# netctl

From ArchWiki

**netctl** is a CLI-based tool used to configure and manage network connections via profiles. It is a native Arch Linux project for network configuration.

Related articles

Bridge with netctl

Network configuration

Wireless network configuration

Category:Network managers

## Contents



## Installation

Install netctl (https://www.archlinux.org/packages/?name=netctl) from the official repositories.

Optional dependencies are shown in the table below.

| Feature | Dependency | netctl program (if relevant) |
|---|---|---|
| Automatic wireless connections | wpa_actiond (https://www.archlinux.org/packages/?name=wpa_actiond) | netctl-auto |
| Automatic wired connections | ifplugd (https://www.archlinux.org/packages/?name=ifplugd) | netctl-ifplugd |
| WPA | wpa_supplicant (https://www.archlinux.org/packages/?name=wpa_supplicant) | |
| DHCP | dhcpcd (https://www.archlinux.org/packages/?name=dhcpcd) or dhclient (https://www.archlinux.org/packages/?name=dhclient) | |
| Wifi menus | dialog (https://www.archlinux.org/packages/?name=dialog) | |
| PPPoE | ppp (https://www.archlinux.org/packages/?name=ppp) | |

**Warning:** Do not enable concurrent, conflicting network service. Use `systemctl --type=service` to ensure that no other network service is running before enabling a *netctl* profile/service.

# Usage

It is advisable to read the following man pages before using netctl:

- netctl (https://github.com/joukewitteveen/netctl/blob/master/docs/netctl.1.txt)
- netctl.profile (https://github.com/joukewitteveen/netctl/blob/master/docs/netctl.profile.5.txt)
- netctl.special (https://github.com/joukewitteveen/netctl/blob/master/docs/netctl.special.7.txt)

# Configuration

*netctl* uses profiles to manage network connections and different modes of operation to start profiles automatically or manually on demand.

## Profile configuration

The *netctl* profile files are stored in `/etc/netctl/` and example configuration files are available in `/etc/netctl/examples/`. Common configurations include:

- ethernet-dhcp
- ethernet-static
- wireless-wpa
- wireless-wpa-static

To use an example profile, simply copy it from `/etc/netctl/examples/` to `/etc/netctl/` and configure it to your needs; see basic #Example profiles below. The first parameter you need to create a profile is the network `Interface`, see Network configuration#Device names for details.

**Tip:**

- For wireless settings, you can use `wifi-menu -o` as root to generate the profile file in `/etc/netctl/`.
- To enable a static IP profile on wired interface no matter if the cable is connected or not, use `SkipNoCarrier=yes` in your profile.

Once you have created your profile, attempt to establish a connection (use only the profile name, not the full path):

```
# netctl start profile
```

If the above command results in a failure, then use `journalctl -xn` and `netctl status profile` to obtain a more in depth explanation of the failure.

## Automatic operation

If you use only one profile (per interface) or want to switch profiles manually, the Basic method will do. Most common examples are servers, workstations, routers etc.

If you need to switch multiple profiles frequently, use Automatic switching of profiles. Most common examples are laptops.

### Basic method

With this method, you can statically start only one profile per interface. First manually check that the profile can be started successfully, then it can be enabled using

```
# netctl enable profile
```

This will create and enable a systemd service that will start when the computer boots. Changes to the profile file will not propagate to the service file automatically. After such changes, it is necessary to reenable the profile:

```
# netctl reenable profile
```

After enabling a profile, it will be started at next boot. Obviously this can only be successful, if the network cable for a wired connection is plugged in, or the wireless access point used in a profile is in range respectively.

### Automatic switching of profiles

*netctl* provides two special systemd services for automatic switching of profiles:

- Package ifplugd (https://www.archlinux.org/packages/?name=ifplugd) for wired interfaces: After starting and enabling `netctl-ifplugd@`*interface*`.service` profiles are started/stopped when the network cable is plugged in and out.
- Package wpa_actiond (https://www.archlinux.org/packages/?name=wpa_actiond) for wireless interfaces: After starting and enabling `netctl-auto@`*interface*`.service` profiles are started/stopped automatically as you move from the range of one network into the range of another network (roaming).

Both services will refer to *all* profiles created at `/etc/netctl/`. The following options can be used:

- If you want some wireless profile **not** to be started automatically by `netctl-auto@`*interface*`.service`, you have to explicitly add `ExcludeAuto=yes` to that profile.
- The `netctl-ifplugd@`*interface*`.service` will prefer profiles, which use DHCP. To prefer a profile with a static IP, you can use `AutoWired=yes`. See `netctl.profile(5)` for details.
- You can use `Priority=` in the *WPAConfigSection* (see `/etc/netctl/examples/wireless-wpa-configsection`) to set priority of a profile when multiple wireless access points are available. Note that automatic selection of a WPA profile by *netctl-auto* is not possible with option `Security=wpa-config`, use `Security=wpa-configsection` instead.

**Warning:**

- If any of the profiles contain errors, such as an empty or misquoted `Key=` variable, the unit will fail to load with the message `"Failed to read or parse configuration '/run/network/wpa_supplicant_wlan0.conf'`, even when that profile is not being used.
- This method conflicts with the Basic method. If you have previously enabled a profile through *netctl*, run `netctl disable` *profile* to prevent the profile from starting twice at boot.

Since netctl 1.3, it possible to manually control an interface otherwise managed by netctl-auto without having to stop the netctl-auto service. This is done using the netctl-auto command. To have a list of available actions just run:

```
# netctl-auto --help
```

## Example profiles

### Wired

For a DHCP connection, only the `Interface` has to be configured after copying the `/etc/netctl/ethernet-dhcp` example profile to `/etc/netctl`.

For a static IP configuration see the example in Beginners' guide#Static IP.

### Wireless (WPA-PSK)

The following applies for the standard wireless connections using a pre-shared key (WPA-PSK). See WPA2 Enterprise#netctl for example profiles with other authentication methods.

The standard *netctl* tool to connect to a wireless network (WPA-PSK, WEP) interactively is *wifi-menu*; used with the `-o` option:

```
wifi-menu -o
```

it generates the configuration file in `/etc/netctl/` for the network to use for #Automatic operation at the same time.

Alternatively, the profile may also be configured manually, as follows:

Copy the example file `wireless-wpa` from `/etc/netctl/examples` to `/etc/netctl` (name of your choice):

```
# cp /etc/netctl/examples/wireless-wpa /etc/netctl/.
```

Edit the profile as needed (modifying `Interface`, `ESSID` and `Key`) and it is done.

At this step you may want to re-confirm the new profile you created is `chmod 600` and confirm it works by starting it:

```
netctl start wireless-wpa
```

before configuring any #Automatic operation.

Optionally you can also follow the following step to obfuscate the wireless passphrase (*wifi-menu* does it automatically):

Users **not** wishing to have the passphrase to their wireless network stored in *plain text* have the option of storing the corresponding 256-bit pre-shared key instead, which is calculated from the passphrase and the SSID using standard algorithms.

Calculate your 256-bit PSK using wpa_passphrase:

```
$ wpa_passphrase your_essid passphrase
```

```
network={
  ssid="your_essid"
  #psk="passphrase"
  psk=64cf3ced850ecef39197bb7b7b301fc39437a6aa6c6a599d0534b16af578e04a
}
```

The *pre-shared key* (psk) now needs to replaces the plain text passphrase of the `Key` variable in the profile. Once completed your network profile `wireless-wpa` containing a 256-bit PSK should resemble:

/etc/netctl/wireless-wpa

```
Description='A simple WPA encrypted wireless connection using 256-bit PSK'
Interface=wlp2s2
Connection=wireless
Security=wpa
IP=dhcp
ESSID=your_essid
Key=\"64cf3ced850ecef39197bb7b7b301fc39437a6aa6c6a599d0534b16af578e04a
```

**Note:**

- Make sure to use the **special quoting rules** for the `Key` variable as explained at the end of netctl.profile(5) (https://github.com/joukewitteveen/netctl/blob/master/docs/netctl.profile.5.txt)
- If the passphrase fails, try removing the `\"` in the `Key` variable.
- Although "encrypted", the key that you put in the profile configuration is enough to connect to a WPA-PSK network. Therefore this process is only useful for hiding the human-readable version of the passphrase. This will not prevent anyone with read access to this file from connecting to the network.

# Tips and tricks

## Using an Experimental GUI

If you want a graphical user interface to manage *netctl* and your connections and you are not afraid of highly experimental unofficial packages you can install `netgui` (https://aur.archlinux.org/packages/netgui/) from AUR. Note, however, that *netgui* is still in beta status and you should be familiar with the general *netctl* syntax to debug possible issues. Another GUI alternative is `netctl-gui` (https://aur.archlinux.org/packages/netctl-gui/) which provides a Qt-based graphical interface, DBus daemon and KDE widget.

## Replace 'netcfg current'

If you used `netcfg current` in the past, you can use `# netctl-auto current` as a replacement for connections started with `netctl-auto` (feature since netctl-1.3).

To manually parse the connections, you can also use:

```
# netctl list | awk '/*/ {print $2}'
```

## Eduroam

See WPA2_Enterprise#netctl.

## Bonding

From kernel documentation (https://www.kernel.org/doc/Documentation/networking/bonding.txt):

> *The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface. The behavior of the bonded interfaces depends on the mode. Generally speaking, modes provide either hot standby or load balancing services. Additionally, link integrity monitoring may be performed.*

### Load balancing

To use bonding with netctl, additional package from official repositories is required: `ifenslave (https://www.archlinux.org/packages/?name=ifenslave)`.

Copy `/etc/netctl/examples/bonding` to `/etc/netctl/bonding` and edit it, for example:

/etc/netctl/bonding

```
Description='Bond Interface'
Interface='bond0'
Connection=bond
BindsToInterfaces=('eth0' 'eth1')
IP=dhcp
IP6=stateless
```

Now you can disable your old configuration and set *bonding* to be started automatically. Switch to the new profile, for example:

```
# netctl switch-to bonding
```

**Note:** This uses the round-robin policy, which is the default for the `bonding` driver. See official documentation (https://www.kernel.org/doc/Documentation/networking/bonding.txt) for details.

**Tip:** To check the status and bonding mode:

```
$ cat /proc/net/bonding/bond0
```

### Wired to wireless failover

This example describes how to use *bonding* to fallback to wireless when the wired ethernet goes down. This is most useful when both the wired and wireless interface will be connected to the same network. Your wireless router/access point must be configured in *bridge* mode.

You will need additional packages from the official repositories: `ifenslave` (https://www.archlinux.org/packages/?name=ifenslave) and `wpa_supplicant` (https://www.archlinux.org/packages/?name=wpa_supplicant).

Fisrt enable the bonding module to be loaded upon boot time, as instructed on Kernel modules#Loading:

/etc/modules-load.d/bonding.conf

```
bonding
```

Then, configure the options of the `bonding` driver to use `active-backup` and configure the `primary` parameter to the device you want to be the active one (normally the wired interface). Also, be sure to use the same device name as returned when running `ip link`:

/etc/modprobe.d/bonding.conf

```
options bonding mode=active-backup miimon=100 primary=eth0 max_bonds=0
```

The `miimon` option is needed, for the link failure detection. The `max_bonds` option avoids the `Interface bond0 already exists` error. More information can be obtained on the kernel documentation (https://www.kernel.org/doc/Documentation/networking/bonding.txt).

Next, configure a netctl profile to enslave the two hardware interfaces. Use the name of all the devices you want to enslave. If you have more than two wired or wireless interfaces, you can enslave all of them on a bond interface. But, for most cases you will have only two devices, a wired and a wireless one:

/etc/netctl/failover

```
Description='A wired connection with failover to wireless'
Interface='bond0'
Connection=bond
```

```
BindsToInterfaces=('eth0' 'wlan0')
IP='dhcp'
```

Disable any other profiles (specially a wired or wireless) you had enabled before and then enable the failover profile on startup:

```
# netctl enable failover
```

Now you need to configure *wpa_supplicant* to connect to any know network you wish. You should create a file for each interface and enable it on systemd. Create the following file with this content:

```
/etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

```
ctrl_interface=/run/wpa_supplicant
update_config=1
```

And append to the end of this file any networks you want to connect:

```
network={
    ssid="SSID"
    psk=PSK
}
```

To generate the obfuscated PSK you can run *wpa_passphrase* as on the WPA supplicant#Connecting with wpa_passphrase page.

Now, enable the `wpa_supplicant` service on the network interface:

```
# systemctl enable wpa_supplicant@wlan0
```

You can try now to reboot your machine and see if your configuration worked.

**Note:** If you get this error on boot bonding:

```
wlan0 is up - this may be due to an out of date ifenslave
```

Then this is happening because the *wpa_supplicant* is being run before the `failover` netctl profile. This happens because systemd runs everything in parallel, unless told otherwise. *ifenslave* need all the interfaces to be down before bonding them to the `bond0` interface. And, since the *wpa_supplicant* need to put the interface up to be able to scan for networks, this might cause the interface to not be enslaved and your bonding to only have the wired interface.

If this is your case, then you will need to setup a custom dependency on the `wpa_supplicant@wlan0` service in relation with the `netctl@failover` profile. More specifically, the *wpa_supplicant* must be started **after** the netctl profile. To accomplish this, create a custom dependency file based on the instructions provided here: systemd#Handling dependencies

/etc/systemd/system/wpa_supplicant@wlan0.service.d/customdependency.conf

```
[Unit]
After=netctl@failover.service
```

After that you can try to reboot your system again and see if it works. You can check the status of your bonding by running:

```
# journalctl -u netctl@failover.service
```

And:

```
# ip link
```

You should see something like this:

```
1: eth0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond0 state UP
    link/ether xx:xx:xx:xx:xx:xx brd ff:ff:ff:ff:ff:ff
2: wlan0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond0 state UP mode D
    link/ether xx:xx:xx:xx:xx:xx brd ff:ff:ff:ff:ff:ff
3: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
    link/ether xx:xx:xx:xx:xx:xx brd ff:ff:ff:ff:ff:ff
```

Now, you can test your failover setup, by initiating a big download. Unplug your wired interface. Your download should keep going over the wireless interface. Then, plug your wired interface again and it should keep working. You can debug with the following comands:

```
# journalctl -u netctl@failover.service
```

And:

```
# journalctl -u wpa_supplicant@wlan0.service
```

## Using any interface

In some cases it may be desirable to allow a profile to use any interface on the system. A common example use case is using a common disk image across many machines with differing hardware (this is especially useful if they are headless). If you use the kernel's naming scheme, and your machine has only one ethernet interface, you can probably guess that eth0 is the right interface. If you use udev's Predictable Network Interface Names (http://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/), however, names will be assigned based on the specific hardware itself (e.g. enp1s0), rather than simply the order that the hardware was detected (e.g. eth0, eth1). This means that a netctl profile may work on one machine and not another, because they each have different interface names.

A quick and dirty solution is to make use of the `/etc/netctl/interfaces/` directory. Choose a name for your interface alias ( `en-any` in this example), and write the following to a file with that name (making sure it is executable).

/etc/netctl/interfaces/en-any

```
#!/bin/bash
for interface in /sys/class/net/en*; do
        break;
done
Interface=$(basename $interface)
echo "en-any: using interface $Interface";
```

Then create a profile that uses the interface. Pay special attention to the `Interface` directive. The rest are only provided as examples.

/etc/netctl/wired

```
Description='Wired'
Interface=en-any
Connection=ethernet
IP=static
Address=('192.168.1.15/24')
Gateway='192.168.1.1'
DNS=('192.168.1.1')
```

When the `wired` profile is started, any machine using the two files above will automatically bring up and configure the first ethernet interface found on the system, regardless of what name udev assigned to it. Note that this is not the most robust way to go about configuring interfaces. If you use multiple interfaces, netctl may try to assign the same interface to them, and will likely cause a disruption in connectivity. If you do not mind a more complicated solution, `netctl-auto` is likely to be more reliable.

## Using hooks

netctl supports hooks in `/etc/netctl/hooks/` and per interface hooks in `/etc/netctl/interfaces/`. You can set any option in a hook/interface that you can in a profile. They are read the same way! Most importantly this includes `ExecUpPost` and `ExecDownPre`.

When a profile is read, netctl sources *all executable* scripts in `hooks`, then it reads the profile file for the connection and finally it sources an executable script with the name of the interface used in the profile from the `interfaces` directory. Therefore, declarations in an interface script override declarations in the profile, which override declarations in hooks.

The variables `$INTERFACE`, `$SSID`, `$ACTION` and `$Profile` are available in hooks/interfaces **only** when using `netctl-auto`

## Examples

### Execute commands on established connection

```
/etc/netctl/hooks/myservices
```

```
#!/bin/sh
ExecUpPost="systemctl start crashplan.service; systemctl start dropbox@<username>.service"
ExecDownPre="systemctl stop crashplan.service; systemctl stop dropbox@<username>.service"
```

### Activate network-online.target

```
/etc/netctl/hooks/status
```

```
#!/bin/sh
ExecUpPost="systemctl start network-online.target"
ExecDownPre="systemctl stop network-online.target"
```

Using this, systemd services requiring an active network connection can be ordered to start only after the `network-online.target` is reached, and can be stopped before the connection is brought down.

### Set default DHCP client

To set or change the DHCP client used for all profiles:

```
/etc/netctl/hooks/dhcp
```

```
#!/bin/sh
DHCPClient='dhclient'
```

Alternatively, it may also be specified for a specific network interface by creating an executable file `/etc/netctl/interfaces/<interface>` with the following line:

```
DHCPClient='dhclient'
```

# Troubleshooting

## Job for netctl@wlan(...).service failed

Some people have an issue when they connect to a network with *netctl*, for example:

```
# netctl start wlan0-ssid
```

```
Job for netctl@wlan0\x2ssid.service failed. See 'systemctl status netctl@wlan0\x2ssid.service' and
```

When then looking at `journalctl -xn`, either of the following are shown:

1. If your device (`wlan0` in this case) is up:

```
network[2322]: The interface of network profile 'wlan0-ssid' is already up
```

Setting the interface down should resolve the problem:

```
# ip link set wlan0 down
```

Then retry:

```
# netctl start wlan0-ssid
```

2. If it is down:

```
dhcpcd[261]: wlan0: ipv4_sendrawpacket: Network is down
```

One way to solve this is to use a different DHCP client, for example `dhclient` (https://www.archlinux.org/packages/?name=dhclient). After installing the package configure *netctl* to use it:

```
/etc/netctl/wlan0-ssid
```

```
...
DHCPClient='dhclient'
```

Adding the `ForceConnect` option may also be helpful:

/etc/netctl/wlan0-ssid

```
...
ForceConnect=yes
```

Save it and try to connect with the profile:

```
# netctl start wlan0-ssid
```

## dhcpcd: ipv4_addroute: File exists

On some systems dhcpcd in combination with netctl causes timeout issues on resume, particularly when having swichted networks in the meantime. netctl will report that you are successfully connected but you still receive timeout issues. In this case, the old default route still exists and is not being renewed. A workaround to avoid this misbehaviour is to switch to dhclient as the default dhcp client. More information on the issue can be found here (https://bbs.archlinux.org/viewtopic.php?pid=1399842#p1399842).

## DHCP timeout issues

If you are having timeout issues when requesting leases via DHCP you can set the timeout value higher than netctl's 30 seconds by default. Create a file in `/etc/netctl/hooks/` or `/etc/netctl/interfaces/`, add `TimeoutDHCP=40` to it for a timeout of 40 seconds and make the file executable.

## Connection timeout issues

If you are having timeout issues that are unrelated to DHCP (on a static ethernet connection for example), and are experiencing errors similar to the following when starting your profile:

```
# journalctl _SYSTEMD_UNIT=netctl@profile.service
```

```
Starting network profile 'profile'...
No connection found on interface 'eth0' (timeout)
Failed to bring the network up for profile 'profile'
```

Then you should increase carrier and up timeouts by adding `TimeoutUp=` and `TimeoutCarrier=` to your profile file:

/etc/netctl/*profile*

```
...
TimeoutUp=300
```

```
TimeoutCarrier=300
```

Do not forget to reenable your profile:

```
# netctl reenable profile
```

## Problems with netctl-auto on resume

Sometimes *netctl-auto* fails to reconnect when the system resumes from suspend. An easy solution is to restart the service for *netctl-auto*. This can be automated with an additional service like the following:

```
/etc/systemd/system/netctl-auto-resume@.service
```

```
[Unit]
Description=restart netctl-auto on resume.
Requisite=netctl-auto@%i.service
After=suspend.target

[Service]
Type=oneshot
ExecStart=/usr/bin/systemctl restart netctl-auto@%i.service

[Install]
WantedBy=suspend.target
```

To enable this service for your wireless card, for example, run `systemctl enable netctl-auto-resume@wlan0.service` as root. Change `wlan0` to the required network interface.

## Migrating from netcfg

*netctl* uses `/etc/netctl/` to store its profiles, **not** `/etc/network.d/` (used by *netcfg*).

In order to migrate from *netcfg*, at least the following is needed:

- Disable the netcfg service: `systemctl disable netcfg.service` .
- Uninstall *netcfg* and install *netctl*.
- Move network profile files to the new directory.
- Rename variables therein according to `netctl.profile(5)` (Most variable names have only `UpperCamelCase` i.e `CONNECTION` becomes `Connection` ).
- For static IP configuration make sure the `Address` variables have a netmask after the IP (e.g. `Address=('192.168.1.23`**`/24`**`' '192.168.1.87`**`/24`**`')` in the example profile).
- If you setup a wireless profile according in the `wireless-wpa-configsection` example, note that this overrides `wpa_supplicant` options defined above the brackets. For a connection to a hidden wireless network, add `scan_ssid=1` to the options in the `wireless-wpa-configsection` ; `Hidden=yes` does not work there.

- Unquote interface variables and other variables that do not strictly need quoting (this is mainly a style thing).
- Run `netctl enable *profile*` for every profile in the old `NETWORKS` array. *last* option does not work this way, see the description of *netctl.service* in netctl.special(7) (https://github.com/joukewitteveen/netctl/blob/master/docs/netctl.special.7.txt).
- Use `netctl list` and `netctl start *profile*` instead of *netcfg-menu*. *wifi-menu* remains available.
- Unlike *netcfg*, by default *netctl* fails to bring up a NIC when it is not connected to another powered up NIC. To solve this problem, add `SkipNoCarrier=yes` at the end of your `/etc/netctl/*profile*`.

# See also

- Official announcement thread (https://bbs.archlinux.org/viewtopic.php?id=157670)
- There is a cinnamon applet available in the AUR: `cinnamon-applet-netctl-systray-menu` (https://aur.archlinux.org/packages/cinnamon-applet-netctl-systray-menu/)

Retrieved from "https://wiki.archlinux.org/index.php?title=Netctl&oldid=356284"
Category: Network managers

---