

NAME

dracut.cmdline – dracut kernel command line options

DESCRIPTION

The root device used by the kernel is specified in the boot configuration file on the kernel command line, as always.

The traditional `root=/dev/sda1` style device specification is allowed, but not encouraged. The root device should better be identified by LABEL or UUID. If a label is used, as in `root=LABEL=<label_of_root>` the initramfs will search all available devices for a filesystem with the appropriate label, and mount that device as the root filesystem. `root=UUID=<uuidnumber>` will mount the partition with that UUID as the root filesystem.

In the following all kernel command line parameters, which are processed by dracut, are described.

"rd.*" parameters mentioned without "=" are boolean parameters. They can be turned on/off by setting them to {0|1}. If the assignment with "=" is missing "=1" is implied. For example `rd.info` can be turned off with `rd.info=0` or turned on with `rd.info=1` or `rd.info`. The last value in the kernel command line is the value, which is honored.

Standard

init=<path to real init>

specify the path to the init program to be started after the initramfs has finished

root=<path to blockdevice>

specify the block device to use as the root filesystem.

Example

```
root=/dev/sda1
root=/dev/disk/by-path/pci-0000:00:1f.1-scsi-0:0:1:0-part1
root=/dev/disk/by-label/Root
root=LABEL=Root
root=/dev/disk/by-uuid/3f5ad593-4546-4a94-a374-bcfb68aa11f7
root=UUID=3f5ad593-4546-4a94-a374-bcfb68aa11f7
root=PARTUUID=3f5ad593-4546-4a94-a374-bcfb68aa11f7
```

rootfstype=<filesystem type>

"auto" if not specified.

Example

```
rootfstype=ext4
```

rootflags=<mount options>

specify additional mount options for the root filesystem. If not set, `/etc/fstab` of the real root will be parsed for special mount options and mounted accordingly.

ro

force mounting / and `/usr` (if it is a separate device) read-only. If none of `ro` and `rw` is present, both are mounted according to `/etc/fstab`.

rw

force mounting / and `/usr` (if it is a separate device) read-write. See also `ro` option.

rootfallback=<path to blockdevice>

specify the block device to use as the root filesystem, if the normal root cannot be found. This can only be a simple block device with a simple file system, for which the filesystem driver is either compiled in, or added manually to the initramfs. This parameter can be specified multiple times.

rd.auto rd.auto=1

enable autoassembly of special devices like cryptoLUKS, dmraid, mdraid or lvm. Default is off as of dracut version >= 024.

rd.hostonly=0

removes all compiled in configuration of the host system the initramfs image was built on. This helps booting, if any disk layout changed, especially in combination with rd.auto or other parameters specifying the layout.

rd.cmdline=ask

prompts the user for additional kernel command line parameters

rd.fstab=0

do not honor special mount options for the root filesystem found in */etc/fstab* of the real root.

resume=<path to resume partition>

resume from a swap partition

Example

```
resume=/dev/disk/by-path/pci-0000:00:1f.1-scsi-0:0:1:0-part1
resume=/dev/disk/by-uuid/3f5ad593-4546-4a94-a374-bcfb68aa11f7
resume=UUID=3f5ad593-4546-4a94-a374-bcfb68aa11f7
```

rd.skipfsck

skip fsck for rootfs and */usr*. If you're mounting */usr* read-only and the init system performs fsck before remount, you might want to use this option to avoid duplication.

iso-scan/filename

Mount all mountable devices and search for ISO pointed by the argument. When the ISO is found set it up as a loop device. Device containing this ISO image will stay mounted at */run/initramfs/iscandev*. Using *iso-scan/filename* with a Fedora/Red Hat/CentOS Live iso should just work by copying the original kernel cmdline parameters. Requires the dracut *dmsquash-live* module.

Example

```
menuentry 'Live' --class gnu-linux --class gnu --class os {
    set isolabel=Live-x86_64
    set isofile="/boot/iso/Live-x86_64.iso"
    loopback loop $isofile
    linux (loop)/isolinux/vmlinuz0 boot=isolinux iso-scan/filename=$isofile root=live
    initrd (loop)/isolinux/initrd0.img
}
```

Misc

rd.emergency=[reboot|poweroff|halt]

specify, what action to execute in case of a critical failure. *rd.shell=0* must also be specified.

rd.driver.blacklist=<drivername>[,<drivername>,...]

do not load kernel module *<drivername>*. This parameter can be specified multiple times.

rd.driver.pre=<drivername>[,<drivername>,...]

force loading kernel module <drivername>. This parameter can be specified multiple times.

rd.driver.post=<drivername>[,<drivername>,...]

force loading kernel module <drivername> after all automatic loading modules have been loaded. This parameter can be specified multiple times.

rd.driver.export

making all drivers from initramfs available to rootfs after switch_root. This parameter can be set to *force* for exporting over existing modules. Requires the dracut *kernel-modules-export* module.

rd.retry=<seconds>

specify how long dracut should retry the initqueue to configure devices. The default is 180 seconds. After 2/3 of the time, degraded raids are force started. If you have hardware, which takes a very long time to announce its drives, you might want to extend this value. Requires the dracut *initqueue* module.

rd.timeout=<seconds>

specify how long dracut should wait for devices to appear. The default is 0, which means *forever*. Note that this timeout should be longer than rd.retry to allow for proper configuration.

rd.noverifyssl

accept self-signed certificates for ssl downloads. Requires the dracut *url-lib* module.

rd.ctty=<terminal device>

specify the controlling terminal for the console. This is useful, if you have multiple "console=" arguments.

rd.shutdown.timeout.umount=<seconds>

specify how long dracut should wait for an individual umount to finish during shutdown. This avoids the system from blocking when unmounting a file system cannot complete and waits indefinitely. Value 0 means to wait *forever*. The default is 90 seconds. Requires the dracut *shutdown* module.

Debug

If you are dropped to an emergency shell, the file */run/initramfs/rdsosreport.txt* is created, which can be saved to a (to be mounted by hand) partition (usually /boot) or a USB stick. Additional debugging info can be produced by adding **rd.debug** to the kernel command line. */run/initramfs/rdsosreport.txt* contains all logs and the output of some tools. It should be attached to any report about dracut problems.

rd.info

print informational output though "quiet" is set

rd.shell

allow dropping to a shell, if root mounting fails

rd.debug

set -x for the dracut shell. If systemd is active in the initramfs, all output is logged to the systemd journal, which you can inspect with "journalctl -ab". If systemd is not active, the logs are written to dmesg and */run/initramfs/init.log*. If "quiet" is set, it also logs to the console.

rd.memdebug=[0-5]

Print memory usage info at various points, set the verbose level from 0 to 5.

Higher level means more debugging output:

0 - no output

- 1 - partial /proc/meminfo
- 2 - /proc/meminfo
- 3 - /proc/meminfo + /proc/slabinfo
- 4 - /proc/meminfo + /proc/slabinfo + memstrack summary

NOTE: memstrack is a memory tracing tool that tracks the total memory consumption, and peak memory consumption of each kernel modules and userspace progress during the whole initramfs runtime, report is generated and the end of initramfs run.
- 5 - /proc/meminfo + /proc/slabinfo + memstrack (with top memory stacktrace)

NOTE: memstrack (with top memory stacktrace) will print top memory allocation stack traces during the whole initramfs runtime.

rd.break

drop to a shell at the end

rd.break={cmdline|pre-udev|pre-trigger|initqueue|pre-mount|mount|pre-pivot|cleanup}

drop to a shell before the defined breakpoint starts. This parameter can be specified multiple times.

rd.udev.log_level={err|info|debug}

set udev log level. The default is *err*.

I18N

Requires the dracut *i18n* module.

rd.vconsole.keymap=<keymap base file name>

keyboard translation table loaded by loadkeys; taken from keymaps directory; will be written as KEYMAP to */etc/vconsole.conf* in the initramfs.

Example

```
rd.vconsole.keymap=de-latin1-nodeadkeys
```

rd.vconsole.keymap.ext=<list of keymap base file names>

list of extra keymaps to be loaded (sep. by space); will be written as EXT_KEYMAP to */etc/vconsole.conf* in the initramfs

rd.vconsole.unicode

boolean, indicating UTF-8 mode; will be written as UNICODE to */etc/vconsole.conf* in the initramfs

rd.vconsole.font=

console font; taken from consolefonts directory; will be written as FONT to */etc/vconsole.conf* in the initramfs.

Example

```
rd.vconsole.font=eurlatgr
```

rd.vconsole.font.map=<console map base file name>

see description of *-m* parameter in setfont manual; taken from consoletrans directory; will be written as FONT_MAP to */etc/vconsole.conf* in the initramfs

rd.vconsole.font.unimap=<unicode table base file name>

see description of *-u* parameter in setfont manual; taken from unimaps directory; will be written as FONT_UNIMAP to */etc/vconsole.conf* in the initramfs

rd.locale.LANG=<locale>

taken from the environment; if no UNICODE is defined we set its value in basis of LANG value (whether it ends with ".utf8" (or similar) or not); will be written as LANG to */etc/locale.conf* in the initramfs.

Example

```
rd.locale.LANG=pl_PL.utf8
```

rd.locale.LC_ALL=<locale>

taken from the environment; will be written as LC_ALL to */etc/locale.conf* in the initramfs

LVM

Requires the dracut *lvm* module.

rd.lvm=0

disable LVM detection

rd.lvm.vg=<volume group name>

only activate all logical volumes in the the volume groups with the given name. *rd.lvm.vg* can be specified multiple times on the kernel command line.

rd.lvm.lv=<volume group name>/<logical volume name>

only activate the logical volumes with the given name. *rd.lvm.lv* can be specified multiple times on the kernel command line.

rd.lvm.conf=0

remove any */etc/lvm/lvm.conf*, which may exist in the initramfs

crypto LUKS

Requires the dracut *crypt* module.

rd.luks=0

disable crypto LUKS detection

rd.luks.uuid=<luks uuid>

only activate the LUKS partitions with the given UUID. Any "luks-" of the LUKS UUID is removed before comparing to *<luks uuid>*. The comparisons also matches, if *<luks uuid>* is only the beginning of the LUKS UUID, so you don't have to specify the full UUID. This parameter can be specified multiple times. *<luks uuid>* may be prefixed by the keyword *keysource:*, see *rd.luks.key* below.

rd.luks.allow-discards=<luks uuid>

Allow using of discards (TRIM) requests for LUKS partitions with the given UUID. Any "luks-" of the LUKS UUID is removed before comparing to *<luks uuid>*. The comparisons also matches, if *<luks uuid>* is only the beginning of the LUKS UUID, so you don't have to specify the full UUID. This parameter can be specified multiple times.

rd.luks.allow-discards

Allow using of discards (TRIM) requests on all LUKS partitions.

rd.luks.crypttab=0

do not check, if LUKS partition is in */etc/crypttab*

rd.luks.timeout=<seconds>

specify how long dracut should wait when waiting for the user to enter the password. This avoid blocking the boot if no password is entered. It does not apply to luks key. The default is 0, which

means *forever*.

crypto LUKS – key on removable device support

NB: If systemd is included in the dracut initrd, dracut's built in removable device keying support won't work. systemd will prompt for a password from the console even if you've supplied **rd.luks.key**. You may be able to use standard systemd `man:fstab[5,external]` syntax to get the same effect. If you do need **rd.luks.key** to work, you will have to exclude the "systemd" dracut module and any modules that depend on it. See `man:dracut.conf[5]` and https://bugzilla.redhat.com/show_bug.cgi?id=905683 for more information.

rd.luks.key=<keypath>[:<keydev>[:<luksdev>]]

<keypath> is the pathname of a key file, relative to the root of the filesystem on some device. It's REQUIRED. When <keypath> ends with *.gpg* it's considered to be key encrypted symmetrically with GPG. You will be prompted for the GPG password on boot. GPG support requires the *crypt-gpg* dracut module, which needs to be added explicitly.

<keydev> identifies the device on which the key file resides. It may be the kernel name of the device (should start with *"/dev/"*), a UUID (prefixed with "UUID=") or a label (prefix with "LABEL="). You don't have to specify a full UUID. Just its beginning will suffice, even if its ambiguous. All matching devices will be probed. This parameter is recommended, but not required. If it's not present, all block devices will be probed, which may significantly increase boot time.

If <luksdev> is given, the specified key will only be used for the specified LUKS device. Possible values are the same as for <keydev>. Unless you have several LUKS devices, you don't have to specify this parameter. The simplest usage is:

Example

```
rd.luks.key=/foo/bar.key
```

As you see, you can skip colons in such a case.

Note

Your LUKS partition must match your key file.

dracut provides keys to cryptsetup with *-d* (an older alias for *--key-file*). This uses the entire binary content of the key file as part of the secret. If you pipe a password into cryptsetup **without** *-d* or *--key-file*, it will be treated as text user input, and only characters before the first newline will be used. Therefore, when you're creating an encrypted partition for dracut to mount, and you pipe a key into *cryptsetup luksFormat*, you must use *-d -*.

Here is an example for a key encrypted with GPG (warning: *--batch-mode* will overwrite the device without asking for confirmation):

```
gpg --quiet --decrypt rootkey.gpg | \
cryptsetup --batch-mode --key-file - \
luksFormat /dev/sda47
```

If you use unencrypted key files, just use the key file pathname instead of the standard input. For a random key with 256 bits of entropy, you might use:

```
head -32c /dev/urandom > rootkey.key
cryptsetup --batch-mode --key-file rootkey.key \
luksFormat /dev/sda47
```

You can also use regular key files on an encrypted *keydev*.

Compared to using GPG encrypted keyfiles on an unencrypted device this provides the following advantages:

- you can unlock your disk(s) using multiple passphrases
- better security by not losing the key stretching mechanism

To use an encrypted *keydev* you **must** ensure that it becomes available by using the keyword *keysource*, e.g. `rd.luks.uuid=keysource:aaaa aaaa` being the uuid of the encrypted *keydev*.

Example:

Lets assume you have three disks *A*, *B* and *C* with the uuids *aaaa*, *bbbb* and *cccc*.
You want to unlock *A* and *B* using keyfile *keyfile*.
The unlocked volumes be *A'*, *B'* and *C'* with the uuids *AAAA*, *BBBB* and *CCCC*.
keyfile is saved on *C'* as */keyfile*.

One luks keyslot of each *A*, *B* and *C* is setup with a passphrase.
Another luks keyslot of each *A* and *B* is setup with *keyfile*.

To boot this configuration you could use:

```
rd.luks.uuid=aaaa
rd.luks.uuid=bbbb
rd.luks.uuid=keysource:cccc
rd.luks.key=/keyfile:UUID=CCCC
```

Dracut asks for the passphrase for *C* and uses the keyfile to unlock *A* and *B*.
If getting the passphrase for *C* fails it falls back to asking for the passphrases for *A* and *B*.

If you want *C'* to stay unlocked, specify a luks name for it, e.g. `rd.luks.name=cccc=mykeys`, otherwise it gets closed when not needed anymore.

rd.luks.key.tout=0

specify how many times dracut will try to read the keys specified in `rd.luks.key`. This gives a chance to the removable device containing the key to initialize.

MD RAID

Requires the dracut *mdraid* module.

rd.md=0

disable MD RAID detection

rd.md.imsm=0

disable MD RAID for imsm/isw raids, use DM RAID instead

rd.md.ddf=0

disable MD RAID for SNIA ddf raids, use DM RAID instead

rd.md.conf=0

ignore `mdadm.conf` included in `initramfs`

rd.md.waitclean=1

wait for any resync, recovery, or reshape activity to finish before continuing

rd.md.uuid=<md raid uuid>

only activate the raid sets with the given UUID. This parameter can be specified multiple times.

DM RAID

Requires the dracut *dmraid* module.

rd.dm=0

disable DM RAID detection

rd.dm.uuid=<dm raid uuid>

only activate the raid sets with the given UUID. This parameter can be specified multiple times.

MULTIPATH

Requires the dracut *multipath* module.

rd.multipath=0

disable multipath detection

rd.multipath=default

use default multipath settings

FIPS

Requires the dracut *fips* module.

rd.fips

enable FIPS

boot=<boot device>

specify the device, where /boot is located.

Example

boot=/dev/sda1

boot=/dev/disk/by-path/pci-0000:00:1f.1-scsi-0:0:1:0-part1

boot=UUID=<uuid>

boot=LABEL=<label>

rd.fips.skipkernel

skip checksum check of the kernel image. Useful, if the kernel image is not in a separate boot partition.

Network

Important

It is recommended to either bind an interface to a MAC with the **ifname** argument, or to use the systemd-udev predictable network interface names.

Predictable network interface device names based on:

- firmware/bios-provided index numbers for on-board devices
- firmware-provided pci-express hotplug slot index number
- physical/geographical location of the hardware
- the interface's MAC address

See: <http://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames>

Two character prefixes based on the type of interface:

en
ethernet

wl
wlan

ww
wwan

Type of names:

o<index>
on-board device index number

s<slot>[f<function>][d<dev_id>]
hotplug slot index number

x<MAC>
MAC address

[P<domain>]p<bus>s<slot>[f<function>][d<dev_id>]
PCI geographical location

[P<domain>]p<bus>s<slot>[f<function>][u<port>][.][c<config>][i<interface>]
USB port number chain

All multi-function PCI devices will carry the [f<function>] number in the device name, including the function 0 device.

When using PCI geography, The PCI domain is only prepended when it is not 0.

For USB devices the full chain of port numbers of hubs is composed. If the name gets longer than the maximum number of 15 characters, the name is not exported. The usual USB configuration == 1 and interface == 0 values are suppressed.

PCI ethernet card with firmware index "1"

- eno1

PCI ethernet card in hotplug slot with firmware index number

- ens1

PCI ethernet multi-function card with 2 ports

- enp2s0f0
- enp2s0f1

PCI wlan card

- wlp3s0

USB built-in 3G modem

- wwp0s29u1u4i6

USB Android phone

- enp0s29u1u2

The following options are supported by the *network-legacy* dracut module. Other network modules might support a slightly different set of options; refer to the documentation of the specific network module in use. For NetworkManager, see `man:nm-initrd-generator[8]`.

ip={*dhcp|on|any|dhcp6|auto6|either6|link6|single-dhcp*}

dhcp|on|any

get ip from dhcp server from all interfaces. If `netroot=dhcp`, loop sequentially through all interfaces (`eth0`, `eth1`, ...) and use the first with a valid DHCP root-path.

single-dhcp

Send DHCP on all available interfaces in parallel, as opposed to one after another. After the first DHCP response is received, stop DHCP on all other interfaces. This gives the fastest boot time by using the IP on interface for which DHCP succeeded first during early boot. Caveat: Does not apply to Network Manager.

auto6

IPv6 autoconfiguration

dhcp6

IPv6 DHCP

either6

if *auto6* fails, then *dhcp6*

link6

bring up interface for IPv6 link-local addressing

ip=<interface>:{*dhcp|on|any|dhcp6|auto6|link6*}[:[<mtu>]][:<macaddr>]]

This parameter can be specified multiple times.

dhcp|on|any|dhcp6

get ip from dhcp server on a specific interface

auto6

do IPv6 autoconfiguration

link6

bring up interface for IPv6 link local address

<macaddr>

optionally **set** <macaddr> on the <interface>. This cannot be used in conjunction with the **ifname** argument for the same <interface>.

ip=<client-IP>[:[<peer>]]:<gateway-IP>:<netmask>:<client_hostname>:<interface>:{*none|off|dhcp|on|any|dhcp6|auto6*}

explicit network configuration. If you want to define a IPv6 address, put it in brackets (e.g. `[2001:DB8::1]`). This parameter can be specified multiple times. <peer> is optional and is the address of the remote endpoint for pointpoint interfaces and it may be followed by a slash and a decimal

number, encoding the network prefix length.

<macaddr>

optionally **set** **<macaddr>** on the **<interface>**. This cannot be used in conjunction with the **ifname** argument for the same **<interface>**.

ip=<client-IP>[:<peer>]:<gateway-IP>:<netmask>:<client_hostname>:<interface>:{none|off|dhcp|on|any|dhcp6|auto6|}
explicit network configuration. If you want to define a IPv6 address, put it in brackets (e.g. [2001:DB8::1]). This parameter can be specified multiple times. **<peer>** is optional and is the address of the remote endpoint for pointpoint interfaces and it may be followed by a slash and a decimal number, encoding the network prefix length.

ifname=<interface>:<MAC>

Assign network device name **<interface>** (i.e. "bootnet") to the NIC with MAC **<MAC>**.

Warning

Do **not** use the default kernel naming scheme for the interface name, as it can conflict with the kernel names. So, don't use "eth[0-9]+" for the interface name. Better name it "bootnet" or "bluesocket".

rd.route=<net>/<netmask>:<gateway>[:<interface>]

Add a static route with route options, which are separated by a colon. IPv6 addresses have to be put in brackets.

Example

```
rd.route=192.168.200.0/24:192.168.100.222:ens10
rd.route=192.168.200.0/24:192.168.100.222
rd.route=192.168.200.0/24::ens10
rd.route=[2001:DB8:3::/8]:[2001:DB8:2::1]:ens10
```

bootdev=<interface>

specify network interface to use routing and netroot information from. Required if multiple ip= lines are used.

BOOTIF=<MAC>

specify network interface to use routing and netroot information from.

rd.bootif=0

Disable BOOTIF parsing, which is provided by PXE

nameserver=<IP> [nameserver=<IP> ...]

specify nameserver(s) to use

rd.peerdns=0

Disable DNS setting of DHCP parameters.

biosdevname=0

boolean, turn off biosdevname network interface renaming

rd.neednet=1

boolean, bring up network even without netroot set

vlan=<vlanname>:<phydevice>

Setup vlan device named **<vlanname>** on **<phydevice>**. We support the four styles of vlan names:

VLAN_PLUS_VID (vlan0005), VLAN_PLUS_VID_NO_PAD (vlan5), DEV_PLUS_VID (eth0.0005), DEV_PLUS_VID_NO_PAD (eth0.5)

bond=<bondname>[:<bondslaves>[:<options>[:<mtu>]]]

Setup bonding device <bondname> on top of <bondslaves>. <bondslaves> is a comma-separated list of physical (ethernet) interfaces. <options> is a comma-separated list on bonding options (modinfo bonding for details) in format compatible with initscripts. If <options> includes multi-valued arp_ip_target option, then its values should be separated by semicolon. if the mtu is specified, it will be set on the bond master. Bond without parameters assumes bond=bond0:eth0,eth1:mode=balance-rr

team=<teammaster>:<teamslaves>[:<teamrunner>]

Setup team device <teammaster> on top of <teamslaves>. <teamslaves> is a comma-separated list of physical (ethernet) interfaces. <teamrunner> is the runner type to be used (see man:teamd.conf[5,external]); defaults to activebackup. Team without parameters assumes team=team0:eth0,eth1:activebackup

bridge=<bridgename>:<ethnames>

Setup bridge <bridgename> with <ethnames>. <ethnames> is a comma-separated list of physical (ethernet) interfaces. Bridge without parameters assumes bridge=br0:eth0

NFS

Requires the dracut *nfs* module.

root=_<root-dir>_[:<nfs-options>]

mount nfs share from <server-ip>:/<root-dir>, if no server-ip is given, use dhcp next_server. If server-ip is an IPv6 address it has to be put in brackets, e.g. [2001:DB8::1]. NFS options can be appended with the prefix ":" or "," and are separated by ",".

root=nfs[:<server-ip>:]_<root-dir>_[:<nfs-options>],

root=nfs4[:<server-ip>:]_<root-dir>_[:<nfs-options>], **root**={dhcp|dhcp6}

netroot=dhcp alone directs initrd to look at the DHCP root-path where NFS options can be specified.

Example

```
root-path=<server-ip>:<root-dir>[,<nfs-options>]
root-path=nfs:<server-ip>:<root-dir>[,<nfs-options>]
root-path=nfs4:<server-ip>:<root-dir>[,<nfs-options>]
```

root=/dev/nfs nfsroot=_<root-dir>_[:<nfs-options>]

Deprecated! kernel Documentation/filesystems/nfsroot.txt defines this method. This is supported by dracut, but not recommended.

rd.nfs.domain=<NFSv4 domain name>

Set the NFSv4 domain name. Will override the settings in /etc/idmap.conf.

rd.net.dhcp.retry=<cnt>

If this option is set, dracut will try to connect via dhcp <cnt> times before failing. Default is 1.

rd.net.timeout.dhcp=<arg>

If this option is set, dhclient is called with "--timeout <arg>".

rd.net.timeout.iflink=<seconds>

Wait <seconds> until link shows up. Default is 60 seconds.

rd.net.timeout.ifup=<seconds>

Wait <seconds> until link has state "UP". Default is 20 seconds.

rd.net.timeout.route=<seconds>

Wait <seconds> until route shows up. Default is 20 seconds.

rd.net.timeout.ipv6dad=<seconds>

Wait <seconds> until IPv6 DAD is finished. Default is 50 seconds.

rd.net.timeout.ipv6auto=<seconds>

Wait <seconds> until IPv6 automatic addresses are assigned. Default is 40 seconds.

rd.net.timeout.carrier=<seconds>

Wait <seconds> until carrier is recognized. Default is 10 seconds.

CIFS

Requires the dracut *cifs* module.

root=cifs://[<username>[:<password>]@]<server-ip>:<root-dir>

mount cifs share from <server-ip>:/<root-dir>, if no server-ip is given, use dhcp next_server. if server-ip is an IPv6 address it has to be put in brackets, e.g. [2001:DB8::1]. If a username or password are not specified as part of the root, then they must be passed on the command line through cifsuser/cifspass.

Warning

Passwords specified on the kernel command line are visible for all users via the file */proc/cmdline* and via dmesg or can be sniffed on the network, when using DHCP with DHCP root-path.

cifsuser=<username>

Set the cifs username, if not specified as part of the root.

cifspass=<password>

Set the cifs password, if not specified as part of the root.

Warning

Passwords specified on the kernel command line are visible for all users via the file */proc/cmdline* and via dmesg or can be sniffed on the network, when using DHCP with DHCP root-path.

iSCSI

Requires the dracut *iscsi* module.

root=iscsi:[<username>:<password>[:<reverse>:<password>]@][<servername>]:[<protocol>]:[<port>][:[<iscsi_iface_n

protocol defaults to "6", LUN defaults to "0". If the "servername" field is provided by BOOTP or DHCP, then that field is used in conjunction with other associated fields to contact the boot server in the Boot stage. However, if the "servername" field is not provided, then the "targetname" field is then used in the Discovery Service stage in conjunction with other associated fields. See rfc4173 <<http://tools.ietf.org/html/rfc4173#section-5>>.

Warning

Passwords specified on the kernel command line are visible for all users via the file */proc/cmdline* and via dmesg or can be sniffed on the network, when using DHCP with DHCP root-path.

Example

```
root=iscsi:192.168.50.1::::iqn.2009-06.dracut:target0
```

If servername is an IPv6 address, it has to be put in brackets:

Example

```
root=iscsi:[2001:DB8::1]:::iqn.2009-06.dracut:target0
```

root=???

netroot=iscsi:`[<username>:<password>[:<reverse>:<password>]@][<servername>]:[<protocol>]:[<port>][:<iscsi_iface>]`

...

multiple netroot options allow setting up multiple iscsi disks:

Example

```
root=UUID=12424547
netroot=iscsi:192.168.50.1:::iqn.2009-06.dracut:target0
netroot=iscsi:192.168.50.1:::iqn.2009-06.dracut:target1
```

If servername is an IPv6 address, it has to be put in brackets:

Example

```
netroot=iscsi:[2001:DB8::1]:::iqn.2009-06.dracut:target0
```

Warning

Passwords specified on the kernel command line are visible for all users via the file `/proc/cmdline` and via `dmesg` or can be sniffed on the network, when using DHCP with DHCP root-path. You may want to use `rd.iscsi.firmware`.

root=??? rd.iscsi.initiator=<initiator> rd.iscsi.target.name=<target name> rd.iscsi.target.ip=<target ip> rd.iscsi.target.port=<target port> rd.iscsi.target.group=<target group> rd.iscsi.username=<username> rd.iscsi.password=<password> rd.iscsi.in.username=<in username> rd.iscsi.in.password=<in password>

manually specify all iscsistart parameter (see `iscsistart --help`)

Warning

Passwords specified on the kernel command line are visible for all users via the file `/proc/cmdline` and via `dmesg` or can be sniffed on the network, when using DHCP with DHCP root-path. You may want to use `rd.iscsi.firmware`.

root=??? netroot=iscsi rd.iscsi.firmware=1

will read the iscsi parameter from the BIOS firmware

rd.iscsi.login_retry_max=<num>

maximum number of login retries

rd.iscsi.param=<param>

`<param>` will be passed as `--param <param>` to `iscsistart`. This parameter can be specified multiple times.

Example

```
"netroot=iscsi rd.iscsi.firmware=1 rd.iscsi.param=node.session.timeo.replacement_
```

will result in

`iscsistart -b --param node.session.timeo.replacement_timeout=30`

rd.iscsi.ibft **rd.iscsi.ibft=1**: Turn on iBFT autoconfiguration for the interfaces

rd.iscsi.mp **rd.iscsi.mp=1**: Configure all iBFT interfaces, not only used for booting (multipath)

rd.iscsi.waitnet=0: Turn off waiting for all interfaces to be up before trying to login to the iSCSI targets.

rd.iscsi.testroute=0: Turn off checking, if the route to the iSCSI target IP is possible before trying to login.

rd.iscsi.transport=*<transport_name>*

Set the iSCSI transport name (see `man:iscsiadm[8,external]`). iSCSI offload transports like **bnx2i** don't need the network to be up in order to bring up iSCSI connections. This parameter indicates that network setup can be skipped in the initramfs, which makes booting with iSCSI offload cards faster and more reliable. This parameter currently only has an effect for *<transport_name>=bnx2i* or *<transport_name>=qedi*.

FCoE

Requires the dracut *fcoe* module.

rd.fcoe=0

disable FCoE and lldpad

fcoe=*<edd|interface|MAC>:{dcb|nodcb}:{fabric|vn2vn}*

Try to connect to a FCoE SAN through the NIC specified by *<interface>* or *<MAC>* or EDD settings. The second argument specifies if DCB should be used. The optional third argument specifies whether fabric or VN2VN mode should be used. This parameter can be specified multiple times.

Note

letters in the MAC-address must be lowercase!

NVMf

Requires the dracut *nvmf* module.

rd.nonvmf

Disable NVMf

rd.nvmf.nonbft

Disable connecting to targets from the NVMe Boot Firmware Table. Without this parameter, NBFT connections will take precedence over *rd.nvmf.discover*.

rd.nvmf.nostatic

Disable connecting to targets that have been statically configured when the initramfs was built. Targets specified with *rd.nvmf.discover* on the kernel command line will still be tried.

rd.nvmf.hostnqn=*<hostNQN>*

NVMe host NQN to use

rd.nvmf.hostid=*<hostID>*

NVMe host id to use

rd.nvmf.discover=*{rdma|fc|tcp},<traddr>,[<host_traddr>],[<trsvcid>]*

Discover and connect to a NVMe-over-Fabric controller specified by *<traddr>* and the optionally *<host_traddr>* or *<trsvcid>*. The first argument specifies the transport to use; currently only *rdma*, *fc*, or *tcp* are supported. This parameter can be specified multiple times.

Examples

```
rd.nvmf.discover=tcp,192.168.10.10,,4420
```

```
rd.nvmf.discover=fc,nn-0x201700a05634f5bf:pn-0x201900a05634f5bf,nn-0x200000109b57
```

rd.nvmf.discover=fc,auto

This special syntax determines that Fibre Channel autodiscovery is to be used rather than regular NVMe discovery. It takes precedence over all other *rd.nvmf.discover=* arguments.

NBD

Requires the dracut *nbd* module.

```
root=??? netroot=nbd:<server>[:<port/exportname>[:<fstype>[:<mountopts>[:<nbdopts>]]]  
mount nbd share from <server>.
```

NOTE: If "exportname" instead of "port" is given the standard port is used. Newer versions of nbd are only supported with "exportname".

root=/dev/root netroot=dhcp with dhcp

```
root-path=nbd:<server>[:<port/exportname>[:<fstype>[:<mountopts>[:<nbdopts>]]]  
netroot=dhcp alone directs initrd to look at the DHCP root-path where NBD options can be specified.  
This syntax is only usable in cases where you are directly mounting the volume as the rootfs.
```

NOTE: If "exportname" instead of "port" is given the standard port is used. Newer versions of nbd are only supported with "exportname".

VIRTIOFS

Requires the dracut *virtiofs* module.

```
root=virtiofs:<mount-tag>  
mount virtiofs share using the tag <mount-tag>. The tag name is arbitrary and must match the tag  
given in the qemu -device command.
```

```
rootfstype=virtiofs root=<mount-tag>  
mount virtiofs share using the tag <mount-tag>. The tag name is arbitrary and must match the tag  
given in the qemu -device command.
```

Both formats are supported by the *virtiofs* dracut module. See <https://gitlab.com/virtio-fs/virtiofsd> for more information.

Example

```
root=virtiofs:host rw
```

DASD

Requires the dracut *dasd* module.

rd.dasd=...

same syntax as the kernel module parameter (s390 only). For more details on the syntax see the IBM book "Linux on IBM Z and IBM LinuxONE – Device Drivers, Features, and Commands" <https://www.ibm.com/docs/en/linux-on-systems?topic=overview-device-drivers-features-commands>. This parameter can be specified multiple times.

NOTE: This parameter is no longer handled by dracut itself but with the exact same syntax by <https://github.com/ibm-s390-linux/s390-tools/tree/master/zdev/dracut/95zdev>.

ZFCP

Requires the dracut *zfc*p module.

rd.zfcp=<zfc adaptor device bus ID>,<WWPN>,<FCPLUN>

rd.zfc can be specified multiple times on the kernel command line.

NOTE: This parameter is no longer handled by dracut itself but with the exact same syntax by <https://github.com/ibm-s390-linux/s390-tools/tree/master/zdev/dracut/95zdev>.

rd.zfcp=<zfc adaptor device bus ID>

If NPIV is enabled and the *allow_lun_scan* parameter to the zfc module is set to Y then the zfc driver will be initiating a scan internally and the <WWPN> and <FCPLUN> parameters can be omitted.

NOTE: This parameter is no longer handled by dracut itself but with the exact same syntax by <https://github.com/ibm-s390-linux/s390-tools/tree/master/zdev/dracut/95zdev>.

Example

```
rd.zfc=0.0.4000,0x5005076300C213e9,0x5022000000000000
rd.zfc=0.0.4000
```

rd.zfc.conf=0

ignore zfc.conf included in the initramfs

ZNET

Requires the dracut *znet* module.

rd.znet=<nettype>,<subchannels>,<options>

Activates a channel-attached network interface on s390 architecture. <nettype> is one of: qeth, lcs, etc. <subchannels> is a comma-separated list of ccw device bus-IDs. The list consists of 3 entries with nettype qeth, and 2 for other nettype. <options> is a comma-separated list of <name>=<value> pairs, where <name> refers to a device sysfs attribute to which <value> gets written. rd.znet can be specified multiple times on the kernel command line.

rd.znet_ifname=<ifname>:<subchannels>

Assign network device name <interface> (i.e. "bootnet") to the NIC corresponds to the subchannels. This is useful when dracut's default "ifname=" doesn't work due to device having a changing MAC address.

Example

```
rd.znet=qeth,0.0.0600,0.0.0601,0.0.0602,layer2=1,portname=foo
rd.znet=ctc,0.0.0600,0.0.0601,protocol=bar
```

OverlayFS

Requires the dracut *overlayfs* module.

rd.overlay

Enables the use of the **OverlayFS** kernel module, if available, to provide a copy-on-write union directory for the root filesystem. OverlayFS overlays are directories of the files that have changed on the read-only base (lower) filesystem. The root filesystem is provided through a special overlay type mount that merges at least two directories, designated the lower and the upper. If an OverlayFS upper directory is not present on the boot device, a tmpfs directory will be created at /run/overlayfs to provide temporary storage. Persistent storage can be provided on vfat or msdos formatted devices by supplying the OverlayFS upper directory within an embedded filesystem that supports the creation of

trusted.* extended attributes and provides a valid `d_type` in `readdir` responses, such as with `btrfs`, `ext4`, `f2fs`, & `xfs`. On non-`vfat`-formatted devices, a persistent OverlayFS overlay can extend the available root filesystem storage up to the capacity of the LiveOS disk device.

The **`rd.overlay.readonly`** option, which allows a persistent overlayfs to be mounted read-only through a higher level transient overlay directory, has been implemented through the multiple lower layers feature of OverlayFS.

`rd.overlay=tmpfs:[size=<size>][,nr_blocks=<n>][,nr_inodes=<n>]`

Mounts a dedicated tmpfs with the specified options for the OverlayFS upper directory, instead of using the default `/run tmpfs`. This allows controlling the size of the writable overlay at boot time, which is useful for deployments where the available RAM is known in advance (e.g. MAAS ephemeral environments).

Supported options (comma-separated):

- `size=<size>` – Maximum size of the tmpfs (e.g. 4G, 512M, 25%). Accepts the same values as the tmpfs `size=` mount option.
- `nr_blocks=<n>` – Maximum number of blocks allocated to the tmpfs.
- `nr_inodes=<n>` – Maximum number of inodes in the tmpfs.

Examples

```
rd.overlay=tmpfs:size=4G
rd.overlay=tmpfs:size=25%
rd.overlay=tmpfs:size=512M,nr_inodes=100000
```

`rd.overlay={LABEL=<label>|UUID=<uuid>|PARTUUID=<uuid>|PARTLABEL=<label>|/dev/<device>}`

Specifies the storage device for persistent overlay (for both live images and regular OverlayFS). When a block device is specified, it will be mounted and used for persistent overlay storage. For live images with `LABEL=` specification, if the labeled partition does not exist, it will be automatically created using available free space on the device.

Supported device specifications:

- `LABEL=<label>` – Use the device with the specified filesystem label.
- `UUID=<uuid>` – Use the device with the specified UUID.
- `PARTUUID=<uuid>` – Use the device with the specified partition UUID.
- `PARTLABEL=<label>` – Use the device with the specified partition label.
- `/dev/<device>` – Use the specified device path directly.

Examples

```
rd.overlay=LABEL=persist
rd.overlay=UUID=12345678-1234-1234-1234-123456789abc
rd.overlay=/dev/sdb1
```

`rd.overlay.crypt=<device>[,<option>]...`

Requires the dracut `overlayfs-crypt` module.

Specifies an encrypted (LUKS) device for persistent overlay storage. If the device already contains a valid LUKS volume, the user will be prompted for the passphrase interactively (via Plymouth if available, otherwise via TTY). Otherwise, the device will be wiped, formatted with LUKS using a

randomly generated passphrase, and a new filesystem will be created on it. The random passphrase is stored in `/run/initramfs/overlayfs.passwd` for the duration of the boot.

The device is specified as the first argument, followed by optional comma-separated options:

- `<device>` – (Required) Device specification: `LABEL=`, `UUID=`, `PARTLABEL=`, `PARTUUID=`, or `/dev/<device>`.
- `mapname=<name>` – Device mapper name (default: `overlay-crypt`).
- `fstype=<type>` – Filesystem type: `ext2`, `ext3`, or `ext4` (default: `ext4`).
- `mkfs=<0|1>` – Filesystem creation behavior: `0`=never create (require existing LUKS), `1`=create if needed (default).
- `timeout=<seconds>` – Time to wait for the device to appear. By default, the global `rd.timeout` value is used. Set to `0` to skip waiting entirely.

Examples

```
rd.overlay.crypt=LABEL=CRYPTOVL
rd.overlay.crypt=UUID=12345678-1234-1234-1234-123456789abc, fstype=ext3
```

Bootable live images

Requires the dracut `dmsquash-live` module. Dracut offers multiple options for live booted images:

SquashFS (read-only) base filesystem image

Note — There are 3 separate overlay types available:

- Device-mapper snapshots (the original offering),
- Device-mapper thin provisioning snapshots (see `rd.live.overlay.thin`, a later offering), and
- OverlayFS based overlay mounts (a more recent offering).

Using one of these technologies, the system will provide a writable overlay for the base, read-only SquashFS root filesystem. These methods enable a relatively fast boot and lower RAM usage.

With the original Device-mapper snapshot overlay, users **must be careful** to avoid writing too many blocks to the snapshot device. Once the blocks of the snapshot overlay are exhausted, the whole root filesystem becomes read-only leading to application failures. The snapshot overlay device is marked *Overflow*, and a difficult recovery is required to repair and enlarge the overlay offline.

When `rd.overlay=` is not specified for persistent overlay storage, or the specified file is not found or writable, a Device-mapper snapshot based non-persistent or temporary overlay is automatically created as a sparse file in RAM of the initramfs. This file will only consume content space as required blocks are allocated. This snapshot based overlay defaults to an apparent size of 32 GiB in RAM, and can be adjusted with the `rd.live.overlay.size=` kernel command line option. This file is hidden (and appears deleted) when the boot process switches out of the initramfs to the main root filesystem but its loop device remains connected to the Device-mapper snapshot.

Even with large Device-mapper overlay files for write space, the available root filesystem capacity is limited by the total allocated size of the base root filesystem, which often provide only a small number of gigabytes of free space.

This shortage could be remedied by building the root filesystem with more allocated free space,

or the OverlayFS based overlay mount method can be used.

When the `rd.overlay` option is specified or when `rd.overlay=` points to an appropriate directory with a sister at `/../ovlwork`, then an OverlayFS based overlay mount is employed. Such a persistent OverlayFS overlay can extend the available root filesystem storage up to the capacity of the LiveOS disk device.

For non-persistent OverlayFS overlays, the `/run/overlayfs` directory in the `/run tmpfs` is used for temporary storage. This filesystem is typically sized to one half of the RAM total in the system.

The command: `mount -o remount,size=<nbytes> /run` will resize this virtual filesystem after booting.

The internal SquashFS structure is traditionally expected to be:

```
squashfs.img          | SquashFS from LiveCD .iso
!(mount)
/LiveOS
|- rootfs.img         | Usually a ext4 filesystem image to mount read-only
!(mount)
/bin                  | Base Live root filesystem
/boot                 |
/dev                  |
...                   |
```

For OverlayFS mount overlays, the internal SquashFS structure may be a direct compression of the root filesystem:

```
squashfs.img          | SquashFS from LiveCD .iso
!(mount)
/bin                  | Base Live root filesystem
/boot                 |
/dev                  |
...                   |
```

Dracut uses one of the overlay methods of live booting by default. No additional command line options are required other than **root=live:<path to blockdevice>** or **root=live:<URL>** to specify the location of your squashed root filesystem.

- The compressed SquashFS image can be copied during boot to RAM at `/run/initramfs/squashed.img` by using the **rd.live.ram=1** option.
- A device with a persistent overlay can be booted read-only by using the **rd.live.overlay.readonly** option on the kernel command line. This will either cause a temporary, writable overlay to be stacked over a read-only snapshot of the root filesystem or the OverlayFS mount will use an additional lower layer with the root filesystem.

Uncompressed live filesystem image

When the live system was installed with the `--skipcompress` option of the `livecd-iso-to-disk` installation script for Live USB devices, the root filesystem image, `rootfs.img`, is expanded on installation and no SquashFS is involved during boot.

- If **rd.live.ram=1** is used in this situation, the full, uncompressed root filesystem is copied during boot to `/run/initramfs/rootfs.img` in the `/run tmpfs`.

- If **rd.overlay=none** is provided as a kernel command line option, a writable, linear Device-mapper target is created on boot with no overlay.

Writable filesystem image

The system will retrieve a compressed filesystem image, extract it to `/run/initramfs/fsimg/rootfs.img`, connect it to a loop device, create a writable, linear Device-mapper target at `/dev/mapper/live-rw`, and mount that as a writable volume at `/`. More RAM is required during boot but the live filesystem is easier to manage if it becomes full. Users can make a filesystem image of any size and that size will be maintained when the system boots. There is no persistence of root filesystem changes between boots with this option.

The filesystem structure is expected to be:

<code>rootfs.tgz</code>		Compressed tarball containing filesystem image
<code>!(unpack)</code>		
<code>/rootfs.img</code>		Filesystem image at <code>/run/initramfs/fsimg/</code>
<code>!(mount)</code>		
<code>/bin</code>		Live filesystem
<code>/boot</code>		
<code>/dev</code>		
<code>...</code>		

To use this boot option, ensure that **rd.writable.fsimg=1** is in your kernel command line and add the **root=live:<URL>** to specify the location of your compressed filesystem image tarball or SquashFS image.

rd.writable.fsimg=1

Enables writable filesystem support. The system will boot with a fully writable (but non-persistent) filesystem without snapshots (*see notes above about available live boot options*). You can use the **rootflags** option to set mount options for the live filesystem as well (*see documentation about rootflags in the Standard section above*). **This implies that the whole image is copied to RAM before the boot continues.**

Note

There must be enough free RAM available to hold the complete image.

This method is very suitable for diskless boots.

rd.minmem=<megabyte>

Specify minimum free RAM in MB after copying a live disk image into memory. The default is 1024.

This parameter only applies together with the parameters `rd.writable.fsimg` or `rd.live.ram`.

root=live:<url>

Boots a live image retrieved from `<url>`. Requires the dracut *livenet* module. Valid handlers: *http*, *https*, *ftp*, *torrent*, *tftp*.

Examples

```
root=live:http://example.com/liveboot.img
root=live:ftp://ftp.example.com/liveboot.img
root=live:torrent://example.com/liveboot.img.torrent
```

root=<http-url>

This is an alias for **root=live:<url>**. Boots a live image retrieved from *<http-url>*. Requires the dracut *livenet* module. Valid handlers: *http*, *https*.

Examples

```
root=http://example.com/root.squashfs
root=https://example.com/live.sqfs
```

rd.live.debug=1

Enables debug output from the live boot process.

rd.live.dir=<path>

Specifies the directory within the boot device where the *squashfs.img* or *rootfs.img* can be found. By default, this is */LiveOS*.

rd.live.squashimg=<filename of SquashFS image>

Specifies the filename for a SquashFS image of the root filesystem. By default, this is *squashfs.img*.

rd.live.ram=1

Copy the complete image to RAM and use this for booting. This is useful when the image resides on, e.g., a DVD which needs to be ejected later on.

rd.overlay={<devspec>[:{<pathspec>|auto}]|none}

Manage the usage of a persistent overlay for live images.

- *<devspec>* specifies the path to a device with a mountable filesystem.
- *<pathspec>* is a path within the *<devspec>* filesystem to either
 - a file (that is loop mounted for a Device-mapper overlay) or
 - a directory (that is symbolically linked to */run/overlayfs* for a OverlayFS mount overlay). (A required sister directory *<pathspec>/../ovlwork* is automatically made.)
- *none* (the word itself) specifies that no overlay will be used, such as when an uncompressed, writable live root filesystem is available.

The above method shall be used to persist the changes made to the root filesystem specified within the **root=live:<path to blockdevice>** or **root=live:<url>** device.

The default *pathspec*, when **:auto** or no **:<pathspec>** is given, is */<rd.live.dir>/overlay-<label>-<uuid>*, where *<label>* and *<uuid>* are the LABEL and UUID of the filesystem specified by the **root=live:<path|url>** device.

If a persistent overlay *is detected* at the standard LiveOS path, and *rd.overlay* is not set to 1, the overlay type (either Device-mapper or OverlayFS) will be detected and it will be used.

Examples

```
rd.overlay=/dev/sdb1:/persistent-overlay.img
rd.overlay=UUID=99440c1f-8daa-41bf-b965-b7240a8996f4
```

rd.live.overlay.cowfs=[btrfs|ext4|xfs]

Specifies the filesystem to use when formatting the overlay partition. The default is *ext4*.

rd.live.overlay.size=<size_MiB>

Specifies a non-persistent Device-mapper overlay size in MiB. The default is 32768.

rd.live.overlay.readonly=1

This is used to boot in a read-only mode with a normally read-write persistent overlay. With this option,

- Device-mapper overlays will have an additional, non-persistent, writable snapshot overlay stacked over a read-only snapshot (`/dev/mapper/livero`) of the base root filesystem and the persistent overlay, or
- for writable `rootfs.img` images, the above over a read-only loop device, or
- an OverlayFS mount will link the persistent overlay directory at `/run/overlayfsr` as an additional read-only lower layer stacked over the base root filesystem, and `/run/overlayfs` becomes the temporary, writable, upper directory overlay, to complete the bootable root filesystem.

rd.live.overlay.nouserconfirmprompt=

Suppresses the *Using temporary overlay* blocking prompt that asks for a user confirmation before proceeding to boot. This allows the boot process to continue to completion without user interaction.

rd.overlay.reset=1

Specifies that a persistent overlay should be reset on boot. All previous root filesystem changes are vacated by this action.

rd.live.overlay.thin=1

Enables the usage of thin snapshots instead of classic dm snapshots. The advantage of thin snapshots is that they support discards, and will free blocks that are not claimed by the filesystem. In this use case, this means that memory is given back to the kernel when the filesystem does not claim it anymore.

ZIPL

Requires the dracut *zipl* module.

rd.zipl=<path to blockdevice>

Update the dracut commandline with the values found in the *dracut-cmdline.conf* file on the given device. The values are merged into the existing commandline values and the udev events are regenerated.

Example

```
rd.zipl=UUID=0fb28157-99e3-4395-adeb-da3f7d44835a
```

CIO_IGNORE

Requires the dracut *cio_ignore* module.

rd.cio_accept=<device-ids>

Remove the devices listed in `<device-ids>` from the default `cio_ignore` kernel command-line settings. `<device-ids>` is a list of comma-separated CCW device ids. The default for this value is taken from the `/boot/zipl/active_devices.txt` file.

Example

```
rd.cio_accept=0.0.0180,0.0.0800,0.0.0801,0.0.0802
```

Plymouth Boot Splash

Requires the dracut *plymouth* module.

plymouth.enable=0

disable the plymouth bootsplash completely.

rd.plymouth=0

disable the plymouth bootsplash only for the initramfs.

Kernel keys**masterkey=<kernel master key path name>**

Set the path name of the kernel master key.

Example

```
masterkey=/etc/keys/kmk-trusted.blob
```

masterkeytype=<kernel master key type>

Set the type of the kernel master key.

Example

```
masterkeytype=trusted
```

evmkey=<EVM HMAC key path name>

Set the path name of the EVM HMAC key.

Example

```
evmkey=/etc/keys/evm-trusted.blob
```

evmx509=<EVM X.509 cert path name>

Set the path name of the EVM X.509 certificate.

Example

```
evmx509=/etc/keys/x509_evm.der
```

ecryptfskey=<eCryptfs key path name>

Set the path name of the eCryptfs key.

Example

```
ecryptfskey=/etc/keys/ecryptfs-trusted.blob
```

Deprecated, renamed Options

Here is a list of options and their new replacement.

rd_NO_DM

rd.dm=0 (since version 008)

rd_NO_MDADMCONF

rd.md.conf=0 (since version 008)

rd_NO_MDIMSM

rd.md.ism=0 (since version 008)

rd_NO_MD

rd.md=0 (since version 008)

rd_MD_UUID
rd.md.uuid (since version 008)

iscsi_initiator
rd.iscsi.initiator (since version 008)

iscsi_target_name
rd.iscsi.target.name (since version 008)

iscsi_target_ip
rd.iscsi.target.ip (since version 008)

iscsi_target_port
rd.iscsi.target.port (since version 008)

iscsi_target_group
rd.iscsi.target.group (since version 008)

iscsi_username
rd.iscsi.username (since version 008)

iscsi_password
rd.iscsi.password (since version 008)

iscsi_in_username
rd.iscsi.in.username (since version 008)

iscsi_in_password
rd.iscsi.in.password (since version 008)

iscsi_firmware
rd.iscsi.firmware=0 (since version 008)

rdudevdebug
rd.udev.udev_log=debug (since version 102)

rdudevinfo
rd.udev.udev_log=info (since version 102)

rd.udev.debug
rd.udev.udev_log=debug (since version 102)

rd.udev.info
rd.udev.udev_log=info (since version 102)

KEYMAP
vconsole.keymap (since version 008)

KEYTABLE
vconsole.keymap (since version 008)

SYSFONT
vconsole.font (since version 008)

CONTRANS

vconsole.font.map (since version 008)

UNIMAP

vconsole.font.unimap (since version 008)

UNICODE

vconsole.unicode (since version 008)

EXT_KEYMAP

vconsole.keymap.ext (since version 008)

rd.live.overlay.overlayfs

rd.overlay (since version 110)

rd.live.overlay.readonly

rd.overlay.readonly (since version 110)

rd.live.overlay

rd.overlay (since version 110)

rd.live.overlay.reset

rd.overlay.reset (since version 110)

Configuration in the Initramfs

/etc/conf.d/

Any files found in */etc/conf.d/* will be sourced in the initramfs to set initial values. Command line options will override these values set in the configuration files.

/etc/cmdline

Can contain additional command line options. Deprecated, better use */etc/cmdline.d/*.conf*.

/etc/cmdline.d/.conf*

Can contain additional command line options.

AUTHOR

Harald Hoyer

SEE ALSO

man:dracut[8] man:dracut.conf[5]