文件的符号链接。我们可以使用下面的命令简单地完成这个过程：

**清单 8. 构建 BusyBox 环境**

```
$ make install

$
```

默认情况下，这会创建一个新的本地子目录 _install，其中包含了基本的 Linux 环境。在这个根目录中，您会找到一个链接到 BusyBox 的 `linuxrc` 程序。这个 `linuxrc` 程序在构建安装盘或急救盘（允许提前进行模块化的引导）时非常有用。同样是在这个根目录中，还有一个包含操作系统二进制文件的 /sbin 子目录。还有一个包含用户二进制文件的 /bin 目录。在构建软盘发行版或嵌入式初始 RAM 磁盘时，我们可以将这个 _install 目录迁移到目标环境中。我们还可以使用 make 程序的 `PREFIX` 选项将安装目录重定向到其他位置。例如，下面的代码就使用 /tmp/newtarget 根目录来安装这些符号链接，而不是使用 ./_install 目录：

**清单 9. 将符号链接安装到另外一个目录中**

```
$ make PREFIX=/tmp/newtarget install
$
```

使用 `install` make 目标创建的符号链接都来自于 busybox.links 文件。这个文件是在编译 BusyBox 时创建的，它包含了已经配置的命令清单。在执行 `install` 时，就会检查 busybox.links 文件确定要创建的符号链接。

到 BusyBox 的命令行链接也可以使用 BusyBox 在运行时动态创建。`CONFIG_FEATURE_INSTALLER` 选项就可以启用这个特性，在运行时可以这样执行：

**清单 10. 在运行时创建命令链接**

```
$ ./busybox --install -s
```

`-s` 选项强制创建这些符号链接（否则就创建硬链接）。这个选项要求系统中存在 /proc 文件

系统。

BusyBox 编译选项

BusyBox 包括了几个编译选项，可以帮助为我们编译和调试正确的 BusyBox。

**表 1. 为 BusyBox 提供的几个 make 选项**

| make 目标 | 说明 |
|---|---|
| help | 显 示 make 选项的完整列表 |
| defconfig | 启 用默认的（通用）配置 |
| allnoconfig | 禁 用所有的应用程序（空配置） |
| allyesconfig | 启 用所有的应用程序（完整配置） |
| allbareconfig | 启 用所有的应用程序，但是不包括子特性 |
| config | 基 于文本的配置工具 |
| menuconfig | N- curses（基于菜单的）配置工具 |
| all | 编 译 BusyBox 二进制文件和文档（./docs） |
| busybox | 编 译 BusyBox 二进制文件 |
| clean | 清 除源代码树 |
| distclean | 彻底清 除源代码树 |
| sizes | 显示所启用的应 用程序的文本/数据大小 |

在定义配置时，我们只需要输入 make 就可以真正编译 BusyBox 二进制文件。例如，要为所有的应用程序编译 BusyBox，我们可以执行下面的命令：

**清单 11. 编译 BusyBox 二进制程序**

```
$ make allyesconfig
$ make
$
```

压缩 BusyBox

如果您非常关心对 BusyBox 映像的压缩，就需要记住两件事情：

1. 永远不要编译为静态二进制文件（这会将所有需要的库都包含到映像文件中）。相反，如果我们是编译为一个共享映像，那么它会使用其他应用程序使用 的库（例如 /lib/libc.so.X ）。
2. 使用 uClibc 进行编译，这是一个对大小进行过优化的 C 库，它是为嵌入式系统开发的；而不要使用标准的 glibc （GNU C 库）来编译。

BusyBox 命令中支持的选项

BusyBox 中的命令并不支持所有可用选项，不过这些命令都包含了常用的选项。如果我们需要知道一个命令可以支持哪些选项，可以使用 --help 选项来调用这个命令，如清单 12 所示。

### 清单 12. 使用 --help 选项调用命令

```
$ ./busybox wc --help

BusyBox v1.1.1 (2006.04.09-15:27+0000) multi-call binary
Usage: wc [OPTION]... [FILE]...
Print line, word, and byte counts for each FILE, and a total line if
more than one FILE is specified. With no FILE, read standard input.

Options:

        -c      print the byte counts
        -l      print the newline counts
        -L      print the length of the longest line
        -w      print the word counts
$
```

这些特定的数据只有在启用了 CONFIG_FEATURE_VERBOSE_USAGE 选项时才可以使用。如果没有这个选项，我们就无法获得这些详细数据，但是这样可以节省大约 13 KB 的空间。

向 BusyBox 中添加新命令

向 BusyBox 添加一个新命令非常简单，这是因为它具有良好定义的体系结构。第一个步骤是为新命令的源代码选择一个位置。我们要根据命令的类型（网络，shell 等）来选择位置，并与其他命令保持一致。这一点非常重要，因为这个新命令最终会在 menuconfig 的配置菜单中出现（在下面的例子中，是 Miscellaneous Utilities 菜单）。

对于这个例子来说，我将这个新命令称为 newcmd ，并将它放到了 ./miscutils 目录中。这个新命令的源代码如清单 13 所示。

### 清单 13. 集成到 BusyBox 中的新命令的源代码

```
#include "busybox.h"

int newcmd_main( int argc, char *argv[] )
{
  int i;
  printf("newcmd called:/n");
  for (i = 0 ; i < argc ; i++) {
    printf("arg[%d] = %s/n", i, argv[i]);
  }
  return 0;
}
```

接下来，我们要将这个新命令的源代码添加到所选子目录中的 Makefile.in 中。在本例中，我

更新了 ./miscutils/Makefile.in 文件。请按照字母顺序来添加新命令，以便维持与现有命令的一致性：

### 清单 14. 将命令添加到 Makefile.in 中

```
MISCUTILS-$(CONFIG_MT)          += mt.o
MISCUTILS-$(CONFIG_NEWCMD)   += newcmd.o
MISCUTILS-$(CONFIG_RUNLEVEL)    += runlevel.o
```

接下来再次更新 ./miscutils 目录中的配置文件，以便让新命令在配置过程中是可见的。这个文件名为 Config.in，新命令是按照字母顺序添加的：

### 清单 15. 将命令添加到 Config.in 中

```
config CONFIG_NEWCMD
        bool "newcmd"
        default n
        help
          newcmd is a new test command.
```

这个结构定义了一个新配置项（通过 config 关键字）以及一个配置选项（CONFIG_NEWCMD ）。新命令可以启用，也可以禁用，因此我们对配置的菜单属性使用了 bool （Boolean）值。这个命令默认是禁用的（n 表示 No），我们可以最后放上一个简短的 Help 描述。在源代码树的 ./scripts/config/Kconfig-language.txt 文件中，我们可以看到配置语法的完整文法。

接下来需要更新 ./include/applets.h 文件，使其包含这个新命令。将下面这行内容添加到这个文件中，记住要按照字母顺序。维护这个次序非常重要，否则我们的命令就会找不到。

### 清单 16. 将命令添加到 applets.h 中

```
USE_NEWCMD(APPLET(newcmd, newcmd_main, _BB_DIR_USER_BIN, _BB_SUID_NEVER))
```

这定义了命令名（newcmd ），它在 Busybox 源代码中的函数名（newcmd_main ）， 应该在哪里会为这个新命令创建链接（在这种情况中，它在 /usr/bin 目录中），最后这个命令是否有权设置用户 id（在本例中是 no）。

倒数第二个步骤是向 ./include/usage.h 文件中添加详细的帮助信息。正如您可以从这个文件的例子中看到的一样，使用信息可能非常详细。在本例中，我只添加了一点信息，这样就可以编译这个新命令 了：

### 清单 17. 向 usage.h 添加帮助信息

```
#define newcmd_trivial_usage    "None"
```

```
#define newcmd_full_usage       "None"
```

最后一个步骤是启用新命令（通过 `make menuconfig` ，然后在 Miscellaneous Utilities 菜单中启用这个选项）然后使用 `make` 来编译 BusyBox。

使用新的 BusyBox，我们可以对这个新命令进行测试，如清单 18 所示。

### 清单 18. 测试新命令

```
$ ./busybox newcmd arg1

newcmd called:

arg[0] = newcmd

arg[1] = arg1

$ ./busybox newcmd --help
BusyBox v1.1.1 (2006.04.12-13:47+0000) multi-call binary

Usage: newcmd None
None
```

就是这样！BusyBox 开发人员开发了一个优秀但非常容易扩展的工具。

结束语

BusyBox 是为构建内存有限的嵌入式系统和基于软盘系统的一个优秀工具。BusyBox 通过将很多必需的工具放入一个可执行程序，并让它们可以共享代码中相同的部分，从而对它们的大小进行了很大程度的缩减，BusyBox 对于嵌入式系统来说是一个非常有用的工具，因此值得我们花一些时间进行探索。