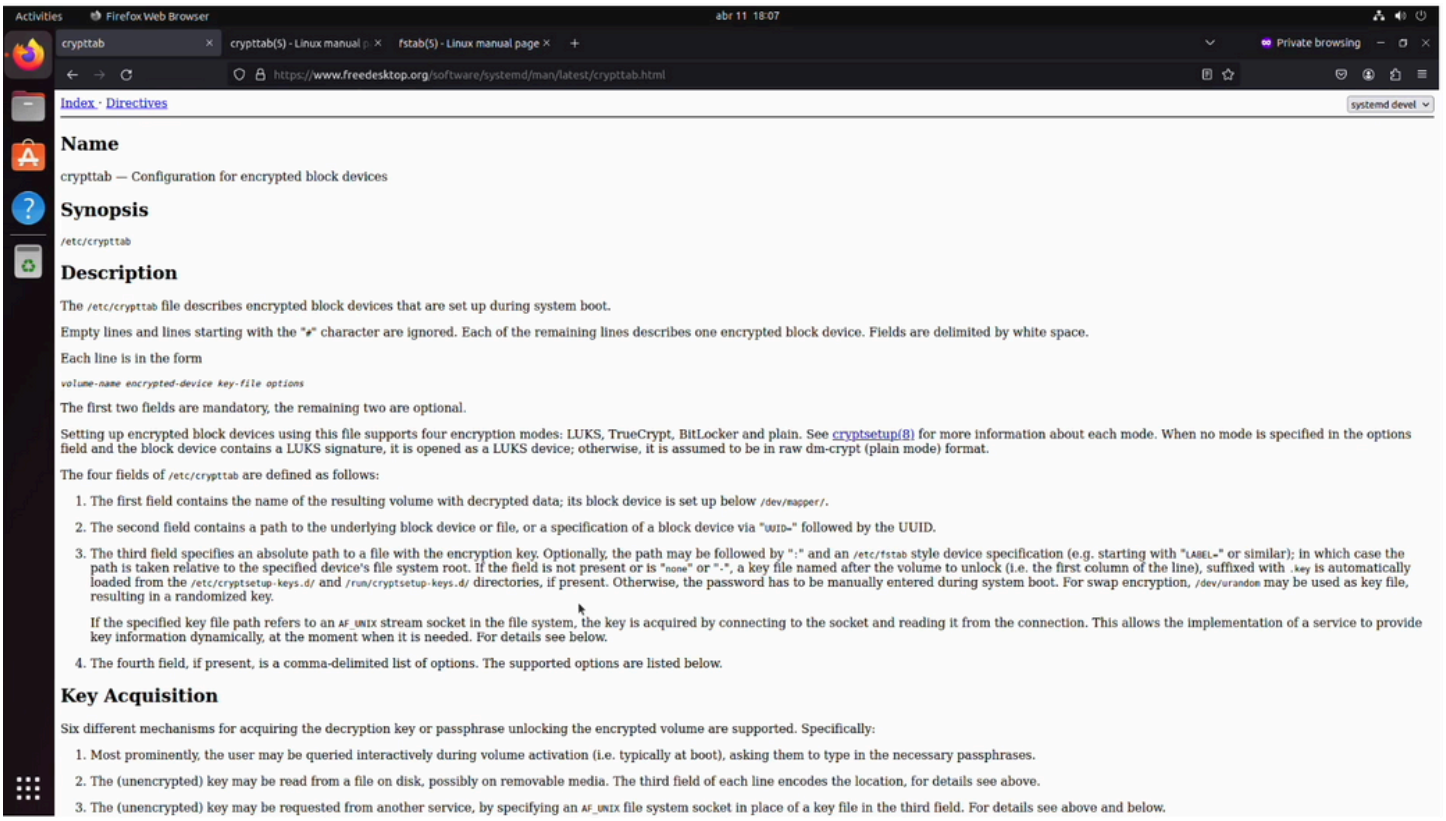
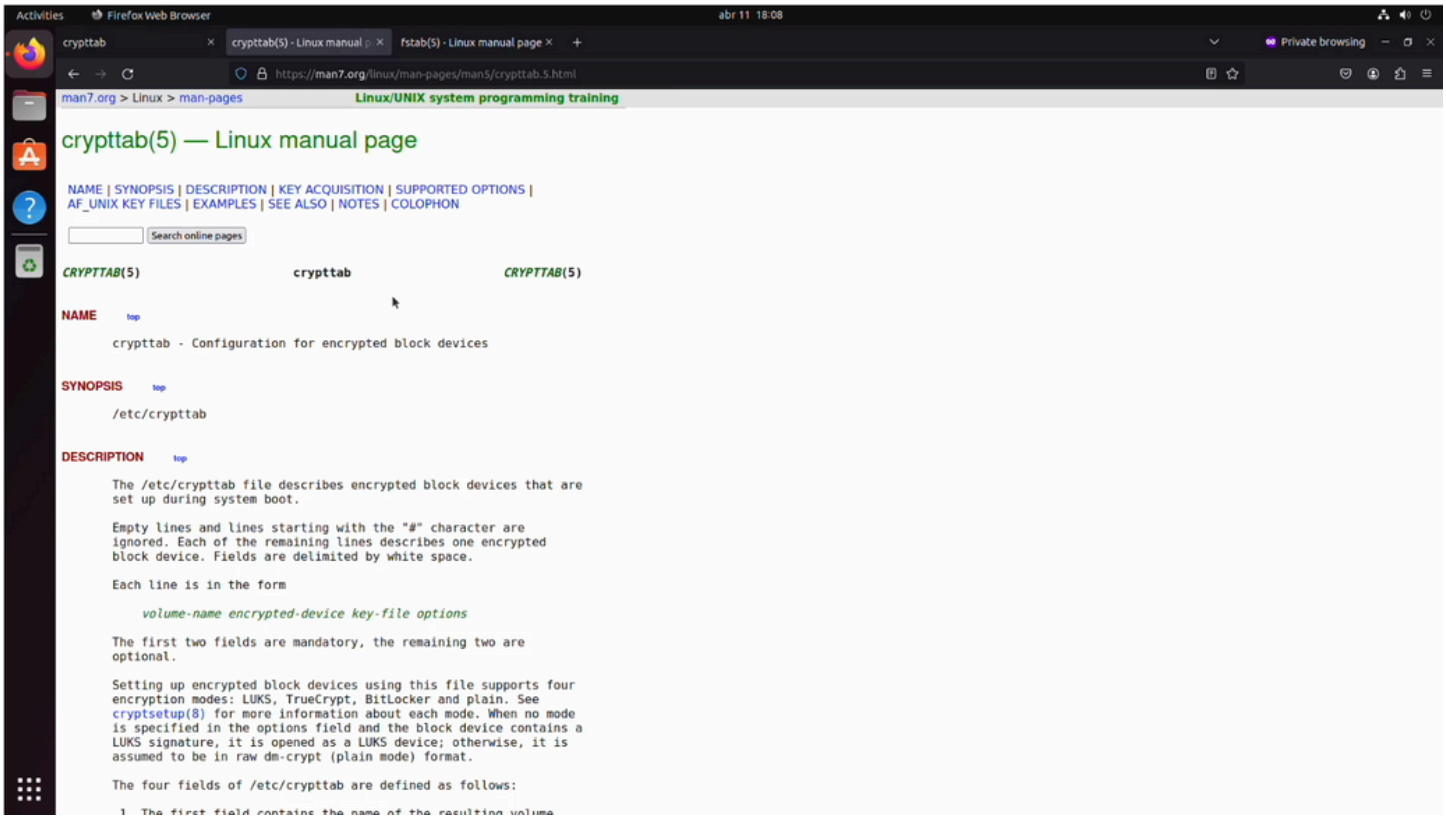
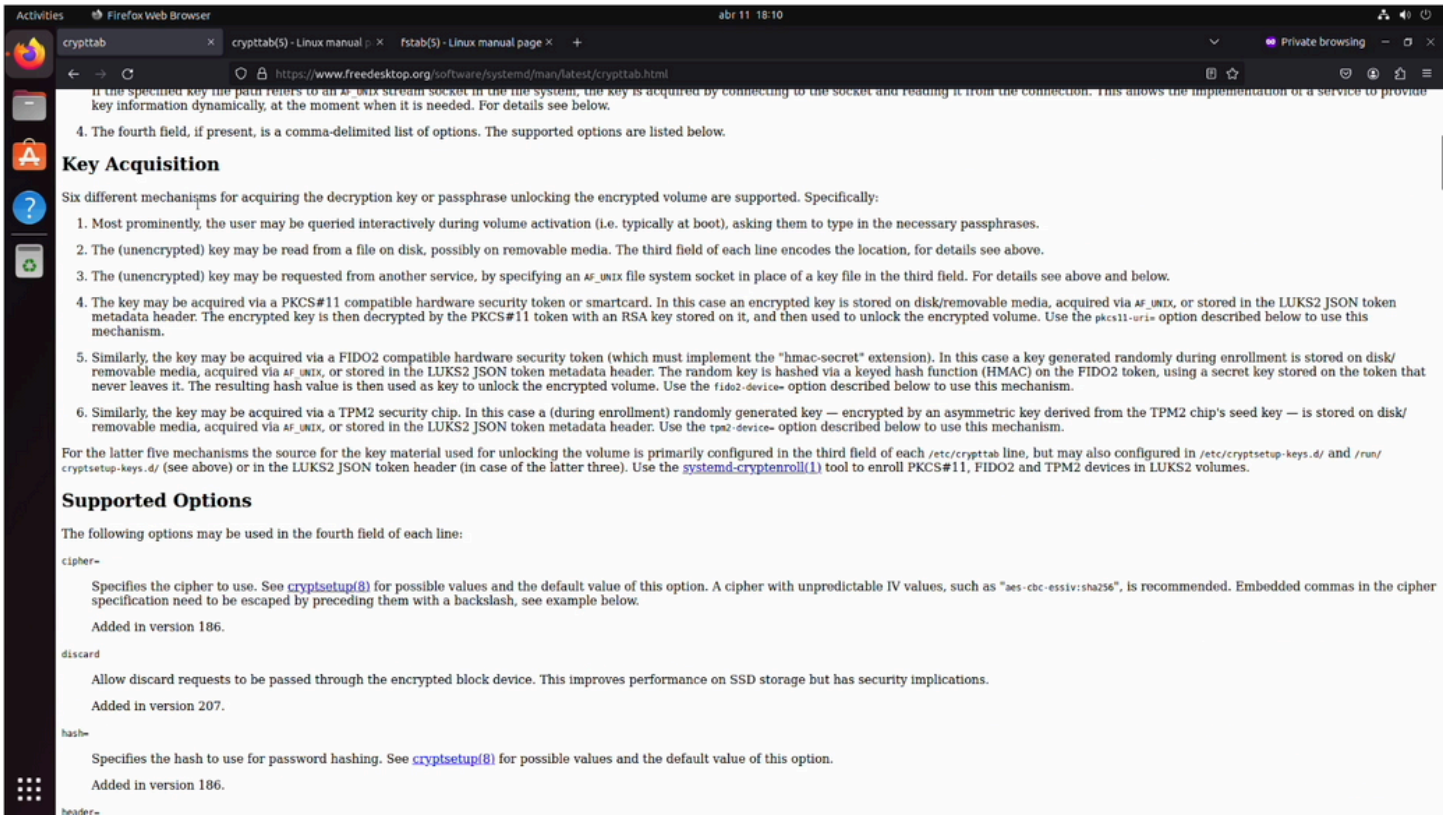

Activities
Firefox Web Browser
abr 11 18:07
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
https://www.freedesktop.org/software/systemd/man/latest/crypttab.html
Index · Directives
systemd devel
Name
crypttab — Configuration for encrypted block devices
Synopsis
/etc/crypttab
Description
The /etc/crypttab file describes encrypted block devices that are set up during system boot.
Empty lines and lines starting with the "#" character are ignored. Each of the remaining lines describes one encrypted block device. Fields are delimited by white space.
Each line is in the form
volume-name encrypted-device key-file options
The first two fields are mandatory, the remaining two are optional.
Setting up encrypted block devices using this file supports four encryption modes: LUKS, TrueCrypt, BitLocker and plain. See cryptsetup(8) for more information about each mode. When no mode is specified in the options field and the block device contains a LUKS signature, it is opened as a LUKS device; otherwise, it is assumed to be in raw dm-crypt (plain mode) format.
The four fields of /etc/crypttab are defined as follows:
1. The first field contains the name of the resulting volume with decrypted data; its block device is set up below /dev/mapper/.
2. The second field contains a path to the underlying block device or file, or a specification of a block device via "UUID=" followed by the UUID.
3. The third field specifies an absolute path to a file with the encryption key. Optionally, the path may be followed by ":" and an /etc/fstab style device specification (e.g. starting with "LABEL=" or similar); in which case the path is taken relative to the specified device's file system root. If the field is not present or is "none" or "-", a key file named after the volume to unlock (i.e. the first column of the line), suffixed with .key is automatically loaded from the /etc/cryptsetup-keys.d/ and /run/cryptsetup-keys.d/ directories, if present. Otherwise, the password has to be manually entered during system boot. For swap encryption, /dev/urandom may be used as key file, resulting in a randomized key.
If the specified key file path refers to an AF_UNIX stream socket in the file system, the key is acquired by connecting to the socket and reading it from the connection. This allows the implementation of a service to provide key information dynamically, at the moment when it is needed. For details see below.
4. The fourth field, if present, is a comma-delimited list of options. The supported options are listed below.
Key Acquisition
Six different mechanisms for acquiring the decryption key or passphrase unlocking the encrypted volume are supported. Specifically:
1. Most prominently, the user may be queried interactively during volume activation (i.e. typically at boot), asking them to type in the necessary passphrases.
2. The (unencrypted) key may be read from a file on disk, possibly on removable media. The third field of each line encodes the location, for details see above.
3. The (unencrypted) key may be requested from another service, by specifying an AF_UNIX file system socket in place of a key file in the third field. For details see above and below.


Activities
Firefox Web Browser
abr 11 18:08
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
https://man7.org/linux/man-pages/man5/crypttab.5.html
man7.org > Linux > man-pages
Linux/UNIX system programming training
crypttab(5) — Linux manual page
NAME | SYNOPSIS | DESCRIPTION | KEY ACQUISITION | SUPPORTED OPTIONS | AF_UNIX KEY FILES | EXAMPLES | SEE ALSO | NOTES | COLOPHON
Search online pages
CRYPTTAB(5) crypttab CRYPTTAB(5)
NAME top
crypttab - Configuration for encrypted block devices
SYNOPSIS top
/etc/crypttab
DESCRIPTION top
The /etc/crypttab file describes encrypted block devices that are set up during system boot.
Empty lines and lines starting with the "#" character are ignored. Each of the remaining lines describes one encrypted block device. Fields are delimited by white space.
Each line is in the form
volume-name encrypted-device key-file options
The first two fields are mandatory, the remaining two are optional.
Setting up encrypted block devices using this file supports four encryption modes: LUKS, TrueCrypt, BitLocker and plain. See cryptsetup(8) for more information about each mode. When no mode is specified in the options field and the block device contains a LUKS signature, it is opened as a LUKS device; otherwise, it is assumed to be in raw dm-crypt (plain mode) format.
The four fields of /etc/crypttab are defined as follows:



If the specified key file path refers to an `AF_UNIX` stream socket in the file system, the key is acquired by connecting to the socket and reading it from the connection. This allows the implementation of a service to provide key information dynamically, at the moment when it is needed. For details see below.

4. The fourth field, if present, is a comma-delimited list of options. The supported options are listed below.

Key Acquisition

Six different mechanisms for acquiring the decryption key or passphrase unlocking the encrypted volume are supported. Specifically:

1. Most prominently, the user may be queried interactively during volume activation (i.e. typically at boot), asking them to type in the necessary passphrases.
2. The (unencrypted) key may be read from a file on disk, possibly on removable media. The third field of each line encodes the location, for details see above.
3. The (unencrypted) key may be requested from another service, by specifying an `AF_UNIX` file system socket in place of a key file in the third field. For details see above and below.
4. The key may be acquired via a PKCS#11 compatible hardware security token or smartcard. In this case an encrypted key is stored on disk/removable media, acquired via `AF_UNIX`, or stored in the LUKS2 JSON token metadata header. The encrypted key is then decrypted by the PKCS#11 token with an RSA key stored on it, and then used to unlock the encrypted volume. Use the `pkcs11-uri=` option described below to use this mechanism.
5. Similarly, the key may be acquired via a FIDO2 compatible hardware security token (which must implement the "hmac-secret" extension). In this case a key generated randomly during enrollment is stored on disk/removable media, acquired via `AF_UNIX`, or stored in the LUKS2 JSON token metadata header. The random key is hashed via a keyed hash function (HMAC) on the FIDO2 token, using a secret key stored on the token that never leaves it. The resulting hash value is then used as key to unlock the encrypted volume. Use the `fido2-device=` option described below to use this mechanism.
6. Similarly, the key may be acquired via a TPM2 security chip. In this case a (during enrollment) randomly generated key — encrypted by an asymmetric key derived from the TPM2 chip's seed key — is stored on disk/removable media, acquired via `AF_UNIX`, or stored in the LUKS2 JSON token metadata header. Use the `tpm2-device=` option described below to use this mechanism.

For the latter five mechanisms the source for the key material used for unlocking the volume is primarily configured in the third field of each `/etc/crypttab` line, but may also configured in `/etc/cryptsetup-keys.d/` and `/run/cryptsetup-keys.d/` (see above) or in the LUKS2 JSON token header (in case of the latter three). Use the systemd-cryptenroll(1) tool to enroll PKCS#11, FIDO2 and TPM2 devices in LUKS2 volumes.

Supported Options

The following options may be used in the fourth field of each line:

`cipher=`

Specifies the cipher to use. See cryptsetup(8) for possible values and the default value of this option. A cipher with unpredictable IV values, such as "`aes-cbc-essiv:sha256`", is recommended. Embedded commas in the cipher specification need to be escaped by preceding them with a backslash, see example below.

Added in version 186.

`discard`

Allow discard requests to be passed through the encrypted block device. This improves performance on SSD storage but has security implications.

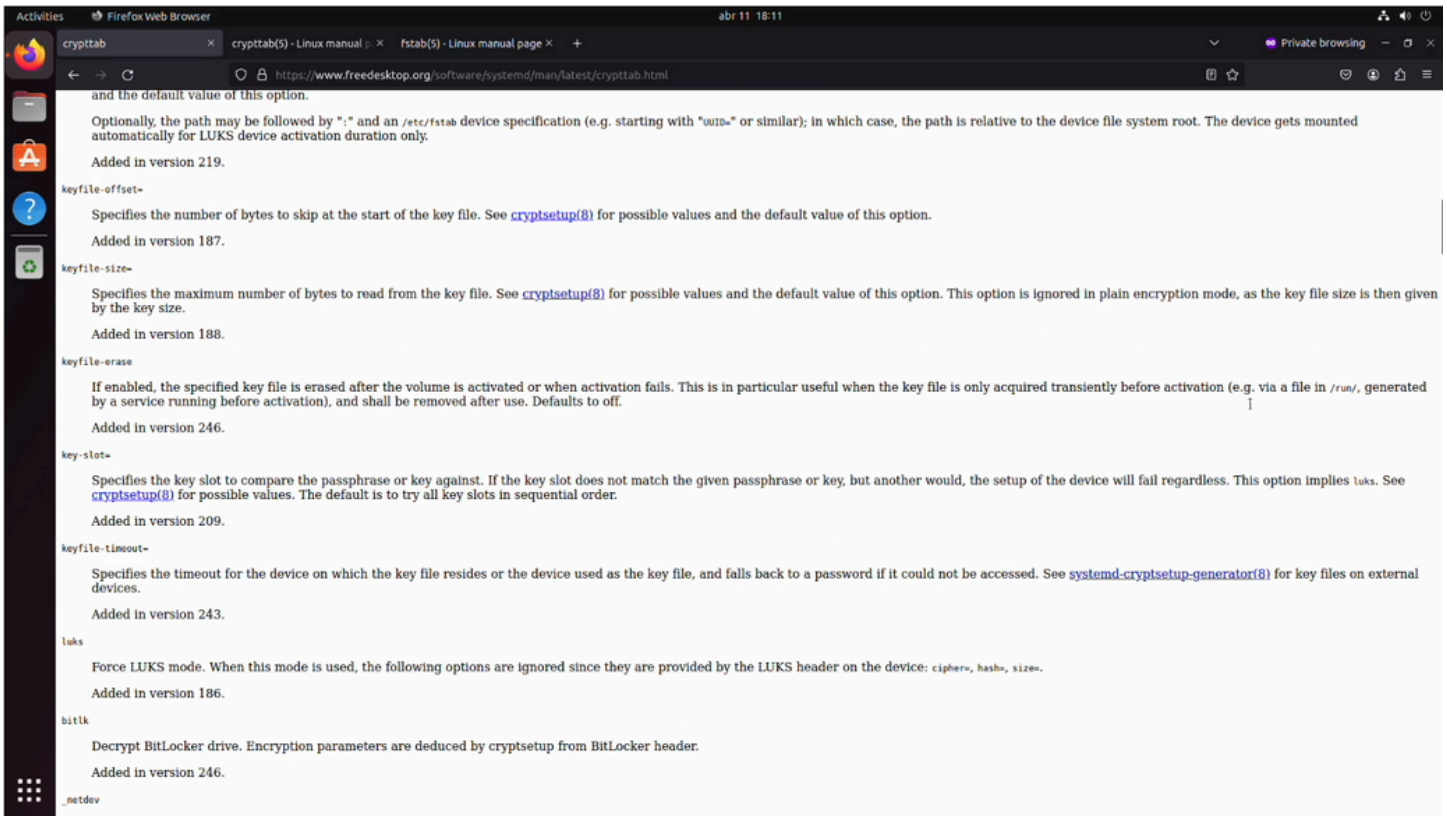
Added in version 207.

`hash=`

Specifies the hash to use for password hashing. See cryptsetup(8) for possible values and the default value of this option.

Added in version 186.

`header=`



and the default value of this option.

Optionally, the path may be followed by "`:`" and an `/etc/fstab` device specification (e.g. starting with "`UUID=`" or similar); in which case, the path is relative to the device file system root. The device gets mounted automatically for LUKS device activation duration only.

Added in version 219.

`keyfile-offset=`

Specifies the number of bytes to skip at the start of the key file. See cryptsetup(8) for possible values and the default value of this option.

Added in version 187.

`keyfile-size=`

Specifies the maximum number of bytes to read from the key file. See cryptsetup(8) for possible values and the default value of this option. This option is ignored in plain encryption mode, as the key file size is then given by the key size.

Added in version 188.

`keyfile-erase`

If enabled, the specified key file is erased after the volume is activated or when activation fails. This is in particular useful when the key file is only acquired transiently before activation (e.g. via a file in `/run/`, generated by a service running before activation), and shall be removed after use. Defaults to off.

Added in version 246.

`key-slot=`

Specifies the key slot to compare the passphrase or key against. If the key slot does not match the given passphrase or key, but another would, the setup of the device will fail regardless. This option implies `luks`. See cryptsetup(8) for possible values. The default is to try all key slots in sequential order.

Added in version 209.

`keyfile-timeout=`

Specifies the timeout for the device on which the key file resides or the device used as the key file, and falls back to a password if it could not be accessed. See systemd-cryptsetup-generator(8) for key files on external devices.

Added in version 243.

`luks`

Force LUKS mode. When this mode is used, the following options are ignored since they are provided by the LUKS header on the device: `cipher=`, `hash=`, `size=`.

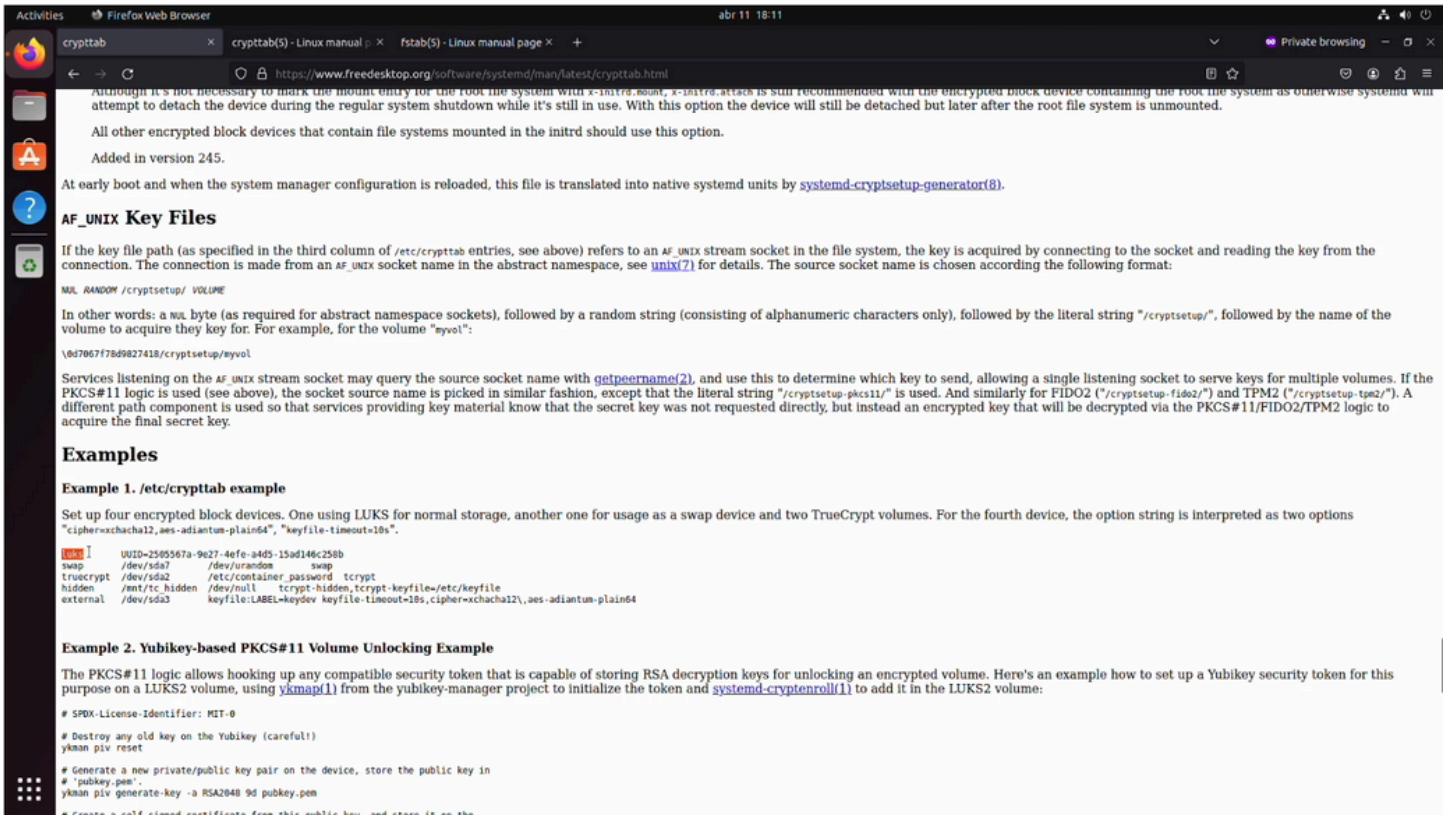
Added in version 186.

`bitlk`

Decrypt BitLocker drive. Encryption parameters are deduced by cryptsetup from BitLocker header.

Added in version 246.

`_netdev`



Although it's not necessary to mark the mount entry for the root file system with `x-initrd.mount`, `x-initrd.attach` is still recommended with the encrypted block device containing the root file system as otherwise systemd will attempt to detach the device during the regular system shutdown while it's still in use. With this option the device will still be detached but later after the root file system is unmounted.

All other encrypted block devices that contain file systems mounted in the initrd should use this option.

Added in version 245.

At early boot and when the system manager configuration is reloaded, this file is translated into native systemd units by systemd-cryptsetup-generator(8).

`AF_UNIX` Key Files

If the key file path (as specified in the third column of `/etc/crypttab` entries, see above) refers to an `AF_UNIX` stream socket in the file system, the key is acquired by connecting to the socket and reading the key from the connection. The connection is made from an `AF_UNIX` socket name in the abstract namespace, see unix(7) for details. The source socket name is chosen according the following format:

```
NUL RANDOM /cryptsetup/ VOLUME
```

In other words: a `NUL` byte (as required for abstract namespace sockets), followed by a random string (consisting of alphanumeric characters only), followed by the literal string "`/cryptsetup/`", followed by the name of the volume to acquire they key for. For example, for the volume "`myvol`":

```
\0d7067f78d9827418/cryptsetup/myvol
```

Services listening on the `AF_UNIX` stream socket may query the source socket name with getpeername(2), and use this to determine which key to send, allowing a single listening socket to serve keys for multiple volumes. If the PKCS#11 logic is used (see above), the socket source name is picked in similar fashion, except that the literal string "`/cryptsetup-pkcs11/`" is used. And similarly for FIDO2 ("`/cryptsetup-fido2/`") and TPM2 ("`/cryptsetup-tpm2/`"). A different path component is used so that services providing key material know that the secret key was not requested directly, but instead an encrypted key that will be decrypted via the PKCS#11/FIDO2/TPM2 logic to acquire the final secret key.

Examples

Example 1. /etc/crypttab example

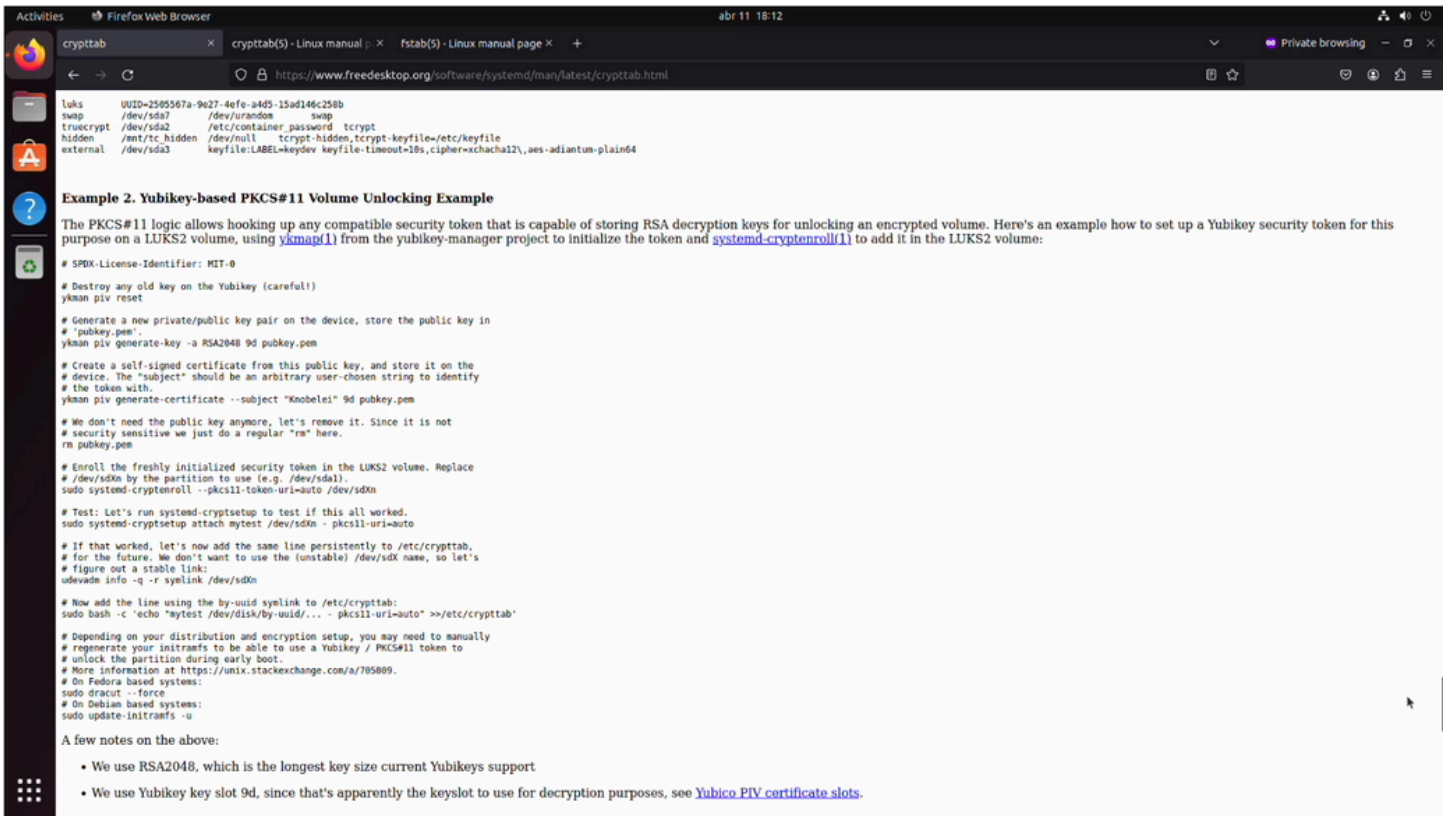
Set up four encrypted block devices. One using LUKS for normal storage, another one for usage as a swap device and two TrueCrypt volumes. For the fourth device, the option string is interpreted as two options "`cipher=xchacha12,aes-adiantum-plain64`", "`keyfile-timeout=10s`".

```
luks      UUID=2505567a-9e27-4efe-a4d5-15ad146c258b
swap      /dev/sda7       /dev/urandom       swap
truecrypt /dev/sda2       /etc/container_password  tcrypt
hidden    /mnt/tc_hidden  /dev/null     tcrypt-hidden,tcrypt-keyfile=/etc/keyfile
external  /dev/sda3       keyfile:LABEL=keydev keyfile-timeout=10s,cipher=xchacha12\,aes-adiantum-plain64
```

Example 2. Yubikey-based PKCS#11 Volume Unlocking Example

The PKCS#11 logic allows hooking up any compatible security token that is capable of storing RSA decryption keys for unlocking an encrypted volume. Here's an example how to set up a Yubikey security token for this purpose on a LUKS2 volume, using ykmap(1) from the yubikey-manager project to initialize the token and systemd-cryptenroll(1) to add it in the LUKS2 volume:

```
# SPDX-License-Identifier: MIT-0

# Destroy any old key on the Yubikey (careful!)
ykman piv reset

# Generate a new private/public key pair on the device, store the public key in
# 'pubkey.pem'.
ykman piv generate-key -a RSA2048 9d pubkey.pem
```




```
luks      UUID=2505567a-9e27-4efe-a4d5-15ad146c258b
swap      /dev/sda7       /dev/urandom       swap
truecrypt /dev/sda2       /etc/container_password  tcrypt
hidden    /mnt/tc_hidden  /dev/null     tcrypt-hidden,tcrypt-keyfile=/etc/keyfile
external  /dev/sda3       keyfile:LABEL=keydev keyfile-timeout=10s,cipher=xchacha12\,aes-adiantum-plain64
```

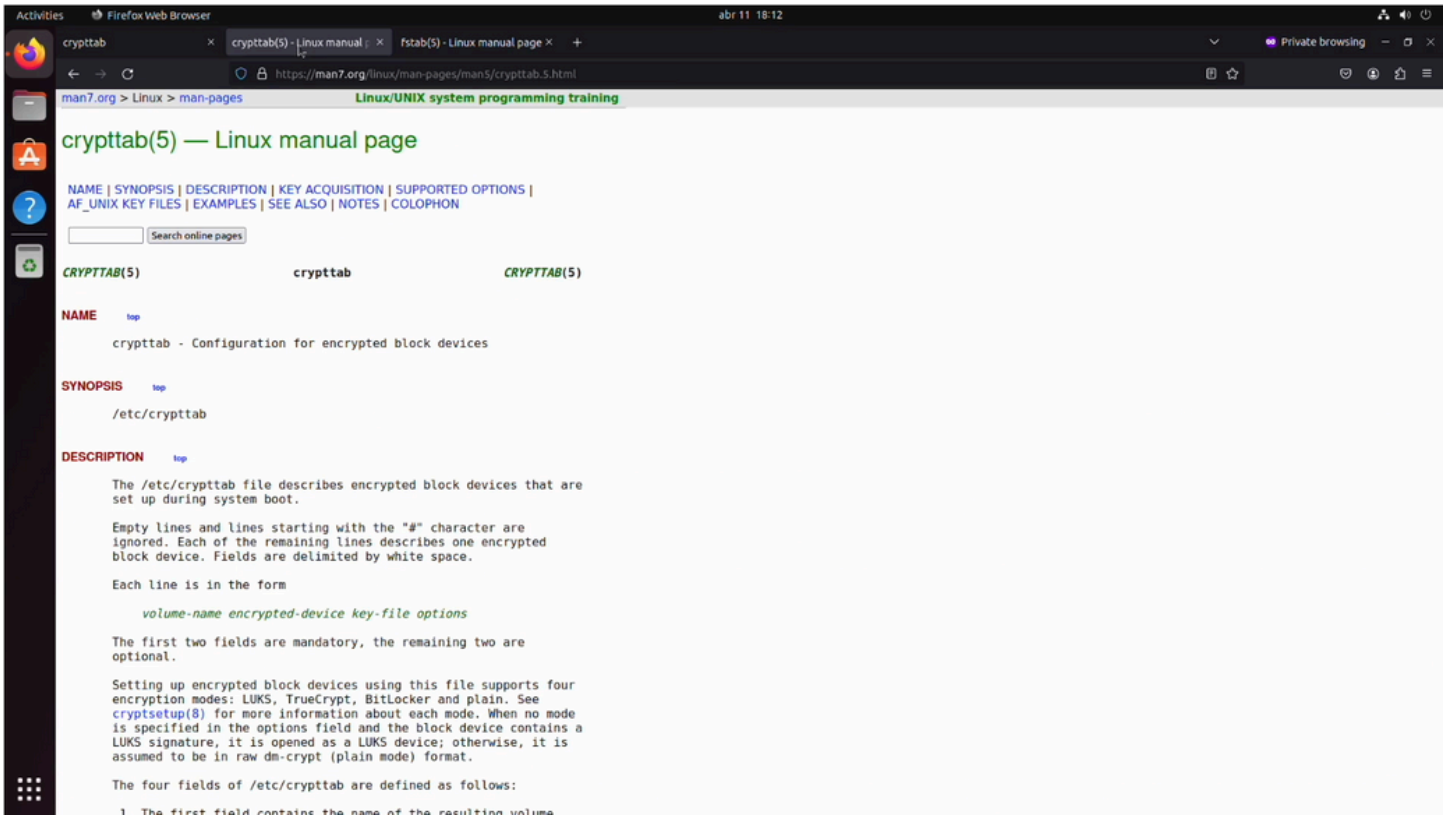
Example 2. Yubikey-based PKCS#11 Volume Unlocking Example

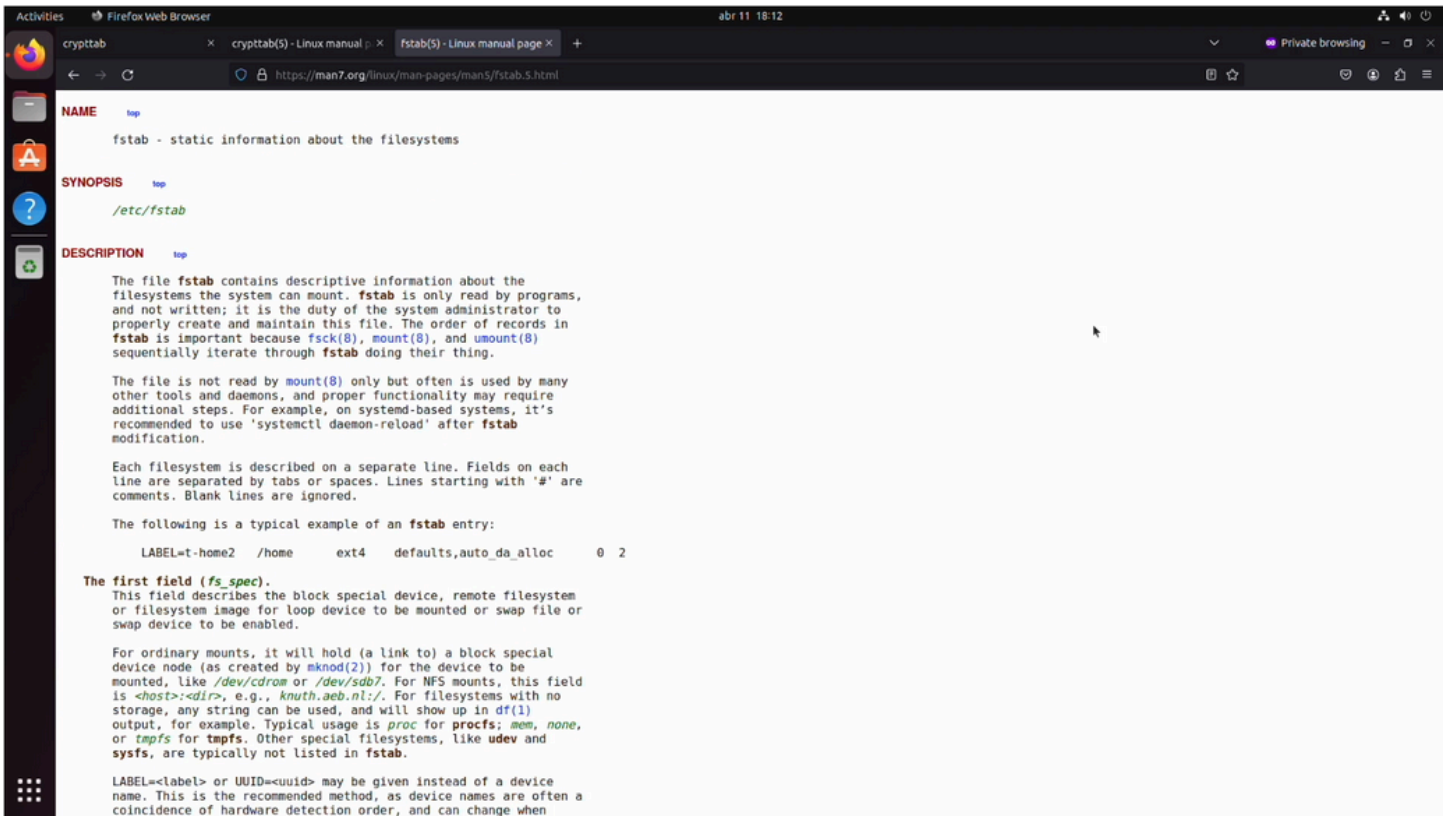
The PKCS#11 logic allows hooking up any compatible security token that is capable of storing RSA decryption keys for unlocking an encrypted volume. Here's an example how to set up a Yubikey security token for this purpose on a LUKS2 volume, using ykmap(1) from the yubikey-manager project to initialize the token and systemd-cryptenroll(1) to add it in the LUKS2 volume:

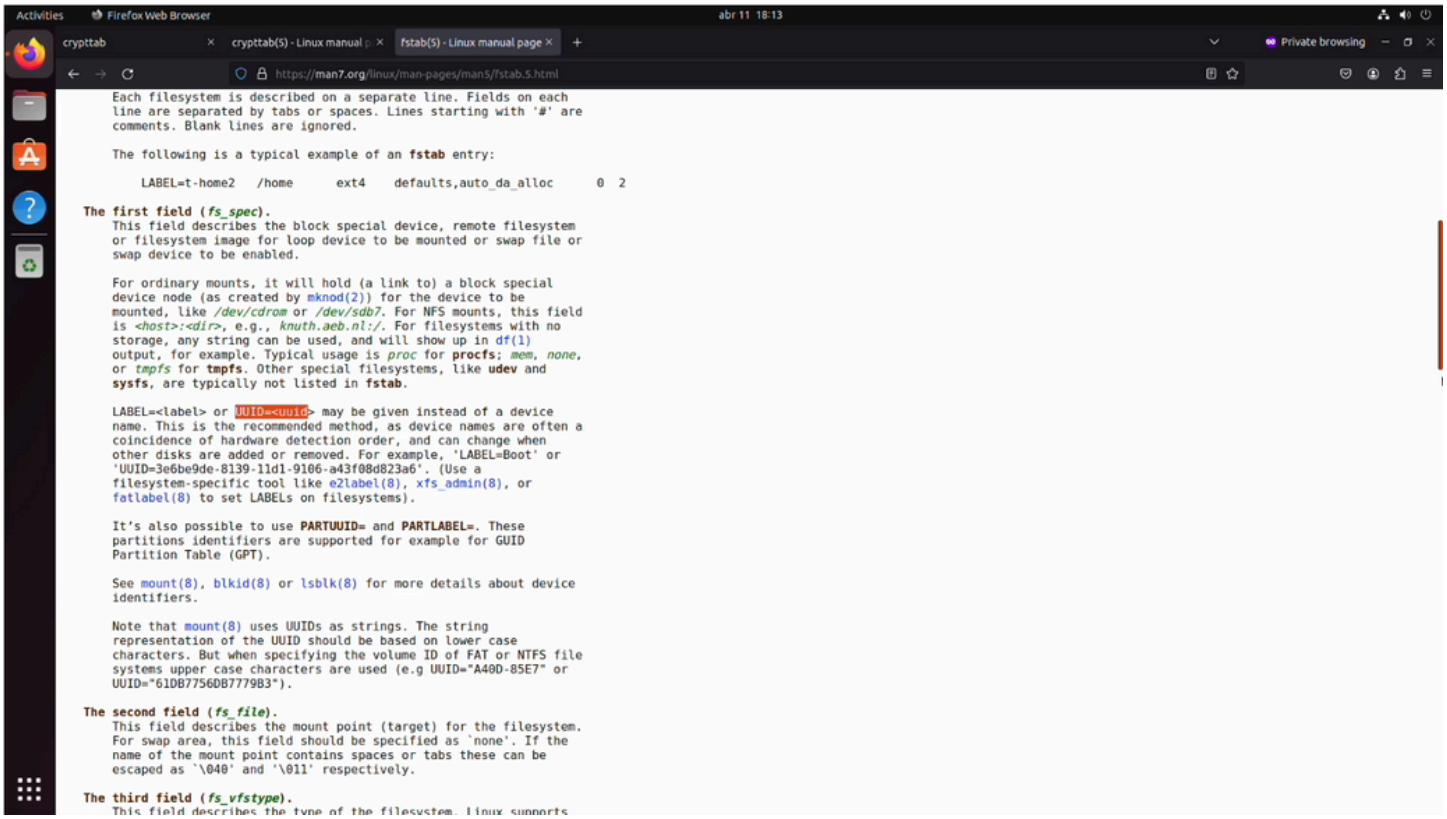
```
# SPDX-License-Identifier: MIT-0

# Destroy any old key on the Yubikey (careful!)
ykman piv reset

# Generate a new private/public key pair on the device, store the public key in
# 'pubkey.pem'.
ykman piv generate-key -a RSA2048 9d pubkey.pem

# Create a self-signed certificate from this public key, and store it on the
# device. The "subject" should be an arbitrary user-chosen string to identify
# the token with.
ykman piv generate-certificate --subject "Knobelei" 9d pubkey.pem

# We don't need the public key anymore, let's remove it. Since it is not
# security sensitive we just do a regular "rm" here.
rm pubkey.pem

# Enroll the freshly initialized security token in the LUKS2 volume. Replace
# /dev/sdXn by the partition to use (e.g. /dev/sda1).
sudo systemd-cryptenroll --pkcs11-token-uri=auto /dev/sdXn

# Test: Let's run systemd-cryptsetup to test if this all worked.
sudo systemd-cryptsetup attach mytest /dev/sdXn - pkcs11-uri=auto

# If that worked, let's now add the same line persistently to /etc/crypttab,
# for the future. We don't want to use the (unstable) /dev/sdX name, so let's
# figure out a stable link:
udevadm info -q -r symlink /dev/sdXn

# Now add the line using the by-uuid symlink to /etc/crypttab:
sudo bash -c 'echo "mytest /dev/disk/by-uuid/... - pkcs11-uri=auto" >>/etc/crypttab'

# Depending on your distribution and encryption setup, you may need to manually
# regenerate your initramfs to be able to use a Yubikey / PKCS#11 token to
# unlock the partition during early boot.
# More information at https://unix.stackexchange.com/a/705809.
# On Fedora based systems:
sudo dracut --force
# On Debian based systems:
sudo update-initramfs -u
```


A few notes on the above:

- We use RSA2048, which is the longest key size current Yubikeys support
- We use Yubikey key slot 9d, since that's apparently the keyslot to use for decryption purposes, see Yubico PIV certificate slots.


Activities
Firefox Web Browser
abr 11 18:12
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
https://man7.org/linux/man-pages/man5/crypttab.5.html
man7.org > Linux > man-pages
Linux/UNIX system programming training
crypttab(5) — Linux manual page
NAME | SYNOPSIS | DESCRIPTION | KEY ACQUISITION | SUPPORTED OPTIONS |
AF_UNIX KEY FILES | EXAMPLES | SEE ALSO | NOTES | COLOPHON
Search online pages
CRYPTTAB(5) crypttab CRYPTTAB(5)
NAME top
crypttab - Configuration for encrypted block devices
SYNOPSIS top
/etc/crypttab
DESCRIPTION top
The /etc/crypttab file describes encrypted block devices that are
set up during system boot.
Empty lines and lines starting with the "#" character are
ignored. Each of the remaining lines describes one encrypted
block device. Fields are delimited by white space.
Each line is in the form
volume-name encrypted-device key-file options
The first two fields are mandatory, the remaining two are
optional.
Setting up encrypted block devices using this file supports four
encryption modes: LUKS, TrueCrypt, BitLocker and plain. See
cryptsetup(8) for more information about each mode. When no mode
is specified in the options field and the block device contains a
LUKS signature, it is opened as a LUKS device; otherwise, it is
assumed to be in raw dm-crypt (plain mode) format.
The four fields of /etc/crypttab are defined as follows:


Activities
Firefox Web Browser
abr 11 18:12
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
https://man7.org/linux/man-pages/man5/fstab.5.html
NAME top
fstab - static information about the filesystems
SYNOPSIS top
/etc/fstab
DESCRIPTION top
The file fstab contains descriptive information about the
filesystems the system can mount. fstab is only read by programs,
and not written; it is the duty of the system administrator to
properly create and maintain this file. The order of records in
fstab is important because fsck(8), mount(8), and umount(8)
sequentially iterate through fstab doing their thing.
The file is not read by mount(8) only but often is used by many
other tools and daemons, and proper functionality may require
additional steps. For example, on systemd-based systems, it's
recommended to use 'systemctl daemon-reload' after fstab
modification.
Each filesystem is described on a separate line. Fields on each
line are separated by tabs or spaces. Lines starting with '#' are
comments. Blank lines are ignored.
The following is a typical example of an fstab entry:
LABEL=t-home2 /home ext4 defaults,auto_da_alloc 0 2
The first field (fs_spec).
This field describes the block special device, remote filesystem
or filesystem image for loop device to be mounted or swap file or
swap device to be enabled.
For ordinary mounts, it will hold (a link to) a block special
device node (as created by mknod(2)) for the device to be
mounted, like /dev/cdrom or /dev/sdb7. For NFS mounts, this field
is <host>:<dir>, e.g., knuth.aeb.nl:/. For filesystems with no
storage, any string can be used, and will show up in df(1)
output, for example. Typical usage is proc for procfs; mem, none,
or tmpfs for tmpfs. Other special filesystems, like udev and
sysfs, are typically not listed in fstab.
LABEL=<label> or UUID=<uuid> may be given instead of a device
name. This is the recommended method, as device names are often a
coincidence of hardware detection order, and can change when



Each filesystem is described on a separate line. Fields on each line are separated by tabs or spaces. Lines starting with '#' are comments. Blank lines are ignored.

The following is a typical example of an **fstab** entry:

```
    LABEL=t-home2   /home      ext4    defaults,auto_da_alloc      0  2
```

The first field (*fs_spec*).
This field describes the block special device, remote filesystem or filesystem image for loop device to be mounted or swap file or swap device to be enabled.

For ordinary mounts, it will hold (a link to) a block special device node (as created by mknod(2)) for the device to be mounted, like */dev/cdrom* or */dev/sdb7*. For NFS mounts, this field is *<host>:<dir>*, e.g., *knuth.aeb.nl:/*. For filesystems with no storage, any string can be used, and will show up in df(1) output, for example. Typical usage is *proc* for **procfs**; *mem*, *none*, or *tmpfs* for **tmpfs**. Other special filesystems, like **udev** and **sysfs**, are typically not listed in **fstab**.

LABEL=<label> or UUID=<uuid> may be given instead of a device name. This is the recommended method, as device names are often a coincidence of hardware detection order, and can change when other disks are added or removed. For example, 'LABEL=Boot' or 'UUID=3e6be9de-8139-11d1-9106-a43f08d823a6'. (Use a filesystem-specific tool like e2label(8), xfs_admin(8), or fatlabel(8) to set LABELs on filesystems).

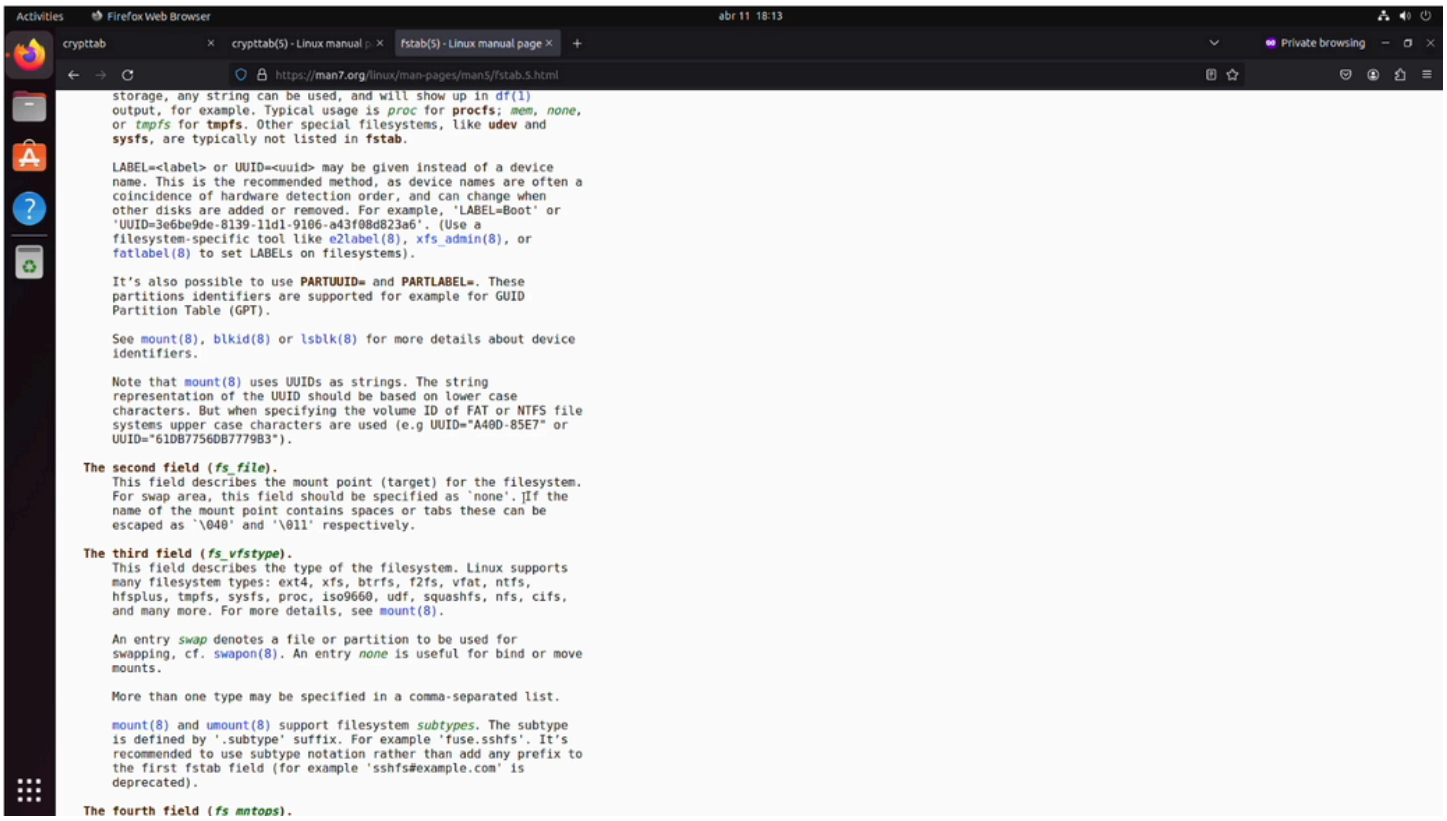
It's also possible to use **PARTUUID=** and **PARTLABEL=**. These partitions identifiers are supported for example for GUID Partition Table (GPT).

See mount(8), blkid(8) or lsblk(8) for more details about device identifiers.

Note that mount(8) uses UUIDs as strings. The string representation of the UUID should be based on lower case characters. But when specifying the volume ID of FAT or NTFS file systems upper case characters are used (e.g UUID="A40D-85E7" or UUID="61DB7756DB7779B3").

The second field (*fs_file*).
This field describes the mount point (target) for the filesystem. For swap area, this field should be specified as `none'. If the name of the mount point contains spaces or tabs these can be escaped as `\040' and '\011' respectively.

The third field (*fs_vfstype*).
This field describes the type of the filesystem. Linux supports



storage, any string can be used, and will show up in df(1) output, for example. Typical usage is *proc* for **procfs**; *mem*, *none*, or *tmpfs* for **tmpfs**. Other special filesystems, like **udev** and **sysfs**, are typically not listed in **fstab**.

LABEL=<label> or UUID=<uuid> may be given instead of a device name. This is the recommended method, as device names are often a coincidence of hardware detection order, and can change when other disks are added or removed. For example, 'LABEL=Boot' or 'UUID=3e6be9de-8139-11d1-9106-a43f08d823a6'. (Use a filesystem-specific tool like e2label(8), xfs_admin(8), or fatlabel(8) to set LABELs on filesystems).

It's also possible to use **PARTUUID=** and **PARTLABEL=**. These partitions identifiers are supported for example for GUID Partition Table (GPT).

See mount(8), blkid(8) or lsblk(8) for more details about device identifiers.

Note that mount(8) uses UUIDs as strings. The string representation of the UUID should be based on lower case characters. But when specifying the volume ID of FAT or NTFS file systems upper case characters are used (e.g UUID="A40D-85E7" or UUID="61DB7756DB7779B3").

The second field (*fs_file*).
This field describes the mount point (target) for the filesystem. For swap area, this field should be specified as `none'. If the name of the mount point contains spaces or tabs these can be escaped as `\040' and '\011' respectively.

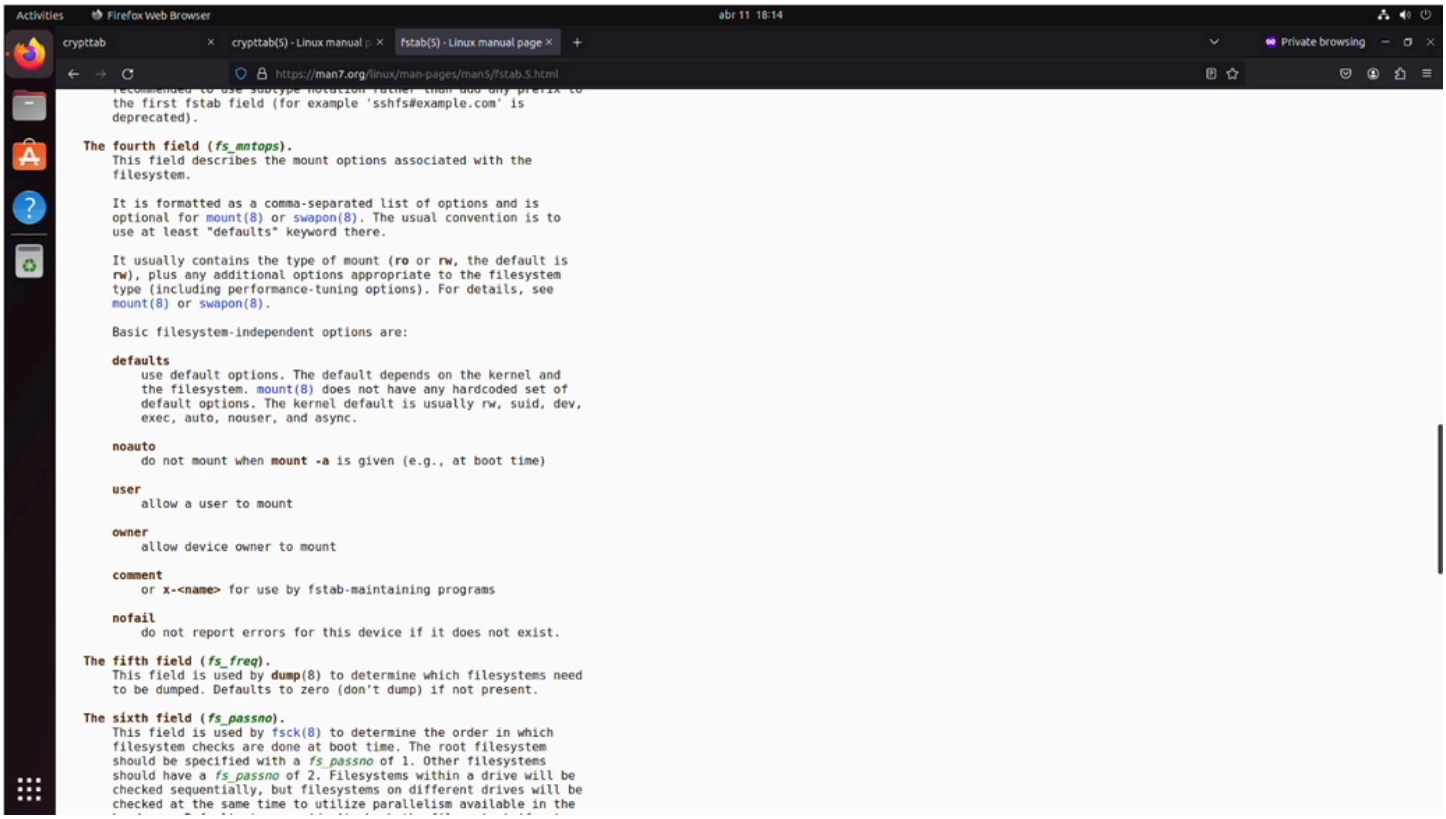
The third field (*fs_vfstype*).
This field describes the type of the filesystem. Linux supports many filesystem types: ext4, xfs, btrfs, f2fs, vfat, ntfs, hfsplus, tmpfs, sysfs, proc, iso9660, udf, squashfs, nfs, cifs, and many more. For more details, see mount(8).

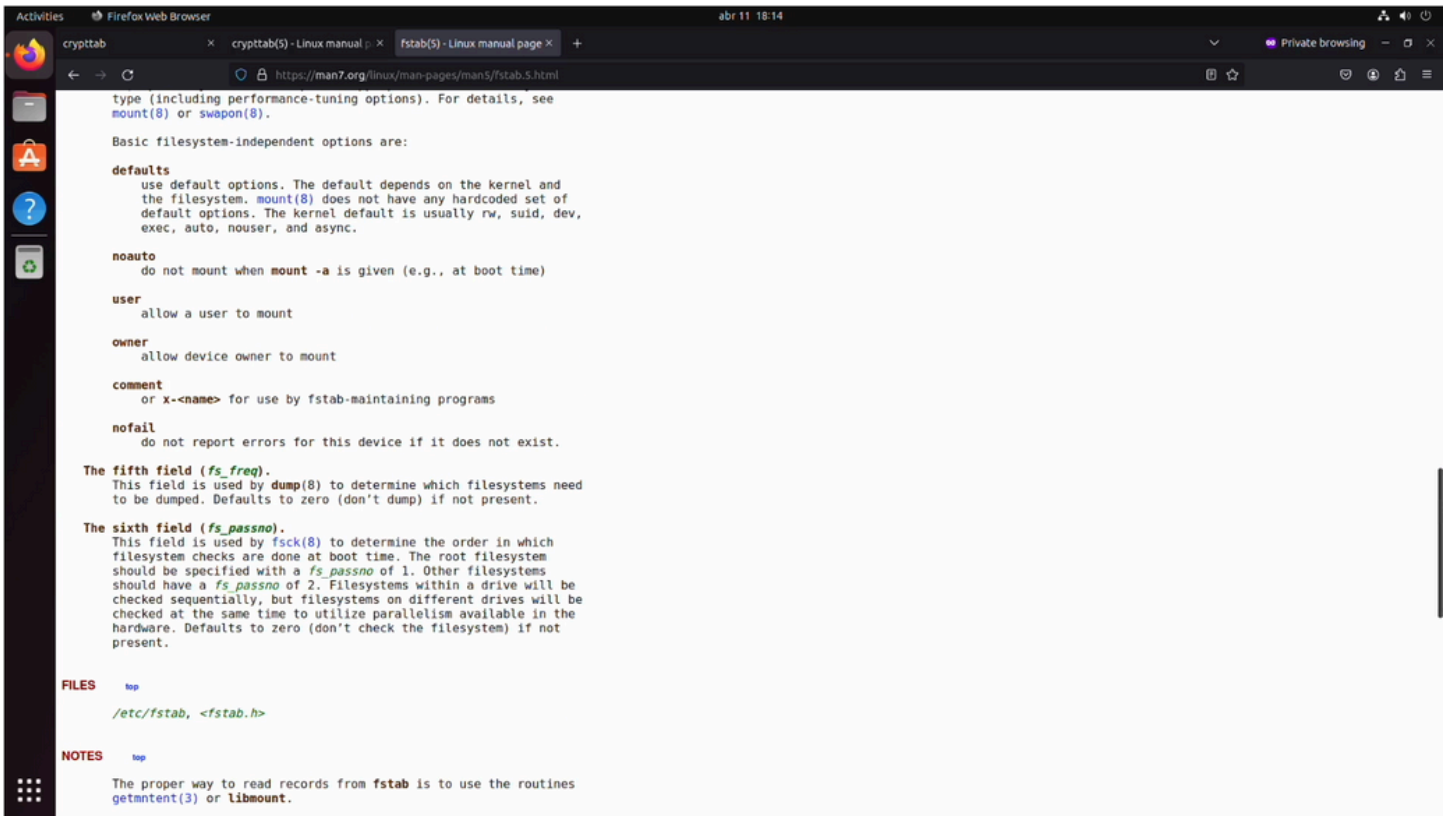
An entry *swap* denotes a file or partition to be used for swapping, cf. swapon(8). An entry *none* is useful for bind or move mounts.

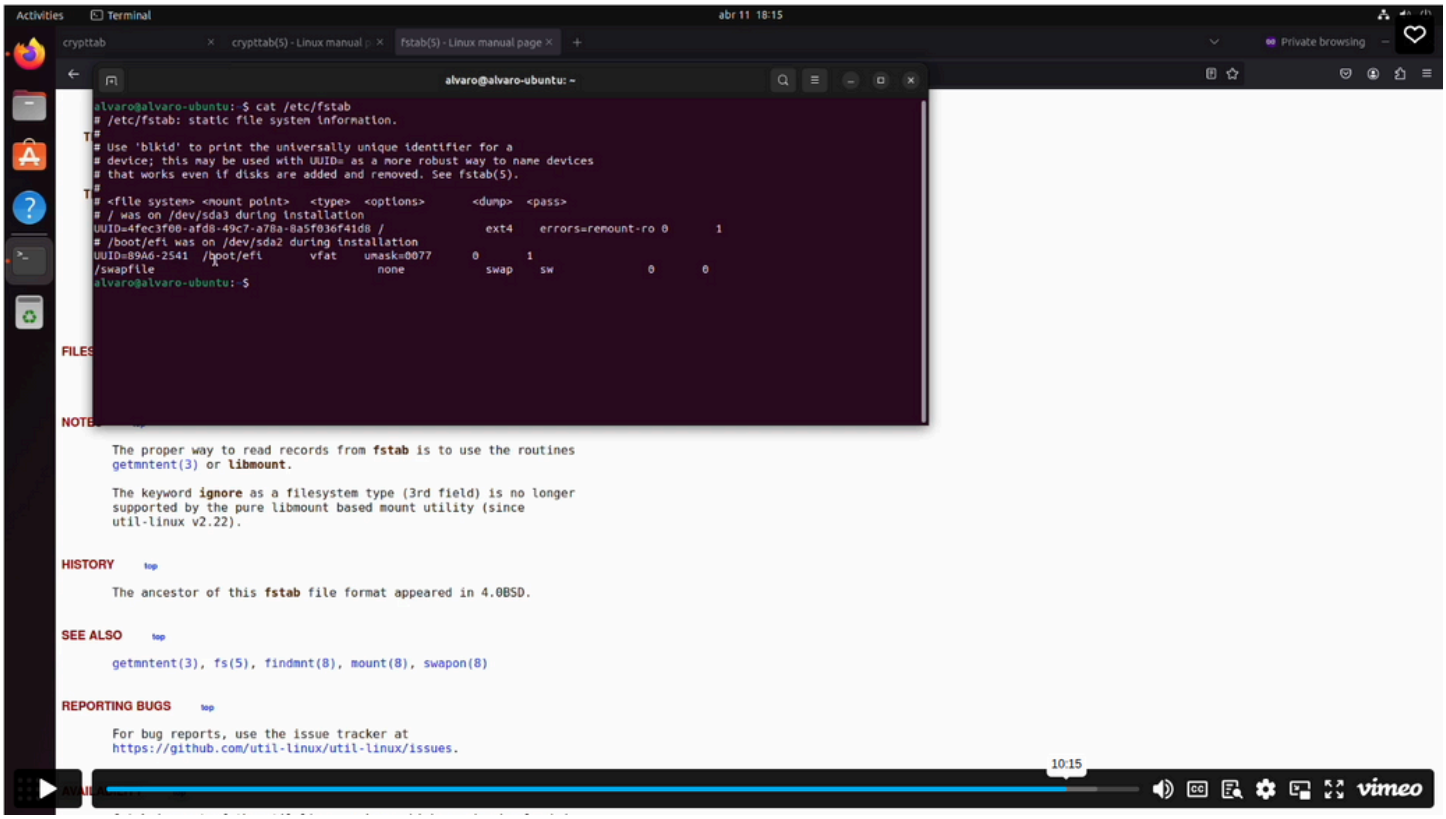
More than one type may be specified in a comma-separated list.

mount(8) and umount(8) support filesystem *subtypes*. The subtype is defined by '.subtype' suffix. For example 'fuse.sshfs'. It's recommended to use subtype notation rather than add any prefix to the first fstab field (for example 'sshfs#example.com' is deprecated).

The fourth field (*fs_mntops*).


Activities
Firefox Web Browser
abr 11 18:14
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
https://man7.org/linux/man-pages/man5/fstab.5.html
the first fstab field (for example 'sshfs#example.com' is
deprecated).
The fourth field (fs_mntops).
This field describes the mount options associated with the
filesystem.
It is formatted as a comma-separated list of options and is
optional for mount(8) or swapon(8). The usual convention is to
use at least "defaults" keyword there.
It usually contains the type of mount (ro or rw, the default is
rw), plus any additional options appropriate to the filesystem
type (including performance-tuning options). For details, see
mount(8) or swapon(8).
Basic filesystem-independent options are:
defaults
use default options. The default depends on the kernel and
the filesystem. mount(8) does not have any hardcoded set of
default options. The kernel default is usually rw, suid, dev,
exec, auto, nouser, and async.
noauto
do not mount when mount -a is given (e.g., at boot time)
user
allow a user to mount
owner
allow device owner to mount
comment
or x-<name> for use by fstab-maintaining programs
nofail
do not report errors for this device if it does not exist.
The fifth field (fs_freq).
This field is used by dump(8) to determine which filesystems need
to be dumped. Defaults to zero (don't dump) if not present.
The sixth field (fs_passno).
This field is used by fsck(8) to determine the order in which
filesystem checks are done at boot time. The root filesystem
should be specified with a fs_passno of 1. Other filesystems
should have a fs_passno of 2. Filesystems within a drive will be
checked sequentially, but filesystems on different drives will be
checked at the same time to utilize parallelism available in the


Activities
Firefox Web Browser
abr 11 18:14
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
https://man7.org/linux/man-pages/man5/fstab.5.html
type (including performance-tuning options). For details, see
mount(8) or swapon(8).
Basic filesystem-independent options are:
defaults
use default options. The default depends on the kernel and
the filesystem. mount(8) does not have any hardcoded set of
default options. The kernel default is usually rw, suid, dev,
exec, auto, nouser, and async.
noauto
do not mount when mount -a is given (e.g., at boot time)
user
allow a user to mount
owner
allow device owner to mount
comment
or x-<name> for use by fstab-maintaining programs
nofail
do not report errors for this device if it does not exist.
The fifth field (fs_freq).
This field is used by dump(8) to determine which filesystems need
to be dumped. Defaults to zero (don't dump) if not present.
The sixth field (fs_passno).
This field is used by fsck(8) to determine the order in which
filesystem checks are done at boot time. The root filesystem
should be specified with a fs_passno of 1. Other filesystems
should have a fs_passno of 2. Filesystems within a drive will be
checked sequentially, but filesystems on different drives will be
checked at the same time to utilize parallelism available in the
hardware. Defaults to zero (don't check the filesystem) if not
present.
FILES
top
/etc/fstab, <fstab.h>
NOTES
top
The proper way to read records from fstab is to use the routines
getmntent(3) or libmount.


Activities
Terminal
abr 11 18:15
crypttab
crypttab(5) - Linux manual p
fstab(5) - Linux manual page
Private browsing
alvaro@alvaro-ubuntu: ~
alvaro@alvaro-ubuntu:~$ cat /etc/fstab
/etc/fstab: static file system information.
#
Use 'blkid' to print the universally unique identifier for a
device; this may be used with UUID= as a more robust way to name devices
that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type> <options>   <dump> <pass>
# / was on /dev/sda3 during installation
UUID=4fec3f00-afd8-49c7-a78a-8a5f036f41d8 /   ext4   errors=remount-ro 0   1
# /boot/efi was on /dev/sda2 during installation
UUID=89A6-2541 /boot/efi   vfat   umask=0077   0   1
/swapfile   none   swap   sw   0   0
alvaro@alvaro-ubuntu:~$
FILES
NOTES
The proper way to read records from fstab is to use the routines
getmntent(3) or libmount.
The keyword ignore as a filesystem type (3rd field) is no longer
supported by the pure libmount based mount utility (since
util-linux v2.22).
HISTORY
top
The ancestor of this fstab file format appeared in 4.0BSD.
SEE ALSO
top
getmntent(3), fs(5), findmnt(8), mount(8), swapon(8)
REPORTING BUGS
top
For bug reports, use the issue tracker at
https://github.com/util-linux/util-linux/issues.
10:15
vimeo