

Если вы видите что-то необычное, просто сообщите мне.

# Linux

- [Запуск скрипта в качестве службы systemd](#)
- [Проблемы различия библиотек при компиляции](#)
- [Проброс портов в туннеле ssh](#)
- [Store the output of cron jobs in systemd's journal](#)
- [How to Reset Forgotten Root Password in Debian 10](#)

# Запуск скрипта в качестве службы systemd

Создаем скрипт runsmth.sh для выполнения команд в вечном цикле:

```
#!/bin/bash

#Строчки указаны для примера, если требуется указание переменных
export SMTH_CONFIG_FILE=/home/admin/.smth/config
export SMTH_CREDENTIALS_FILE=/home/admin/.smth/credentials

while true
do
    ls /home/$USER
    sleep 10
done
```

Создаем файл для systemd:

```
sudo vim /lib/systemd/system/smth.service
```

и вносим в него следующее содержание:

```
[Unit]
Description=script to do smth

[Service]
ExecStart=/home/admin/runsmth.sh

[Install]
```

```
WantedBy=multi-user.target
```

## Перезагружаем настройки systemd:

```
sudo systemctl daemon-reload
```

## Стартуем сервис systemd:

- Чтобы каждый раз включался при загрузке:

```
sudo systemctl enable smth.service -now
```

- Просто запустить сервис, при перезагрузке не запуститься:

```
sudo systemctl start smth.service
```

## Проверяем состояние запущенного сервиса:

```
sudo journalctl -f -n 10 -u smth.service
```

# Проблемы различия библиотек при компиляции

## Ошибка

Во время компиляции и не соответствия версий систем может возникнуть подобная ошибка

```
/lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.28' not found (required by appname)
```

Ошибка говорит о том, что в системе нет версии `GLIBC_2.28`

Чтобы узнать какая версия находится в системе можно воспользоваться следующей командой

```
ls /lib/x86_64-linux-gnu/libc
libc-2.24.so      libcap.so.2.25    libcom_err.so.2.1  libcrypt.so.1
libcap-ng.so.0   libcidn-2.24.so   libcrypt-2.24.so   libc.so.6
libcap-ng.so.0.0.0 libcidn.so.1      libcryptsetup.so.4
libcap.so.2       libcom_err.so.2   libcryptsetup.so.4.7.0
```

Отсюда мы видим, что системная библиотека отличается от системы на которой собирается приложение: 2.24 вместо 2.28

## Решение

Для того, чтобы обойти эту проблему можем воспользоваться docker.

Узнаем какая версия операционной системы у нас стоит:

```
uname -a
```

```
Linux hostname 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1+deb9u1 (2020-06-07) x86_64 GNU/Linux
```

Мы видим, что наша операционная система Debian 9.1 что сильно упрощает нам поиск нужного образа. Находим в Docker hub нужный нам образ, в нашем случае, всё очень просто, дебиан тегирует образы по разному и наш образ будет `debian:9.1`, это завсит от авторов образа. Его мы и будем использовать. Устанавливаем необходимые пакеты `wget` и `gcc` и скачиваем свеженький `go`.

1. В корне проекта над которым работаем создаем `Dockerfile`.

```
FROM debian:9.1
RUN apt update && apt install wget gcc -y
RUN wget https://go.dev/dl/go1.18.4.linux-amd64.tar.gz
RUN rm -rf /usr/local/go && tar -C /usr/local -xzf go1.18.4.linux-amd64.tar.gz
WORKDIR /code
```

2. Там же создаем `docker-compose.yml` который будем использовать для компиляции проекта:

```
version: "3"

services:
  servicename:
    build: .
    volumes:
      - "./code"
```

3. Билдим докер и в нем собираем наш проект:

```
docker-compose build --no-cache
docker-compose run  servicename bash -c 'export PATH=$PATH:/usr/local/go/bin && GOOS=linux
GOARCH=amd64 go build -o bin/appname .'
```

Первую команду в дальнейшем можно не запускать.

# Послесловие

Таким способом можно решать различные проблемы возникающие при сборке приложения на окружениях отличных от того окружения где будет происходить работа. Так в случае необходимости, можно изменять Dockerfile до состояния близкого к состоянию рабочего окружения. В этом случае команда `docker-compose build --no-cache` обязательна.

# Проброс портов в туннеле ssh

## Создаем постоянное подключение к удаленном серверу.

Требования:

1. Удаленный доступ должен иметь настроенный ssh сервер
2. Локально должен быть доступен ключ id\_rsa
3. Публичная часть ключа должна быть на удаленном сервере

Настройки:

Создаем файл `/etc/systemd/system/SERVICENAME.service` с содержанием вида

```
[Unit]
Description=SSH tunnel

[Service]
ExecStart=bash -c "ssh -i /PATH/TO/id_rsa -L 0.0.0.0:9999:localhost:8010 USER@REMOTE.HOST -N"
Restart=on-failure
EnvironmentFile=/etc/environment

[Install]
WantedBy=multi-user.target
```

Мы говорим, что на хосте `REMOTE.HOST` по адресу `localhost:8010` доступно приложение, и его мы мапируем на локальный компьютер на порт 9999.

Теперь удаленно приложение висящее на порту 8010, доступно локально, но на порту 9999. Любое подключение идущее на `localhost:9999` компьютера на котором запускается сервис будет уходить на `REMOTE.HOST` через ssh туннель.

`0.0.0.0:9999` - подключение будет доступно не только на localhost но и еще в локальной сети компьютера на котором запущен текущий сервис. `0.0.0.0` можно не писать.

## Запускаем/останавливаем сервис:

```
systemctl start/stop SERVICENAME.service
```

`start/stop` - выберите что-то одно, или запускаем или останавливаем

**Если демон просит перезагрузить конфиги - перезагружаем**

## Включаем сервис при загрузке:

```
systemctl enable SERVICENAME.service
```

## Проверяем состояние сервиса на наличие ошибок:

```
systemctl status SERVICENAME.service
```

или

```
journalctl -u SERVICENAME
```

# Немного теории

Описываю реальную ситуацию: есть удаленный сервер (назовем его `remoteserver`), на нем крутятся несколько контейнеров, один из них база данных (пусть будет `docker_mysql_1`). Разработчику (его комп пусть будет `developer`), находящемуся в одной локалке с девопсом (его хост - `devops`) понадобился доступ к порту `mysql` на том удаленном сервере. Из них

двоих только у девопса есть доступ до сервера (remoteserver) - по ssh.

То есть, разработчик хочет запустить приложение (которое запросит хост, порт, логин, пароль), укажет этому приложению настройки и оно должно соединиться с базой данных, которая находится на удаленном сервере в контейнере, порт которого не проброшен.

Конечно, можно добавить проброс порта в docker-compose или перезапустить отдельный этот контейнер с нужными параметрами, но как сделать все это не трогая контейнер, не перезапуская и не устанавливая ни одной программы ни на одной машине.

Ответ - последняя команда этой страницы.

Теперь по порядку: ssh может создавать туннели, пробрасывая внутренние порты сервера на машину, с которой была запущена команда.

Например, следующая команда, если ее выполнит девопс, открыла бы на его машине (devops) порт 9999, который был бы связан с портом 3306 (mysql) на сервере remoteserver. Этот порт был бы доступен только ему, так как открылся бы на интерфейсе localhost.

```
ssh -L 9999:localhost:3306 user@remoteserver
```

Если для подключения используется нестандартный порт, то не нужно забывать это указывать:

```
ssh -p 32323 -L 9999:localhost:3306 user@remoteserver
```

Но мне кажется, что хорошей практикой было бы включить такие настройки в .ssh/config.

Далее, если нам не нужен сам ssh-туннель, а только доступ к порту, то мы можем не запускать интерпретатор:

```
ssh -p 32323 -L 9999:localhost:3306 user@remoteserver -N
```

Тут мы вспомнили, что ломиться нужно не на localhost, а на определенный хост, доступный ему по сети (виртуальной или реальной - не важно). Смотрим ip-адрес нужного нам контейнера через

```
docker inspect docker_mysql_1
```

и затем указываем полученный ip-адрес в команде.

```
ssh -p 32323 -L 9999:172.31.0.2:3306 user@remoteserver -N
```

Теперь у нас все получилось и девопс выполняя команду `mysql -h localhost --port 9999` он попадает на порт mysql'a, который работает в контейнере `docker_mysql_1`, запущенном на `remoteserver`.

Осталось дать доступ разработчику. Следующая команда разрешает пользоваться данным портом через все интерфейсы, а значит он доступен не только девопсу, но и разрабу (команда уже будет `mysql -h devops --port 9999`):

```
ssh -p 32323 -L 0.0.0.0:9999:172.31.0.2:3306 user@remoteserver -N
```

# Store the output of cron jobs in systemd's journal

The output from cronjobs is often unceremoniously redirected to `/dev/null` just to keep them quiet. This is a bad practice as you don't get any feedback when potentially important tasks go wrong. (If the task isn't important, why are you running it on a schedule?)

systems provide a simple tool called `systemd-cat` for directing `stdout` and `stderr` to the journal. Below is a complete example of a simple monthly `cron` job that will run certbot once every month and save whatever it outputs to the journal under the arbitrary identifier "certbot-cron". A good identifier will make it easier to look up the output in the journal if you need it later.

```
@monthly systemd-cat -t "certbot-cron" /usr/bin/certbot --renew
```

I'll use the automated TLS certificate renewal program `certbot` from Let's Encrypt as the example command throughout this article. It serves nicely as an example of something you need to know about if it at some point will return an unsuccessful condition.

You can then quickly look at certbot output in the journal using `journalctl` with the same identifier that you gave the `cron` job:

```
journalctl -t "certbot-cron"
```

`systemd-cat` only accepts one executable with any number of arguments. You can't run multiple commands at once or use pipes. It will capture `stdout` and `stderr`, and assign them appropriate properties in the journal.

You can pipe output into `systemd-cat`, but this will only capture stdout. Important and actionable information will often show up in stderr, and you'd have to redirect the flow of `stderr` to `stdout` to capture it. I'll not demonstrate this here, as I believe this is the wrong approach.

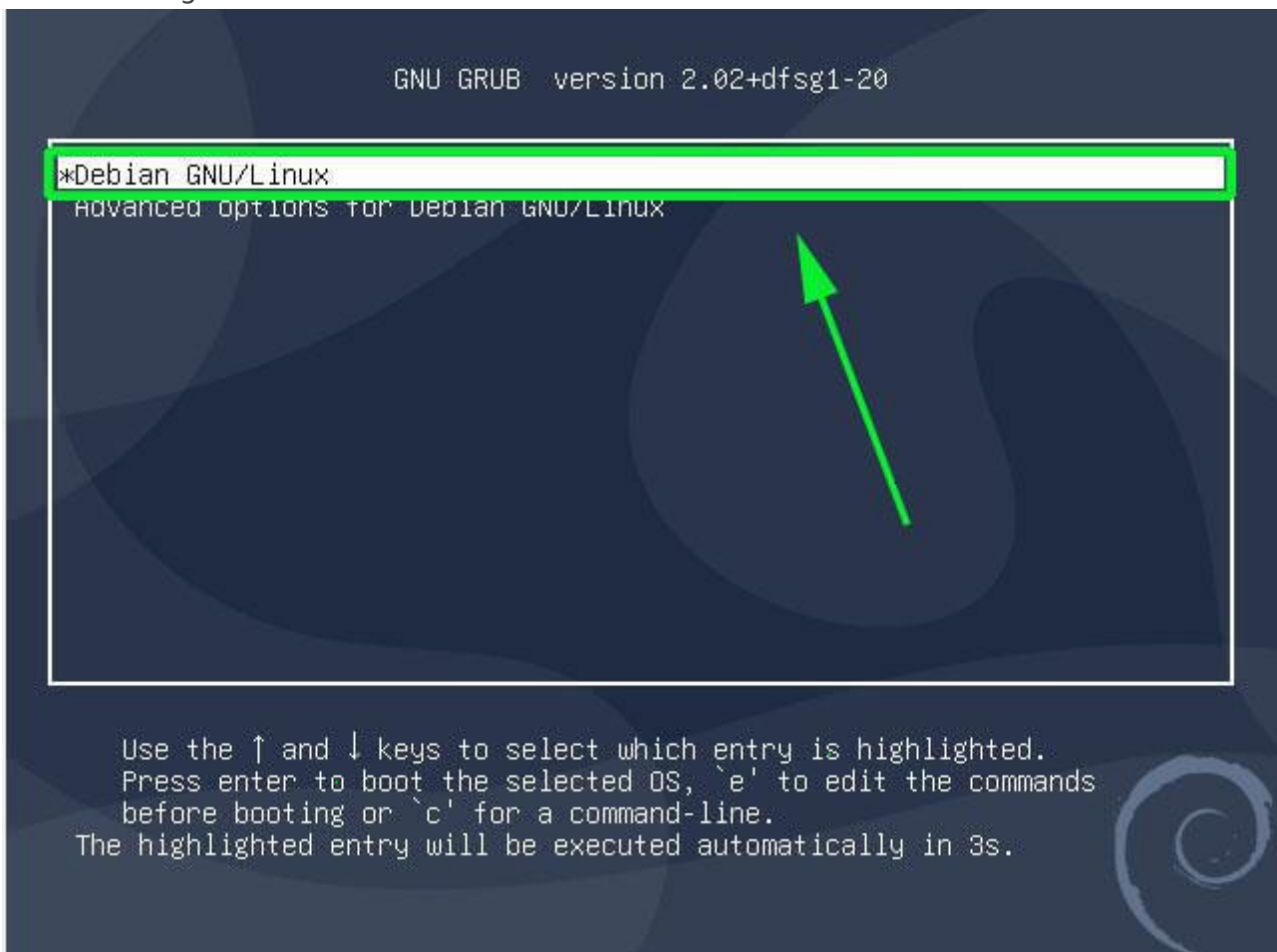
If you need to run more complex cron jobs with multiple commands, you should save them to a script, make that executable, and then run that script through `systemd-cat` as a single command.

This will capture `stdout` as well as `stderr`, and keep your crontab more readable and easier to maintain.

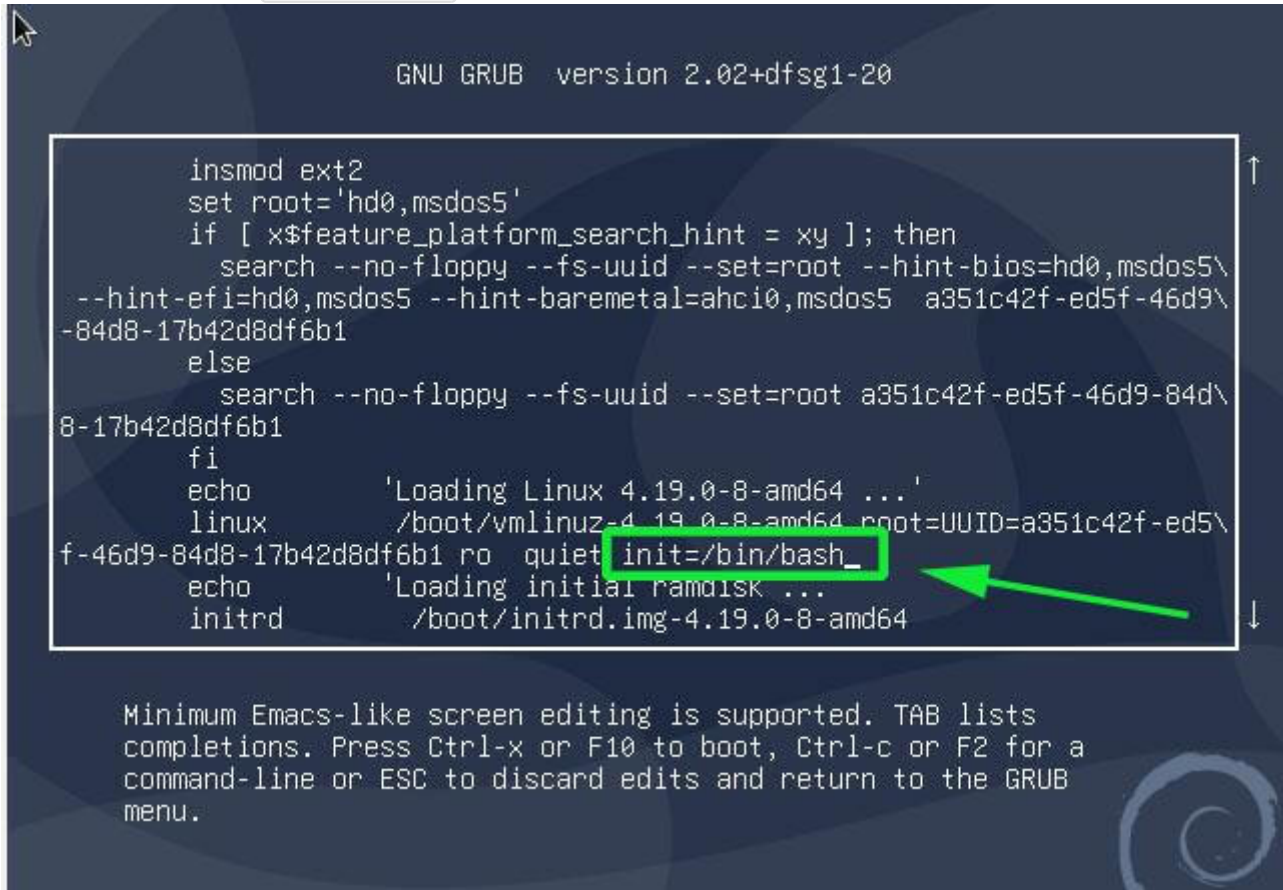
# How to Reset Forgotten Root Password in Debian 10

In this brief tutorial, you will learn how to reset a forgotten root password in a Debian 10 system. This will help you regain the ability to log in as the root user and carry out administrative tasks.

So, first power on or reboot your Debian 10 system. You should be presented with a GRUB menu as shown below. On the first option, proceed and press the **e** key on the keyboard before the system starts booting.



This ushers you to the screen shown below. Scroll down and locate the line that begins with `linux` that precedes the `/boot/vmlinuz-*` section that also specifies the UUID.



```
GNU GRUB  version 2.02+dfsg1-20

insmod ext2
set root='hd0,msdos5'
if [ x$feature_platform_search_hint = xy ]; then
  search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos5\
--hint-efi=hd0,msdos5 --hint-baremetal=ahci0,msdos5  a351c42f-ed5f-46d9\
-84d8-17b42d8df6b1
else
  search --no-floppy --fs-uuid --set=root a351c42f-ed5f-46d9-84d\
8-17b42d8df6b1
fi
echo      'Loading Linux 4.19.0-8-amd64 ...'
linux     /boot/vmlinuz-4.19.0-8-amd64 root=UUID=a351c42f-ed5\
f-46d9-84d8-17b42d8df6b1 ro quiet init=/bin/bash_
echo      'Loading initial ramdisk ...'
initrd    /boot/initrd.img-4.19.0-8-amd64

Minimum Emacs-like screen editing is supported. TAB lists
completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
command-line or ESC to discard edits and return to the GRUB
menu.
```

Move the cursor to the end of this line, just after `ro quiet` and append the parameter `init=/bin/bash`.

Next hit `ctrl + x` to enable it to boot in single-user mode with the root filesystem mounted with read-only (ro) access rights.

For you to reset the password, you need to change the access right from read-only to read-write. Therefore, run the command below to remount the root filesystem with rw attributes.

```
:/# mount -n -o remount,rw /
```

Next, reset the root password by executing the good old `passwd` command as shown.

```
:/# passwd
```

Provide the new password and retype it to confirm. If all went well and the passwords match you should get a 'password updated successfully' notification at the end of the console

Finally press `Ctrl + Alt + Del` to exit and reboot. You can now log in as the root user using the newly created password that you just defined.

And that's how you reset a forgotten root password on Debian 10.