BusyBox

VEDA Solutions
Inspired by open Source

BusyBox

http://www.busybox.net/

- ✓ Most Unix command line utilities within a single executable!
  It even includes a web server!
- ✓ Sizes less than < 500 KB (statically compiled with uClibc) or less than 1 MB (statically compiled with glibc).
- ✓ Easy to configure which features to include.
- ✓ The best choice for
- ✓ Initramfs / initrd with complex scripts
- ✓ Small and medium size embedded systems

See http://www-128.ibm.com/developerworks/linux/library/l-busybox/ for a nice introduction.

Applet highlight - BusyBox vi

- ✓ If you are using BusyBox, adding vi supports only adds 20K. (built with shared libraries, using uClibc).
- ✓ You can select which exact features to compile in.
- ✓ Users hardly realize that they are using a lightweight vi version!
- ✓ Tip: you can learn vi on the desktop, by running the vimtutor command.

Configuring BusyBox

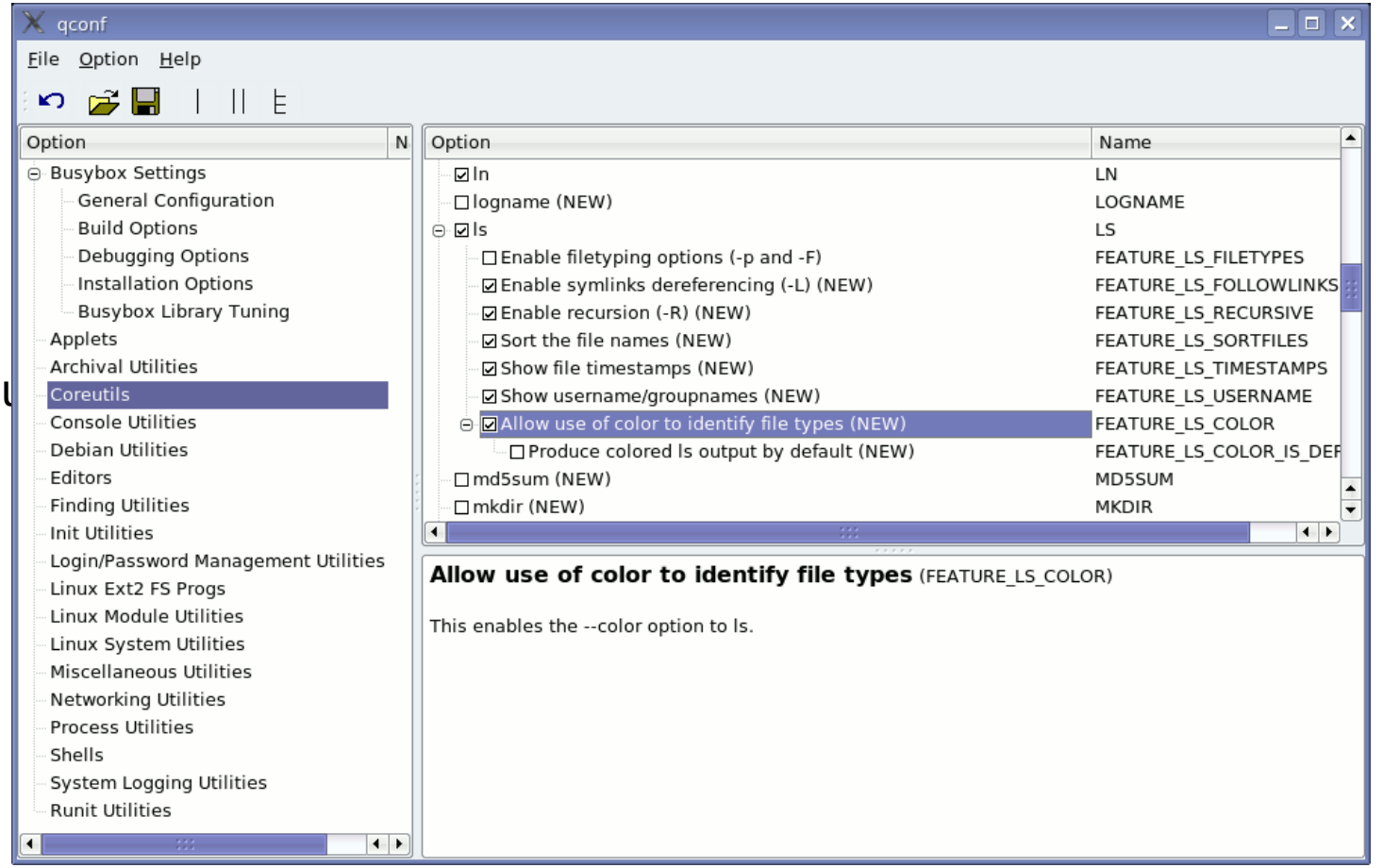
- ✓ Get the latest stable sources from http://busybox.net
- ✓ Configure BusyBox (creates a .config file):
- ✓ make defconfig
  Good to begin with BusyBox.

  Configures BusyBox with all options for regular users.
- ✓ make allnoconfig
  Unselects all options. Good to configure only what you need.
- ✓ make xconfig (graphical) or make menuconfig (text)
  Same configuration interfaces as the ones used by the Linux kernel.

BusyBox make xconfig

You can choose:

- ✓ the commands to compile,
- ✓ and even the command options and features that you need!



Compiling BusyBox

Set the cross-compiler prefix in the configuration interface:
BusyBox Settings -> Build Options -> Cross Compiler prefix
Example: arm-linux-

Set the installation directory in the configuration interface:
BusyBox Settings -> Installation Options -> BusyBox installation prefix

Add the cross-compiler path to the PATH environment variable:
export PATH=/usr/local/arm/3.3.2/bin:$PATH

Compile BusyBox:
make

Install it (this creates a Unix directory structure symbolic links to the busybox executable):
make install

Alternative to BusyBox: embutils

http://www.fefe.de/embutils/
From the creator of diet libc

- A similar set of tiny utilities for embedded systems.

Version 0.19 (Aug. 2008): 90 common commands are implemented.

- Can only be built statically with diet libc!
- Compared to BusyBox: Much less momentum,

user and developer base. Still misses key commands and features (ifconfig, for example)

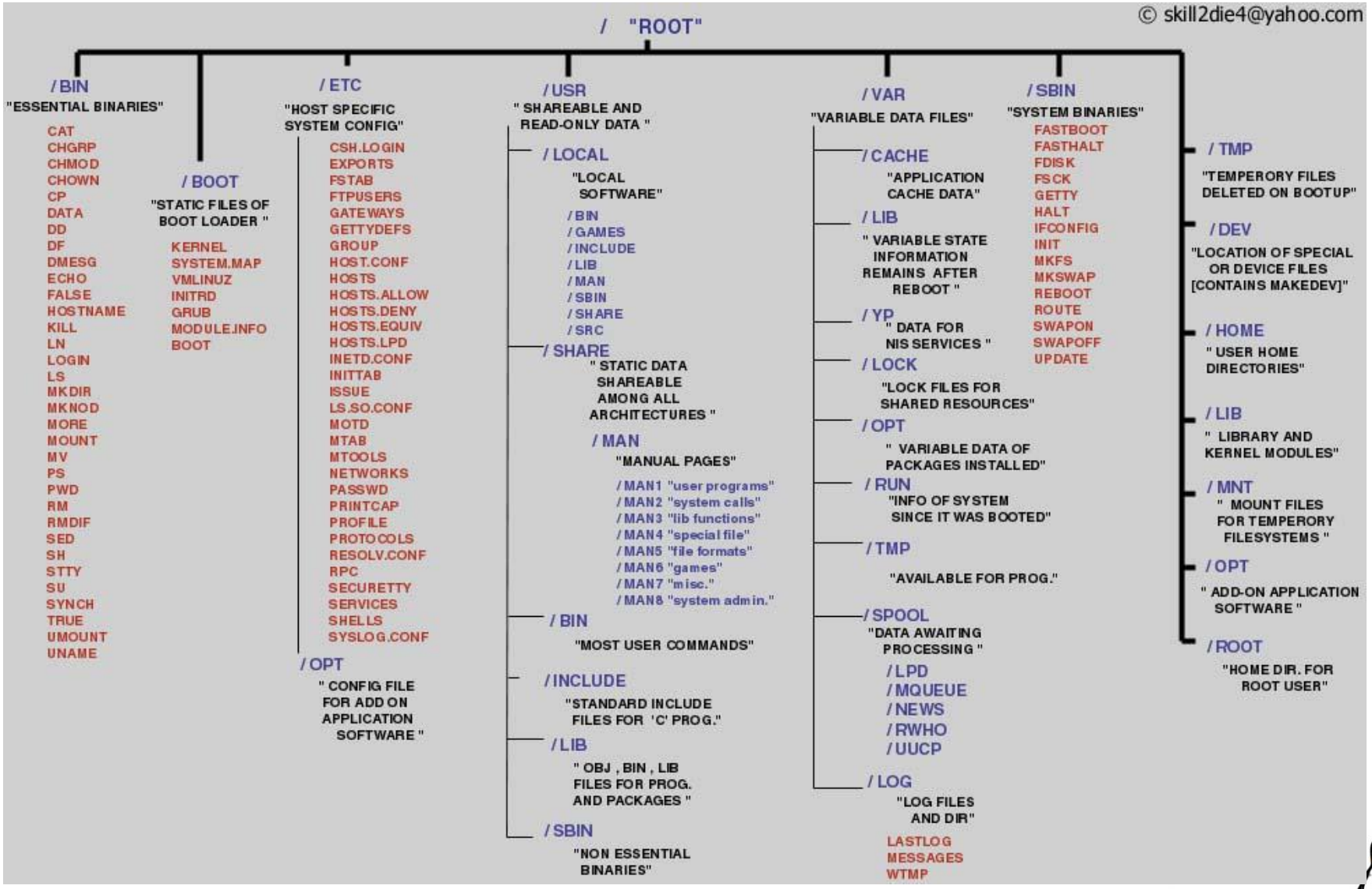
- But can achieve smaller size than BusyBox

on standalone executables.

Linux files structure

VEDA Solutions
Inspired by open Source

Linux files structure



FSSTND : (Filesystem standard)

✓**bin** - Commands needed during booting up that might be needed by normal users

✓**sbin** - Like bin but commands are not intended for normal users. Commands run by LINUX

✓**proc** - This filesystem is not on a disk. It is a virtual filesystem that exists in the kernels imagination which is memory

1 - A directory with info about process number 1. Each process has a directory below proc.

FSSTND : (Filesystem standard)

✓**usr** - Contains all commands, libraries, man pages, games and static files for normal operation.

bin - Almost all user commands. some commands are in /bin or /usr/local/bin.

sbin - System admin commands not needed on the root filesystem. e.g., most server programs.

include - Header files for the C programming language. Should be below /user/lib for consistency.

lib - Unchanging data files for programs and subsystems

local - The place for locally installed software and other files.

man - Manual pages

info - Info documents

doc - Documentation

tmp

X11R6 - The X windows system files. There is a directory similar to usr below this directory.

X386 - Like X11R6 but for X11 release 5

FSSTND : (Filesystem standard)

- ✓**boot** - Files used by the bootstrap loader, LILO. Kernel images are often kept here.
- ✓**lib** - Shared libraries needed by the programs on the root filesystem
- ✓**modules** - Loadable kernel modules, especially those needed to boot the system after disasters.
- ✓**dev** - Device files
- ✓**etc** - Configuration files specific to the machine.
- ✓**skel** - When a home directory is created it is initialized with files from this directory
- ✓**sysconfig** - Files that configure the linux system for devices.

FSSTND : (Filesystem standard)

✓**var -** Contains files that change for mail, news, printers log files, man pages, temp files

file

lib - Files that change while the system is running normally

local - Variable data for programs installed in /usr/local.

lock - Lock files. Used by a program to indicate it is using a particular device or file

log - Log files from programs such as login and syslog which logs all logins and logouts.

run - Files that contain information about the system that is valid until the system is next booted

spool - Directories for mail, printer spools, news and other spooled work.

tmp - Temporary files that are large or need to exist for longer than they should in /tmp.

catman - A cache for man pages that are formatted on demand

✓**mnt** - Mount points for temporary mounts by the system administrator.

✓**tmp** - Temporary files. Programs running after bootup should use /var/tmp

Building the Root File System

VEDA Solutions
Inspired by open Source

Create an ext2 filesystem

Create an Initialized logical file

- ✓ Create a logical file rootfs & filled with zeros by

  dd if=/dev/zero of=/rootfs bs=1k count=4096

- ✓ Making rootfs look like a block device

  losetup /dev/loop0 /rootfs

  where /dev/loop0 is a virtual block device

Create an ext2 fs with the loop device

- Creating an ext2 file system with the loop device

  mkfs -t ext2 /dev/loop0 4096

The file rootfs represents as a logical device (/dev/loop0) with 4MB size

Mount the logical device “rootfs”

- ✓ Create a mount point /mnt and mounting the file system through the loop device

  mount -t ext2 /rootfs /mnt -o loop

- ✓ Change to /mnt and create required directories to build a root file system

Creating Directories

- In /mnt

  mkdir dev etc etc/init.d proc mnt var var/shm tmp

  chmod 755 . dev etc etc/init.d proc mnt var var/shm tmp

- In /mnt/dev
  - Create generic terminal devices
    - mknod tty c 5 0
    - mknod console c 5 1
    - chmod 666 tty console

Creating Directories

- Create a Virtual terminal devices for VGA display
  - mknod tty0 c 4 0
  - chmod 666 tty 0
- Create RAM disk device
  - mknod ram0 b 1 0
  - chmod 660 ram0
- Create null devicess, used to discard unwanted output
  - mknod null c 1 3
  - chmod 666 null
- Create floppy device
  - mknod fd0 b 2 0
  - chmod 666 fd0

Creating Directories

- In /mnt/etc/init.d/
  - Write a script
    - vim rcS

      mount -av /* mount the defualt fs mentioned in /etc/fstab */
  - chmod 744 rcS

Creating Directories

- In /mnt/etc
  - Write a script
    - vim fstab

```
/dev/ram0      /          ext2    defaults
proc           /proc      proc    defaults    0    0
none           /var/shm   shm     defaults    0    0
```

  - chmod 744 fstab

Creating Directories

- In /mnt/etc
  - Write a script
    - vim inittab

      ::sysinit:/etc/init.d/rcS

  - chmod 744 inittab

Convert ext2 into jffs2 filesystem

Create & Mount filesystem for jffs2

- Creating nandflash compatible filesystem for “rootfs”
- For creating jffs2 we need to download mkfs.jffs2 from

  ftp://sources.redhat.com/pub/jffs2/mkfs.jffs2.

To create a jffs2 filesystem

```
mkfs.jffs2 --pad=0x4000 --eraseblock=0x4000 -l --root=/mnt -o   my_file.bin
```

Create & Mount filesystem for jffs2

- To Create a block device mtdblock0 with size 24576 and erase size 128

  modprobe mtdram

  modprobe mtdram total_size=24576 erase_size=128

  modprobe mtdblock

  dd if=/my_file.bin of=/dev/mtdblock0

- Mounting mtdblock as jffs2 filesystem

  mount -t jffs2 /dev/mtdblock0 /target_fs

Exporting filesystem through NFS

VEDA Solutions
Inspired by open Source

Export the /target_fs

- Export the filesystem through NFS
  - vim /etc/exports

  /target_fs192.168.1.1 (rw,sync) /* 192.168.1.1 is the target board address */

- Restart the NFS

  service nfs restart

Uboot Commands

- Set the uboot environment variables
- Server ip
  - Setenv serverip 192.168.1.30
- Target ip
  - Setenv ipaddr 192.168.1.1
- Boot arguments
  - Setenv bootargs console=ttyS0,115200 root=/dev/nfs fsroot=192.168.1.50:/target_fs ip=192.168.1.1

Uboot Commands

- Transferring “uImage” from host to target through tftp server
  - tftpboot 0x21000000 uImage
- Boot the uImage from loadded address
  - bootm 0x21000000
- Copy executable from host to target
  - tftp -g -r a.out 192.168.1.30