ktest.pl – tutorial
(Embedded Edition)

Steven Rostedt
srostedt@redhat.com
rostedt@goodmis.org

4096R/5A56DE73
5ED9 A48F C54C 0A22 D1D0
804C EBC2 6CDB 5A56 DE73

What is ktest.pl?

- A script written in perl
  - But you do not need to know perl!
- Written to automate building, installing, booting and testing kernels
- Tests sets of commits in git
- normal building of kernel (also randconfig)
- bisect (git bisect and config bisect)
- make_min_config

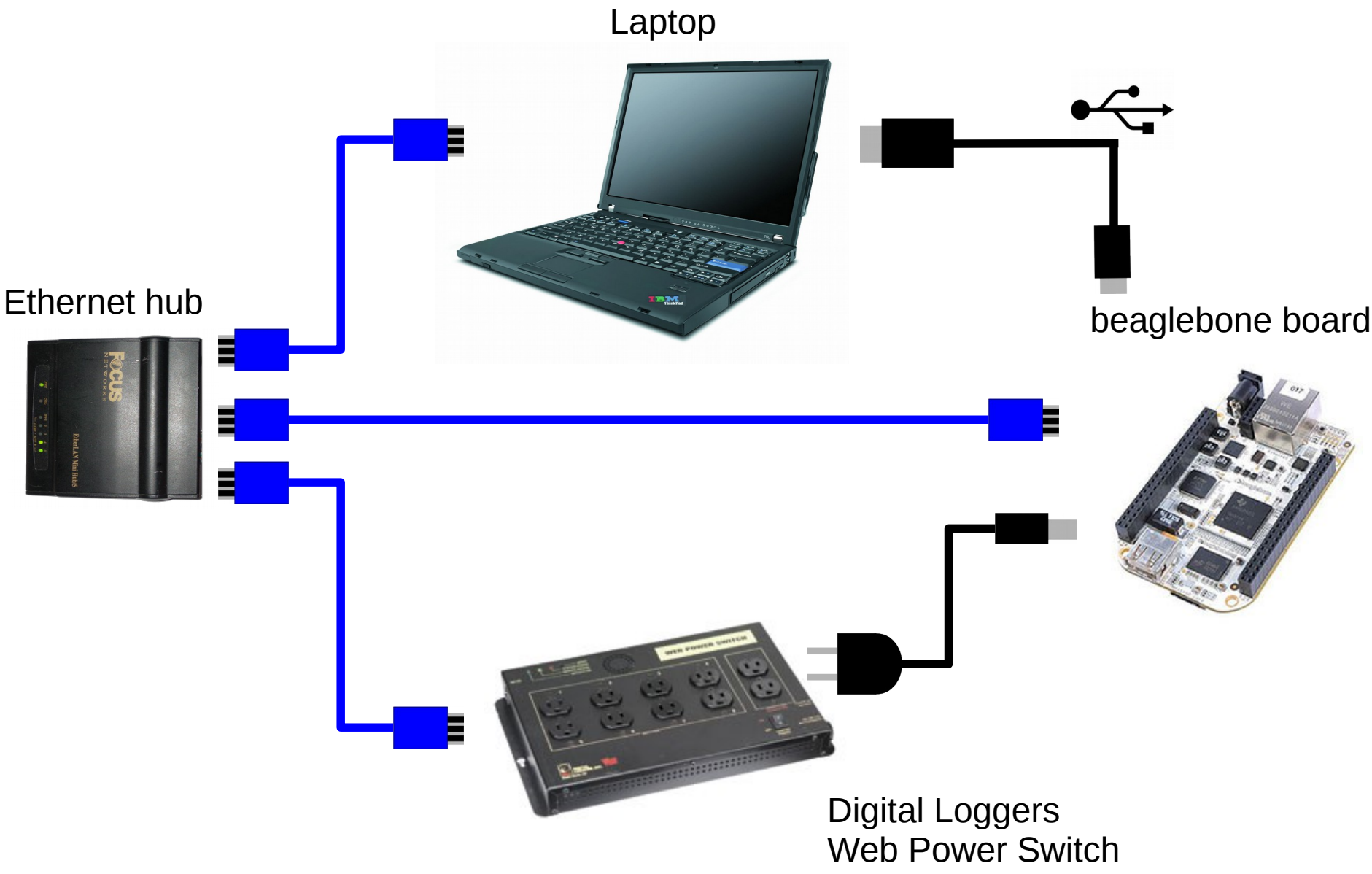
Where is it?

- From Linux 2.6.38 (best to use the latest)
  - tools/testing/ktest
- ktest.pl
  - The script to run
- samples.conf
  - Explains all config options that ktest.pl uses
- examples/
  - Directory of various config examples

Requirements

- Two machines
    - host
    - target (may be external or virtual machine)
- Host be able to remotely power cycle target
- Boot once kernel (boot back to default)
- Host be able to read target's console
- Source and Build directories must be separate
- Some tests require source to be a git repo
    - May add quilt support

My Setup


Laptop
Ethernet hub
beaglebone board
Digital Loggers
Web Power Switch

Digital Loggers Power Cycle

- Cycle box connected to outlet 1 “outlet?1”

wget --no-proxy -O /dev/null -q --auth-no-challenge 'http://admin:admin@power/outlet?1=**CCL**'

Digital Loggers
Turn off

- Power off box connected to outlet 1

  “outlet?1”

wget --no-proxy -O /dev/null -q --auth-no-challenge 'http://admin:admin@power/outlet?1=**OFF**'

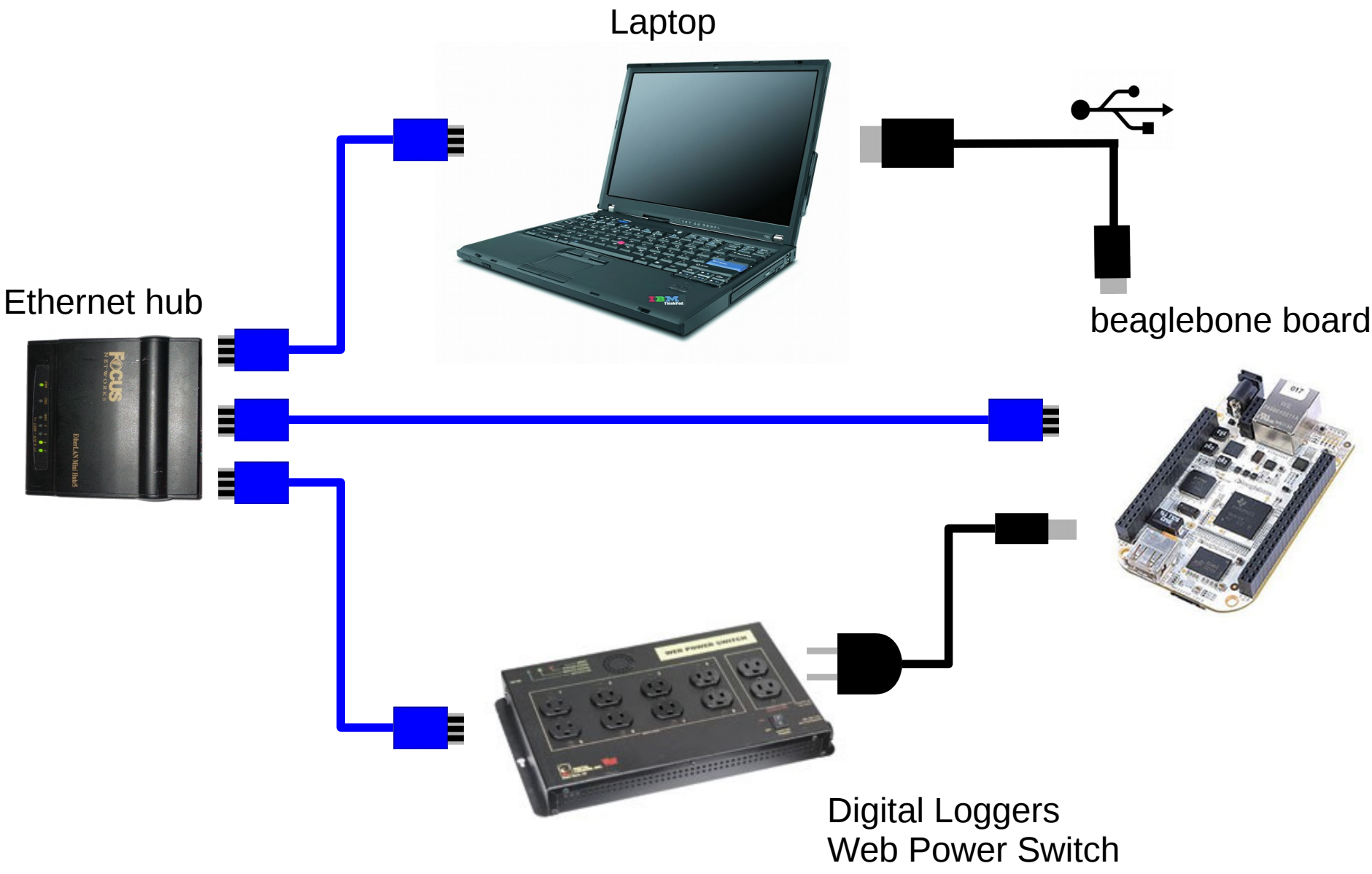
Digital Loggers
Turn on

- Power on box connected to outlet 1

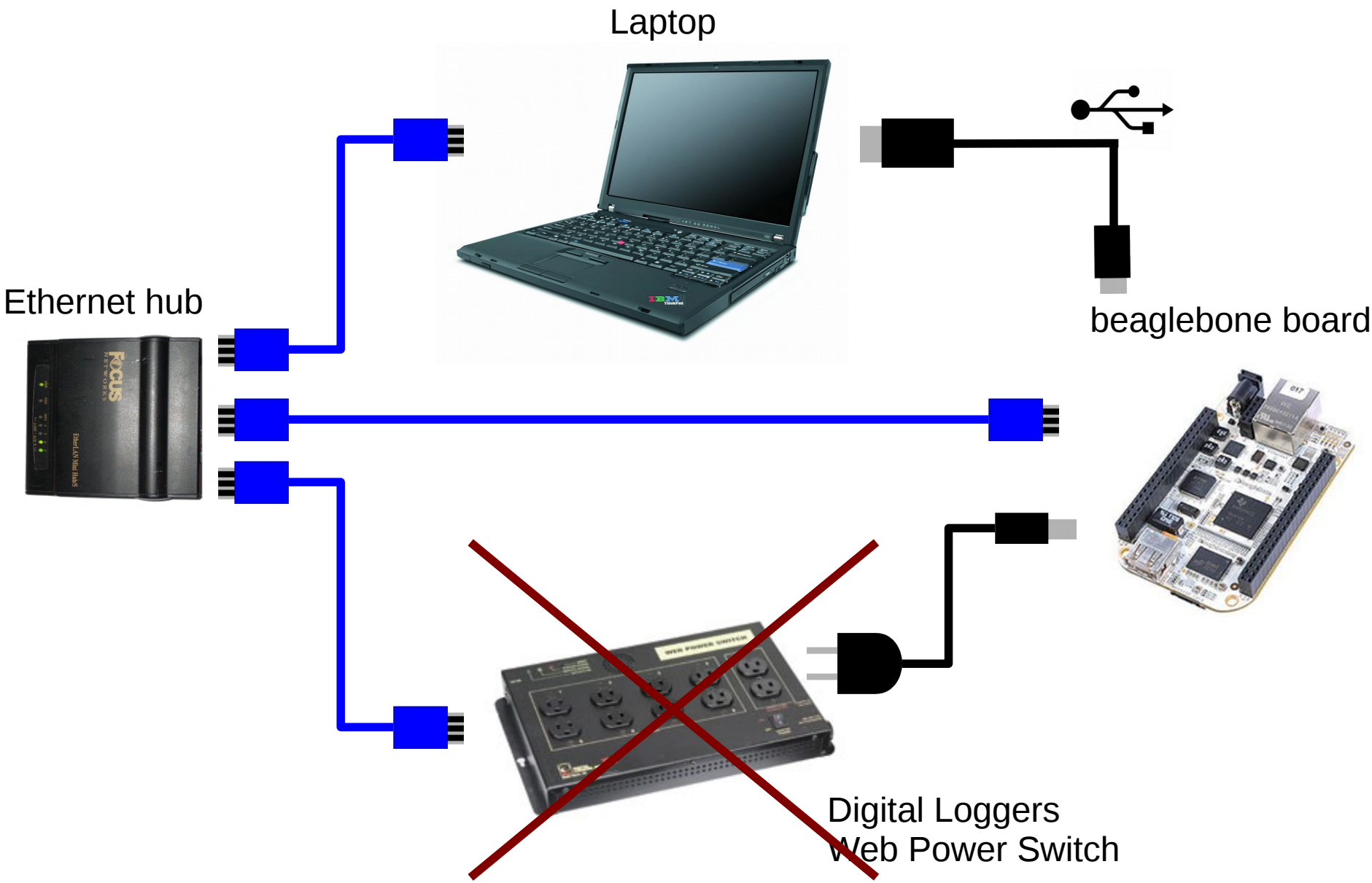
  “outlet?1”

wget --no-proxy -O /dev/null -q --auth-no-challenge 'http://admin:admin@power/outlet?1=**ON**'

My Setup

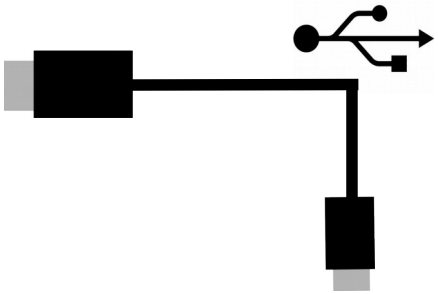

Laptop
Ethernet hub
beaglebone board
Digital Loggers
Web Power Switch

My Setup

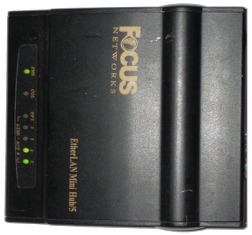

Laptop
Ethernet hub
beaglebone board
Digital Loggers
Web Power Switch

My Setup

Laptop



Ethernet hub



beaglebone board

My Thumb

My Setup (/etc/dhcpd/dhcpd.conf)

```
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.13.0 netmask 255.255.255.0 {
        range dynamic-bootp 192.168.13.100 192.168.13.190;
        option broadcast-address 192.168.13.255;
        next-server 192.168.13.1;
        option subnet-mask 255.255.255.0;
        filename "beagle-image";
}
```

My Setup (/etc/dhcpd/dhcpd.conf)

```
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.13.0 netmask 255.255.255.0 {
        range dynamic-bootp 192.168.13.100 192.168.13.190;
        option broadcast-address 192.168.13.255;
        next-server 192.168.13.1;
        option subnet-mask 255.255.255.0;
        filename "beagle-image";
}
```

My Setup

/etc/xinetd.d/tftp:

```
service tftp
{
    socket_type     = dgram
    protocol        = udp
    wait            = yes
    user            = root
    server          = /usr/sbin/in.tftpd
    server_args     = -s /var/lib/tftpboot
    disable         = no
    per_source      = 11
    cps             = 100 2
    flags           = IPv4
}
```

/etc/inetd.conf:

```
tftp  dgram  udp4   wait  nobody /usr/sbin/tcpd /usr/sbin/in.tftpd --tftpd-timeout 300 \
    --retry-timeout 5 --mcast-port 1758 --mcast-addr 239.239.239.0-255 --mcast-ttl 1 \
    --maxthread 100 --verbose=5 /srv/tftp
```

My Setup
(Problems with tftp?)

```
$ tftp localhost
tftp> get beagle-image
Error code 0: Permission denied
tftp>
```

Turn off selinux

```
# setenforce 0
```

My Setup (beaglebone: printenv)

```
baudrate=115200
board=am335x
bootcmd=bootp;run mmcargs;bootm;
bootcount=1
bootdelay=1
bootfile=zImage
bootm_size=0x10000000
console=ttyO0,115200n8
ethaddr=d4:94:a1:8b:ec:78
importbootenv=echo Importing environment from mmc ...; env import -t -r $loadaddr $filesize
kernel_addr_r=0x82000000
loadaddr=0x82000000
mmcargs=setenv bootargs console=${console} ${optargs} root=${mmcroot} rootfstype=${mmcrootfstype}
mmcdev=0
mmcroot=/dev/mmcblk0p2 ro
mmcrootfstype=ext4 rootwait
netargs=setenv bootargs console=${console} ${optargs} root=/dev/nfs nfsroot=${serverip}:${rootpath},${nfsopts} rw ip=dhcp
```

Reading Console

- ttywatch
  - /etc/ttywatch.conf

    --name USB0 --port /dev/ttyUSB0 --bps 115200 --ipport 3001

  - telnet localhost 3001
  - nc localhost 3001

Reading Console

- ttywatch
  - When beaglebone is power cycled
    - Resets USB0
    - breaks connection with ttywatch
- Direct read from serial

```
stty -F /dev/ttyUSB0 115200 parodd; cat /dev/ttyUSB0
```

Reading Console

- Can't just use “cat”
    - ktest.pl will also get confused on power reset.
- mkfifo beagle-cat
- Make a script “console” that does

```
while :; do
        stty -F /dev/ttyUSB0 115200 parodd 2>/dev/null &&
         cat /dev/ttyUSB0
done > beagle-cat
```

- ./console &
- CONSOLE = cat ${THIS_DIR}/beagle-cat

Start

- Run ktest.pl with no option, or minimum configs
  - Asks the minimum of questions
  - creates a file ktest.conf
  - defaults to randconfig build
    - may change in the future
- Update the config to suite your needs
  - use sample.conf
  - Several examples in tools/testing/ktest/examples

Options

- TEST_TYPE = <what to do>
  - build, install, boot or test?
- MACHINE = <name-of-board>
  - Unique identifier for board
  - Used for commands (like scp files to target)
- BUILD_DIR = <path>
  - directory of the source code (or git repo)
- OUTPUT_DIR = <path>
  - directory to build the code “make O=<path>”

Options

- BUILD_OPTIONS = <options>
  - Added to make of vmlinux
  - Add -j8 to speed up the build
  - Add targets when needed “bzImage” and “modules”
- POWER_CYCLE = <shell-command>
  - Used to power cycle board
    - for kernel crash
    - failed to “ssh user@${MACHINE} reboot”

Options

- CONSOLE = <shell-command>
  - Reads anything that produces stdout of the target's console
  - Must be continuous stream (not reset on reboot)
- SSH_USER = <user> (usually “root”)
  - Privileged user on target that can reboot and install kernel images
- BUILD_TARGET = <relative path to image>
  - Path relative to OUTPUT_DIR
  - arch/x86/boot/bzImage

Options

- TARGET_IMAGE = <path-to-boot-from>
  - /boot/vmlinux-test
- LOCAL_VERSION = <text>
  - localversion file
  - required to prevent you from killing the stable kernel

Options

- REBOOT_TYPE = grub (default)
  - '= script' lets you define how to reboot to kernel
- REBOOT_SCRIPT = <script>
  - script to use when REBOOT_TYPE = script
- GRUB_MENU = <menu title>
  - searches for this title in /boot/grub/menu.lst
    - REBOOT_TYPE = grub2 (is semi-supported)
      - I don't use it ;-)

Options

- REBOOT_TYPE = syslinux
    - syslinux – an alternative to grub (on x86)
- SYSLINUX_LABEL = <label>
    - searches the label to boot

Setup for Beaglebone

- TEST_TYPE = boot
- MACHINE = beagle (what you ssh to)
- BUILD_DIR = ${THIS_DIR}/linux.git
    - THIS_DIR is a special variable that is defined as the location you are running this
- OUTPUT_DIR = ${THIS_DIR}/beagle-build
- BUILD_OPTIONS = -j8 uImage
- POWER_CYCLE =

wget --no-proxy -O /dev/null -q --auth-no-challenge 'http://admin:admin@power/outlet?1=CCL'

Setup for Beaglebone

- TEST_TYPE = boot
- MACHINE = beagle (what you ssh to)
- BUILD_DIR = ${THIS_DIR}/linux.git
    - THIS_DIR is a special variable that is defined as the location you are running this
- OUTPUT_DIR = ${THIS_DIR}/beagle-build
- BUILD_OPTIONS = -j8 uImage
- POWER_CYCLE = echo use the thumb Luke; read a

Setup for Beaglebone

- CONSOLE = cat ${THIS_DIR}/beagle-cat
- SSH_USER = root (but we are not using it)
- BUILD_TARGET = arch/arm/boot/uImage
- TARGET_IMAGE =

  /srv/tftp/beagle-image
- LOCALVERSION = -test
- REBOOT_TYPE = script

Demo

Options

- LOG_FILE = <file>
    - writes all console output and commands run to a file

Extra Options

- LOG_FILE = ${OUTPUT_DIR}/beagle.log

Extra Options

- LOG_FILE = ${OUTPUT_DIR}/beagle.log

Demo

Options

- MAKE_CMD = <command> (default “make”)
  - Used to run all makes in ktest.pl
  - make ARCH=powerpc
- BUILD_TYPE = <type>
  - pass to make, like “randconfig”
    - BUILD_TYPE = randconfig
      - make randconfig
    - BUILD_TYPE = oldconfig
    - BUILD_TYPE = allnoconfig
  - useconfig:<path/to/config>
    - BUILD_TYPE = useconfig:${PWD}/myconfig

Extra Options

- MAKE_CMD =

  PATH=/usr/local/gcc-4.6.3-nolibc/arm-unknown-linux-gnueabi/bin:$PATH
  CROSS_COMPILE=arm-unknown-linux-gnueabi- make ARCH=arm

- BUILD_TYPE = multi_v7_defconfig
  - Opiton used to create config file
  - oldconfig
  - useconfig:<path-to-config>

DEMO

Config file

- Broken up into sections
  - DEFAULTS
    - All options here are used by all tests
    - Multiple sections are the same as a single section
      - except when a section is conditional
  - TEST_START
    - May override DEFAULTS options
    - Each section defines a single test
      - may have an iterator.
  - Options before first section header
    - defaults to DEFAULTS

Options and Variables

- OPTION = value
  - only one definition of an option is allowed in a section
  - used by ktest.pl as commands (persistent)
  - when defined in TEST_START, only for that test
- VARIABLE := value
  - can be overridden throughout the file
  - Used only for reading config file
  - not used by ktest.pl
  - defined in tests are still available in DEFAULTS

Options and Variables

- Defined with '=' or ':=' for option or variable respectively
- both can be used with ${VAR}
    - MACHINE = mybox
    - SSH := ssh root@${MACHINE}
    - TEST = ${SSH} /work/test

SKIP

- Sections marked with SKIP are ignored
    - DEFAULTS SKIP
    - TEST_START SKIP
- It is treated like the section has been commented out
- Even variables within a skipped section is not processed (they too are ignored).

ITERATE

- Run the same test over and over
  - TEST_START ITERATE 10
    - just like cut and pasting the TEST_START section 10 times in a row
- TEST_START ITERATE 10 SKIP
  - Just like normal sections, will be skipped and ignored

OVERRIDE

- Allows a section to set options that have been previously set
    - Only works with DEFAULTS section
    - DEFAULTS OVERRIDE
- Rule still applies
    - option may only be defined once within the section
- Overrides options from previous sections
    - later sections can not duplicate options

Check on Demo

Before and after builds

- PRE_BUILD = <shell script>
  - executed before running a build
- POST_BUILD = <shell script>
  - executed right after running a build
- PRE_BUILD_DIE = 1
- POST_BUILD_DIE = 1
  - set to kill the test if the PRE_BUILD or POST_BUILD fail (exit non zero)

Beaglebone

- PRE_BUILD = ${MAKE_CMD} O=${OUTPUT_DIR} oldnoconfig dtbs
  - Create the dtbs files
- POST_BUILD =

cat ${OUTPUT_BOOT}/zImage ${OUTPUT_BOOT}/dts/am335x-bone.dtb > ${OUTPUT_BOOT}/zImage.beagle; mkimage -A arm -O linux -C none -T kernel -a ${LOADADDR} -e ${LOADADDR} -d ${OUTPUT_BOOT}/zImage.beagle ${OUTPUT_BOOT}/uImage

  - Need to create the uImage with the proper device tree

Beaglebone

- Install mkimage
    - yum install uboot-tools
    - apt-get u-boot-tools
- LOADADDR := 0x80008000
- OUTPUT_BOOT := ${OUTPUT_DIR}/arch/arm/boot
    - use of variables

Beaglebone

- BUILD_NOCLEAN = 1
    - Does not perform a “make mrproper”
- CLEAR_LOG = 1
    - “= 0” appends to LOG_FILE (default)
    - “= 1” truncates file (open with “O_TRUNC”)
    - DEFAULTS option (ignored in TEST_START)

Demo

Beaglebone

- SCP_TO_TARGET =

  scp $SRC_FILE $SSH_USER@$MACHINE:$DST_FILE

  - Used to copy files from host to target

Beaglebone

- SCP_TO_TARGET =

  scp $SRC_FILE $SSH_USER@$MACHINE:$DST_FILE

  - Used to copy files from host to target
- SCP_TO_TARGET = echo “don't do scp”

Demo

Beaglebone

- ktest.pl will try to install modules if
  - CONFIG_MODULES=y
  - Requires ssh access to target
- No ssh access
- No modules needed

Options

- MIN_CONFIG = <file>
  - Best if it is the minimum config to build kernel
- ADD_CONFIG = <file1> <file2> <file3>
  - Add configs to MIN_CONFIG
    - MIN_CONFIG takes precedence
- Both set and unset configs take affect
  - common mistake is to keep the
    - # CONFIG_FOO_BAR is not set
  - grep '^CONFIG' .config > min_config

Beaglebone

- ADD_CONFIG = ${THIS_DIR}/addconfig
  - # CONFIG_MODULES is not set

Beaglebone

- ADD_CONFIG = ${THIS_DIR}/addconfig
    - # CONFIG_MODULES is not set
- Build modules?
    - the real fix!

Beaglebone

- ADD_CONFIG = ${THIS_DIR}/addconfig
  - # CONFIG_MODULES is not set
- Build modules?
  - the real fix!
  - BUILD_OPTIONS = -j8 uImage modules
  - Need to replace TARGET_IMAGE, as it still requires SCP_TO_TARGET to work

Beaglebone

DEMO

IF

- Sections may be conditionally skipped
  - TEST_START IF ${VAR}
    - will only run if VAR is defined and is non zero
- May also handle compares
  - TEST_START IF ${TEST_CNT} > 10
- Complex compares
  - TEST_START IF ${DEFAULTS} || (${TEST_RUN} == ARM)
  - (Note: does not handle line breaks)

IF

- DEFINED
  - Test if a variable or option is defined
    - DEFAULTS IF DEFINED REBOOT
- NOT DEFINED
  - test if a variable is not defined
    - DEFAULTS IF NOT DEFINED BITS
      - BITS := 64
    - TEST = ./hackbench_${BITS} 10

ELSE (IF)

- Followed by a section that has an IF
  - DEFAULTS IF ${ARCH} == x86_64
    - BITS := 64
  - DELAULTS ELSE
    - BITS := 32
- May be followed by IF to do selections
  - DEFAULTS IF ${TEST} == build
  - DEFAULTS ELSE IF ${TEST} == boot
  - DEFAULTS ELSE

INCLUDE

- INCLUDE <file>
  - can be full path
  - searches config file directory
  - searches local director
- Only allowed in DEFAULTS section
- may define TEST_START
- DEFAULTS defined before are seen
- DEFAULTS defined in included files are defined in parent file (like CPP)

mxtest.conf

```
MACHINE = mxtest
BOX := mxtest

CONSOLE = nc -d fedora 3001

# TESTS = patchcheck, randconfig, boot, test, config-bisect, biscet
TEST := patchcheck

# Run allno, ftrace, noftrace, notrace, allmod and allyes
CONFIG_TESTS := 1

CONFIG_ALLYES := 0
CONFIG_ALLYES_TEST_TYPE := build

# REBOOT = none, fail, empty
#REBOOT := fail

MACHINE := mxtest

GCC_VERSION := 4.6.0

BITS:= 64

INCLUDE include/defaults.conf
INCLUDE include/patchcheck.conf
INCLUDE include/tests.conf
INCLUDE include/bisect.conf
INCLUDE include/config-bisect.conf
INCLUDE include/minconfig.conf
INCLUDE include/config-tests.conf

DEFAULTS OVERRIDE
POST_INSTALL =
OUTPUT_DIR = ${THIS_DIR}/nobackup/${MACHINE}
```

defaults.conf

```
DEFAULTS IF NOT DEFINED BITS
BITS := 64

DEFAULTS

SSH := ssh ${SSH_USER}@${MACHINE}
THIS_DIR := /home/rostedt/work/git
CONFIG_DIR := ${THIS_DIR}/configs/${MACHINE}

REBOOT_SUCCESS_LINE = login:

BUILD_DIR = ${THIS_DIR}/linux-${BOOT_TYPE}.git
OUTPUT_DIR = ${THIS_DIR}/nobackup/${MACHINE}/${BOOT_TYPE}

DEFAULTS
REBOOT_ON_SUCCESS = 0
REBOOT_ON_ERROR = 1
POWEROFF_ON_ERROR = 0
POWEROFF_ON_SUCCESS = 0

DEFAULTS
SSH_USER = root
POWER_OFF = ${THIS_DIR}/${MACHINE}-poweroff
POWER_CYCLE = ${THIS_DIR}/${MACHINE}-cycle
BUILD_TARGET = arch/x86/boot/bzImage
CLEAR_LOG = 1
LOCALVERSION = -test
MAKE_CMD = GCC_VERSION=${GCC_VERSION} distmake-${BITS}
BUILD_OPTIONS = -j40
LOG_FILE = ${THIS_DIR}/nobackup/${MACHINE}/${MACHINE}.log
MIN_CONFIG = ${CONFIG_DIR}/config-min
TMP_DIR = /tmp/ktest/${MACHINE}

GRUB_MENU = ${GRUBNAME} Kernel
TARGET_IMAGE = /boot/vmlinuz-test${EXT}
POST_INSTALL = ${SSH} /sbin/dracut -f /boot/initramfs-test${EXT}.img $KERNEL_VERSION

STORE_FAILURES = ${THIS_DIR}/failures/${MACHINE}
```

TEST_START

- build
  - just builds the kernel
- install
  - build and installs the kernel
- boot
  - builds, installs and boots the kernel
- test
  - builds, boots and runs a command
  - TEST = <command>
    - runs from host but may use 'ssh' to target

tests.conf

```
TEST_START IF ${TEST} == boot
TEST_TYPE = boot
BUILD_TYPE = oldconfig
BUILD_NOCLEAN = 1

TEST_START ITERATE 10 IF ${TEST} == randconfig
MIN_CONFIG = ${CONFIG_DIR}/config-net
TEST_TYPE = test
BUILD_TYPE = randconfig
TEST = ${SSH} /work/c/hackbench_${BITS} 50

TEST_START ITERATE 10 IF ${TEST} == randconfig && ${MULTI}
TEST_TYPE = boot
BUILD_TYPE = randconfig
MIN_CONFIG = ${CONFIG_DIR}/config-min
MAKE_CMD = make

TEST_START IF ${TEST} == test
TEST_TYPE = test
#BUILD_TYPE = oldconfig
#BUILD_TYPE = useconfig:${CONFIG_DIR}/config-net
BUILD_TYPE = useconfig:${CONFIG_DIR}/config-bisect
#BUILD_TYPE = nobuild
TEST = ${SSH} /work/bin/test-mod-event
BUILD_NOCLEAN = 1
```

TEST_START

- patchcheck
  - Requires BUILD_DIR be a git repo
  - PATCHCHECK_TYPE = <type>
    - build, boot or test
  - PATCHCHECK_START = <commit>
    - git commit to start testing (SHA1, tag, etc)
  - PATCHCHECK_STOP = <commit>
    - git commit to stop (SHA1, HEAD)

TEST_START

- make_warnings_file
  - Creates a file listing warnings from a build
  - Can be used by other tests to check for new warnings
  - In other tests, the build will fail if a new warning is detected
  - WARNINGS_FILE
    - Can be full path
    - Defaults to being in ${OUTPUT_DIR}

patchcheck.conf

```
DO_WARNINGS := 1
WARNINGS_FILE_NAME := warnings_file_3.17
#FIX_PATCH := fix-3.17-rc5.patch

PATCH_START := HEAD~1
PATCH_END := HEAD
PATCH_CHECKOUT := trace/trace/ftrace/urgent
PATCH_CONFIG = ${CONFIG_DIR}/config-ftrace-patchcheck
PATCH_TEST := ${SSH} "cd /work/bin && ./trace-cmd-filter-stress && ./ftrace-test-
stress /work/c/hackbench_${BITS} 50"

TEST_START IF ${TEST} == patchcheck && ${DO_PATCH_WARNINGS} == 1
TEST_NAME = patchcheck make warnings
TEST_TYPE = make_warnings_file
WARNINGS_FILE = ${WARNINGS_FILE}
BUILD_TYPE = useconfig:${PATCH_CONFIG}
CHECKOUT = ${PATCH_START}~1
BUILD_NOCLEAN = 0

TEST_START IF ${TEST} == patchcheck
TEST_NAME = patchcheck main
TEST_TYPE = patchcheck
MIN_CONFIG = ${PATCH_CONFIG}
TEST = ${PATCH_TEST}
BUILD_NOCLEAN = 1
PATCHCHECK_TYPE = test
PATCHCHECK_START = ${PATCH_START}
PATCHCHECK_END = ${PATCH_END}
CHECKOUT = ${PATCH_CHECKOUT}
WARNINGS_FILE = ${WARNINGS_FILE}
```

Test Name

- TEST_NAME = any name you want
- Printed with the result (PASSED or FAILED)
- If you see a failure, you know which test failed

Test Name

- Example from tools/testing/ktest/examples/crosstests.conf
- TEST_NAME = ${ARCH} ${CROSS}

```
**********************************************
**********************************************
KTEST RESULT: TEST 2 (arm arm-unknown-linux-gnueabi) SUCCESS!!!!   **
**********************************************
**********************************************
```

TEST_START

- bisect
  - Requires BUILD_DIR to be a git repo
  - performs a git bisect
  - BISECT_TYPE (build, boot or test)
  - BISECT_GOOD = <commit>
    - git commit that is marked good
      - (git bisect good <commit>)
  - BISECT_BAD = <commit>
    - git commit that is marked bad
      - (git bisect bad <commit>)

TEST_START

- bisect
  - BISECT_REVERSE = 1
    - good is bad, bad is good
  - BISECT_MANUAL = 1
    - asks you between tests if bisect was good
  - BISECT_CHECK = 1 (good/bad)
    - tests good and bad before doing bisect
  - BISECT_FILES = <file1> <file2> <dir1>
    - Only bisect based on these files or directories
    - runs 'git bisect start -- <file1> <file2> <dir1>'

TEST_START

- bisect
  - BISECT_SKIP = 0
    - fail on failed bisect instead of running
      - git bisect skip
  - BISECT_REPLY = <file>
    - failed bisect, run git bisect log > file
  - BISECT_START = <commit>
    - checks out commit after bisect start and stop
    - runs after BISECT_REPLY if it is defined
  - MIN_CONFIG = <config>
    - future will allow BUILD_TYPE

TEST_START

- bisect
  - BISECT_TRIES = 5
    - Will run each iteration this many times before deciding it’s good
    - Only one failure is needed
  - BISECT_RET_GOOD = 1
  - BISECT_RET_BAD = 0
    - For when tests return something other than zero for good, and non zero for bad
  - BISECT_RET_ABORT
    - If test finds something else wrong

TEST_START

- bisect
  - BISECT_RET_SKIP = 2
    - If the test thinks a git bisect skip should happen, it returns this value
  - BISECT_RET_DEFAULT = good
    - If the test returns something other than defined, what to do?
    - values are:
      - good
      - bad
      - skip
      - abort

bisect.conf

```
TEST_START IF ${TEST} == bisect
TEST_TYPE = bisect
BISECT_TYPE = boot
MIN_CONFIG = ${CONFIG_DIR}/config-ftrace-patchcheck
BISECT_GOOD = v2.6.39
BISECT_BAD = HEAD
CHECKOUT = origin/master
TEST = ssh ${USER}@${MACHINE} /work/bin/test-writeback-sync
#BISECT_REPLAY = /tmp/replay1
```

Check on Demo

Reboot failed?

- Problems with systemd
    - reboot no longer gives a nice exit status!
- Make custom reboot
    - beagle-reboot

```
#!/bin/bash
ssh $1 /sbin/reboot&
sleep 5
kill %1
exit 0
```

REBOOT = ${THIS_DIR}/beagle-reboot ${SSH_USER}@ {$MACHINE}

Options

- POST_INSTALL = <what to do after install>
  - optional
  - `ssh user@target /sbin/dracut -f /boot/initramfs-test.img $KERNEL_VERSION`
  - $KERNEL_VERSION is not a normal variable
    - does not have { }
    - it is replaced by the kernel version found by ktest.pl

Options

- SWITCH_TO_TEST = <shell-command>
    - Run before rebooting to test kernel
- SWITCH_TO_GOOD = <shell-command>
    - Run before rebooting to default kernel

Beaglebone

- TFTPBOOT := /srv/tftp
- TFTPDEF :=${TFTPBOOT}/beagle-default
- TFTPTEST := ${OUTPUT_DIR}/${BUILD_TARGET}
- SWITCH_TO_TEST = cp ${TFTPTEST} ${TARGET_IMAGE}
- SWITCH_TO_GOOD = cp ${TFTPDEF} ${TARGET_IMAGE}
- If you use modules, change TARGET_IMAGE to an option of your choice, and have TARGET_IMAGE be something that can be copied to the beaglebone (although not used)
    - Will add a new option to fix this in the future

Demo

Options

- SUCCESS_LINE = <text-denoting-success>
  - default “login:”
  - Can change to “root@linaro:~#”
- REBOOT_SUCCESS_LINE = <text>
  - Quick way to detect successful good reboot

Options

- POWEROFF_ON_SUCCESS = 1
- REBOOT_ON_SUCCESS = 1
    - ignored if POWER_OFF_ON_SUCCESS is set
- POWEROFF_ON_ERROR= 1
- REBOOT_ON_ERROR = 1
    - ignored if POWEROFF_ON_ERROR is set
- POWERCYCLE_AFTER_REBOOT = <secs>
    - nice when reboot doesn't finish the reboot
- POWEROFF_AFTER_HALT = <secs>

Options

- DIE_ON_FAILURE = 0 (default 1)
  - When set to zero, a failed test will not stop ktest

```
*******************************************
*******************************************
KTEST RESULT: TEST 1 SUCCESS!!!!        **
*******************************************
*******************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
KTEST RESULT: TEST 2 Failed: failed - got a bug report
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Options

- STORE_FAILURES = <dir>
  - Used when DIE_ON_FAILURE = 0
  - Creates directory within this directory
    - MACHINE-TEST-fail-TIMESTAMP
    - mitest-boot-randconfig-fail-20110008154933
  - Saves dmesg, config, build log and test log

TEST_START

- config_bisect
  - Find why one config works and another does not
  - CONFIG_BISECT_TYPE (build, boot, test)
  - CONFIG_BISECT_GOOD = <file> (optional)
    - start config
    - default is to use MIN_CONFIG
    - The current good is saved in the OUTPUT_DIR as “config_good”
  - CONFIG_BISECT = <file>
    - the bad config

config_bisect

- How it works?
  - ignore configs defined in good config
  - try first half
    - test if it changed config
    - test other half
      - only one config needs to be set to continue
  - test passes
    - Have a new “good” config
  - test fails
    - have new “bad” config
      - wash, rinse, repeat

config-bisect.conf

```
TEST_START IF ${TEST} == config-bisect
TEST_TYPE = config_bisect
CONFIG_BISECT_TYPE = boot
#CONFIG_BISECT = ${THIS_DIR}/nobackup/failures/mxtest-boot-randconfig-fail-20110502120128/config
CONFIG_BISECT = ${THIS_DIR}/config-bad
#CHECKOUT = origin/master
#CONFIG_BISECT_GOOD = ${THIS_DIR}/config-good
```

TEST_START

- make_min_config
  - OUTPUT_MIN_CONFIG = <file>
    - The new min config
  - START_MIN_CONFIG = <file> (optional)
    - default uses MIN_CONFIG
  - IGNORE_CONFIG = <file> (optional)
    - Persistent (wont clear it in multiple runs)
    - Only configs that ktest.pl found succeeds to boot
    - Does not add allnoconfig configs
    - Does not add previous MIN_CONFIG configs

make_min_config

- How it works?
  - Read Kconfigs to find depends and selects
  - Pick the config which has the most depending on it
  - Disable that config (make sure new config changes)
  - Fails – enable it and all that depend on it
    - Update OUTPUT_MIN_CONFIG
  - Passes – Keep it permanently disabled
    - Add to IGNORE_CONFIG

cross compiling

- Get binary cross compilers from kernel.org
  - http://www.kernel.org/pub/tools/crosstool/files/bin/x86_64/
- All developers should run cross compilers for all the archs their code affects (even drivers)
- tools/testing/ktest/examples/crosstests.conf

crosstests.conf

```
THIS_DIR := /work/autotest
ARCH_DIR := ${THIS_DIR}/nobackup/linux-test.git/arch

BUILD_DIR = ${THIS_DIR}/nobackup/cross-linux.git

DO_FAILED := 0
DO_DEFAULT := 1
#RUN := m32r

GCC_VER = 4.5.2
MAKE_CMD = PATH=/usr/local/gcc-${GCC_VER}-nolibc/${CROSS}/bin:$PATH CROSS_COMPILE=${CROSS}- make ARCH=$
{ARCH}
TEST_TYPE = build

BUILD_TYPE = defconfig

TEST_NAME = ${ARCH} ${CROSS}

# alpha
TEST_START IF ${RUN} == alpha || ${DO_DEFAULT}
#MIN_CONFIG = ${ARCH_DIR}/alpha/defconfig
CROSS = alpha-linux
ARCH = alpha

# arm
TEST_START IF ${RUN} == arm || ${DO_DEFAULT}
#MIN_CONFIG = ${ARCH_DIR}/arm/configs/cm_x300_defconfig
CROSS = arm-unknown-linux-gnueabi
ARCH = arm

# black fin
TEST_START IF ${RUN} == bfin || ${DO_DEFAULT}
#MIN_CONFIG = ${ARCH_DIR}/blackfin/configs/BF561-EZKIT-SMP_defconfig
CROSS = bfin-uclinux
ARCH = blackfin
BUILD_OPTIONS = -j8 vmlinux
```

crosstests.conf

```
# cris - FAILS?
TEST_START IF ${RUN} == cris || ${RUN} == cris64 || ${DO_FAILED}
#MIN_CONFIG = ${ARCH_DIR}/cris/configs/etraxfs_defconfig
CROSS = cris-linux
ARCH = cris

# cris32 - not right arch?
TEST_START IF ${RUN} == cris || ${RUN} == cris32 || ${DO_FAILED}
#MIN_CONFIG = ${ARCH_DIR}/cris/configs/etrax-100lx_v2_defconfig
CROSS = crisv32-linux
ARCH = cris

# ia64
TEST_START IF ${RUN} == ia64 || ${DO_DEFAULT}
#MIN_CONFIG = ${ARCH_DIR}/ia64/configs/generic_defconfig
CROSS = ia64-linux
ARCH = ia64

# frv
TEST_START IF ${RUN} == frv || ${DO_FAILED}
CROSS = frv-linux
ARCH = frv
GCC_VER = 4.5.1

# h8300 - failed make defconfig??
TEST_START IF ${RUN} == h8300 || 0
CROSS = h8300-elf
ARCH = h8300
GCC_VER = 4.5.1

# m68k fails with error?
TEST_START IF ${RUN} == m68k || ${DO_DEFAULT}
#MIN_CONFIG = ${ARCH_DIR}/m68k/configs/multi_defconfig
CROSS = m68k-linux
ARCH = m68k
```

crosstests.conf

```
[ ... ]

TEST_START IF ${RUN} == x86 || ${RUN} == i386 || ${DO_DEFAULT}
MAKE_CMD = distmake-32
ARCH = i386
CROSS =

TEST_START IF ${RUN} == x86 || ${RUN} == x86_64 || ${DO_DEFAULT}
MAKE_CMD = distmake-64
ARCH = x86_64
CROSS =


DEFAULTS
MACHINE = crosstest
SSH_USER = root
OUTPUT_DIR = ${THIS_DIR}/nobackup/cross-compile
BUILD_TARGET = cross
TARGET_IMAGE = image
POWER_CYCLE = cycle
CONSOLE = console
LOCALVERSION = version
GRUB_MENU = grub
LOG_FILE = ${THIS_DIR}/nobackup/cross-compile/cross.log
BUILD_OPTIONS = -j8

REBOOT_ON_ERROR = 0
POWEROFF_ON_ERROR = 0
POWEROFF_ON_SUCCESS = 0
REBOOT_ON_SUCCESS = 0
DIE_ON_FAILURE = 0
STORE_FAILURES = ${THIS_DIR}/nobackup/failures/cross


CLEAR_LOG = 1
```

Miscellaneous

- bisect and config bisect can restart without user manually saving it
    - Although I may have broken this for conig_bisect
- If all tests is just build, do not require options for boot and test

TODO

- Add output results to all tests
- Fix bisects to use BUILD_TYPE
- Add option to change SIGINT to CONSOLE
    - (done in v3.14!)
- Change config-bisect to diff any two configs
    - (done in v3.17!)
    - Old way required good config to be a subset of bad config