5.9.6. Environment Variables Commands

5.9.6.1. printenv- print environment variables

```
=> help printenv

printenv - print environment variables


Usage:

printenv

    - print values of all environment variables

printenv name ...

    - print value of environment variable 'name'

=>
```

The `printenv` command prints one, several or all variables of the U-Boot environment. When arguments are given, these are interpreted as the names of environment variables which will be printed with their values:

```
=> printenv ipaddr hostname netmask

ipaddr=192.168.100.6

hostname=canyonlands

netmask=255.255.0.0

=>
```

Without arguments, `printenv` prints all a list with all variables in the environment and their values, plus some statistics about the current usage and the total size of the memory available for the environment.

```
=> printenv

bootdelay=5

baudrate=115200

loads_echo=

preboot=echo;echo Type "run flash_nfs" to mount root filesystem over NFS;echo

netdev=eth0

nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath}

ramargs=setenv bootargs root=/dev/ram rw

addip=setenv                               bootargs                                 ${bootargs}
ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:${netdev}:off panic=1

addtty=setenv bootargs ${bootargs} console=ttyS0,${baudrate}

addmisc=setenv bootargs ${bootargs}
```

```
initrd_high=30000000

kernel_addr_r=400000

ramdisk_addr_r=C00000

hostname=canyonlands

ramdisk_file=canyonlands/uRamdisk

rootpath=/opt/eldk/ppc_4xxFP

flash_self=run ramargs addip addtty addmisc;bootm ${kernel_addr} ${ramdisk_addr} ${fdt_addr}

flash_nfs=run nfsargs addip addtty addmisc;bootm ${kernel_addr} - ${fdt_addr}

net_nfs=tftp ${kernel_addr_r} ${bootfile}; tftp ${fdt_addr_r} ${fdt_file}; run nfsargs addip addtty addmisc;bootm ${kernel_addr_r} - ${fdt_addr_r}

net_self_load=tftp ${kernel_addr_r} ${bootfile};tftp ${fdt_addr_r} ${fdt_file};tftp ${ramdisk_addr_r} ${ramdisk_file};

net_self=run net_self_load;run ramargs addip addtty addmisc;bootm ${kernel_addr_r} ${ramdisk_addr_r} ${fdt_addr_r}

fdt_file=canyonlands/canyonlands.dtb

update=protect off 0xFFFA0000 FFFFFFFF;era 0xFFFA0000 FFFFFFFF;cp.b ${fileaddr} 0xFFFA0000 ${filesize};setenv filesize;saveenv

upd=run load update

nload=tftp 200000 canyonlands/u-boot-nand.bin

nupdate=nand erase 0 100000;nand write 200000 0 100000;setenv filesize;saveenv

nupd=run nload nupdate

pciconfighost=1

pcie_mode=RP:RP

ethaddr=00:10:ec:01:08:84

eth1addr=00:10:ec:81:08:84

hostname=canyonlands

sr=tftp 200000 canyonlands/u-boot.bin-sr;protect off 0xFFFA0000 FFFFFFFF;era 0xFFFA0000 FFFFFFFF;cp.b ${fileaddr} 0xFFFA0000 ${filesize};setenv filesize;saveenv

srlinux=setenv bootfile canyonlands/uImage-sr;setenv fdt_file canyonlands/canyonlands.dtb-sr;run net_nfs

bootcmd=run srlinux

fdtaddr=800000

uboot_file=canyonlands/u-boot.bin-duts

load=tftp 200000 ${u-boot}

dzu_net_nfs=setenv bootfile dzu/canyonlands/uImage;setenv fdt_file dzu/canyonlands/canyonlands.dtb;run net_nfs
```

```
bootargs=root=/dev/ram                                                            rw
ip=192.168.100.6:192.168.1.1:192.168.1.254:255.255.0.0:canyonlands:eth0:off       panic=1
console=ttyS0,115200

ethact=ppc_4xx_eth0

bootfile=/tftpboot/duts/canyonlands/uImage

bar=This is a new example.

cons_opts=console=tty0 console=ttyS0,${baudrate}

test=echo This is a test;printenv ipaddr;echo Done.

test2=echo This is another Test;printenv hostname;echo Done.

kernel_addr=0xFC000000

ramdisk_addr=0xFC200000

fdt_addr=0xFC1E0000

fdt_addr_r=0x00b00000

u-boot=/tftpboot/duts/canyonlands/u-boot.bin

fileaddr=200000

gatewayip=192.168.1.254

netmask=255.255.0.0

ipaddr=192.168.100.6

serverip=192.168.1.1

stdin=serial

stdout=serial

stderr=serial

ver=U-Boot 2009.11.1 (Feb 05 2010 - 08:57:12)


Environment size: 2780/16379 bytes

=>
```

5.9.6.2. saveenv - save environment variables to persistent storage

```
=> help saveenv

saveenv - save environment variables to persistent storage


Usage:

saveenv

=>
```

All changes you make to the U-Boot environment are made in RAM only. They are lost as soon as you reboot the system. If you want to make your changes permanent you have to use the `saveenv` command to write a copy of the environment settings to persistent storage, from where they are automatically loaded during startup:

```
=> saveenv

Saving Environment to Flash...

Un-Protected 1 sectors

Un-Protected 1 sectors

Erasing Flash...

. done

Erased 1 sectors

Writing to Flash... done

Protected 1 sectors

Protected 1 sectors

=>
```

5.9.6.3. setenv - set environment variables

```
=> help setenv

setenv - set environment variables



Usage:

setenv name value ...

    - set environment variable 'name' to 'value ...'

setenv name

    - delete environment variable 'name'

=>
```

To modify the U-Boot environment you have to use the `setenv` command. When called with exactly one argument, it will delete any variable of that name from U-Boot's environment, if such a variable exists. Any storage occupied for such a variable will be automatically reclaimed:

```
=> setenv foo This is an example value.

=> printenv foo

foo=This is an example value.

=> setenv foo

=> printenv foo

## Error: "foo" not defined

=>
```

When called with more arguments, the first one will again be the name of the variable, and all following arguments will (concatenated by single space characters) form the value that gets stored for this variable. New variables will be automatically created, existing ones overwritten.

```
=> printenv bar

## Error: "bar" not defined

=> setenv bar This is a new example.

=> printenv bar

bar=This is a new example.

=>
```

Remember standard shell quoting rules when the value of a variable shall contain characters that have a special meaning to the command line parser (like the `$`character that is used for variable substitution or the semicolon which separates commands). Use the backslash (`\`) character to escape such special characters, or enclose the whole phrase in apstrophes (`'`). Use "`${name}`" for variable expansion (see 14.2.17. How the Command Line Parsing Works for details).

```
=> setenv cons_opts 'console=tty0 console=ttyS0,${baudrate}'

=> printenv cons_opts

cons_opts=console=tty0 console=ttyS0,${baudrate}

=>
```

There is no restriction on the characters that can be used in a variable name except the restrictions imposed by the command line parser (like using backslash for quoting, space and tab characters to separate arguments, or semicolon and newline to separate commands). Even strange input like "=-/|()+=" is a perfectly legal variable name in U-Boot.

A common mistake is to write

```
setenv name=value
```

instead of

```
setenv name value
```

There will be no error message, which lets you believe everything went OK, but it didn't: instead of setting the variable *name* to the value *value* you tried to delete a variable with the name *name=value* - this is probably not what you intended! Always remember that name and value have to be separated by space and/or tab characters!

5.9.6.4. run - run commands in an environment variable

```
=> help run

run - run commands in an environment variable

```

```
Usage:

run var [...]

    - run the commands in the environment variable(s) 'var'

=>
```

You can use U-Boot environment variables to store commands and even sequences of commands. To execute such a command, you use the `run` command:

```
=> setenv test echo This is a test\;printenv ipaddr\;echo Done.

=> printenv test

test=echo This is a test;printenv ipaddr;echo Done.

=> run test

This is a test

ipaddr=192.168.100.6

Done.

=>
```

You can call `run` with several variables as arguments, in which case these commands will be executed in sequence:

```
=> setenv test2 echo This is another Test\;printenv hostname\;echo Done.

=> printenv test test2

test=echo This is a test;printenv ipaddr;echo Done.

test2=echo This is another Test;printenv hostname;echo Done.

=> run test test2

This is a test

ipaddr=192.168.100.6

Done.

This is another Test

hostname=canyonlands

Done.

=>
```

If a U-Boot variable contains several commands (separated by semicolon), and one of these commands fails when you "run" this variable, the remaining commands*will be executed anyway*.

If you execute several variables with one call to `run`, any failing command will cause "run" to terminate, i. e. the remaining variables are *not* executed.

5.9.6.5. bootd - boot default, i.e., run 'bootcmd'

```
=> help boot

boot - No help available.



=>
```

The `bootd` (short: `boot`) executes the default boot command, i. e. what happens when you don't interrupt the initial countdown. This is a synonym for the `run bootcmd`command.