Установка и первичная настройка Arch Linux + Xfce, часть 1

Опубликовано 11.02.2013 в 00:33

Обновлено 07.10.2015 в 15:40

В этом цикле статей я хочу подробно показать как устанавливать и настраивать Arch Linux с оболочкой Xfce. Это будет полезно тем, кто хочет попробовать Arch Linux, но так и не решится взяться за это. Также эти статьи можно использовать как шпаргалки по установке в будущем.

Настройка дистрибутива будет ориентирована на обычное использование. Я хочу показать что нужно сделать, чтобы в будущем использовании системы было минимум проблем. Также я порекомендую те пакеты, которые я бы порекомендовал иметь в системе. Я буду ориентироваться на архитектуру x86_64, так как считаю её более современной. Но разницы, с точки зрения пользователя, вы практически не ощутите, так как все пакеты в Arch Linux поддерживают эту архитектуру.

Подготовка

Скачать последний срез дистрибутива можно на официальной странице загрузки:https://www.archlinux.org/download/.

После скачки нужно записать образ на внешнее устройство, такое как CD или USB Flash drive. Записать образ на CD можно программой **wodim** из набора **cdrkit**.

```
$ wodim -v dev=/dev/cdrw archlinux-2013.02.01-dual.iso
```

А записать образ на USB Flash drive можно программой **dd** следующим образом:

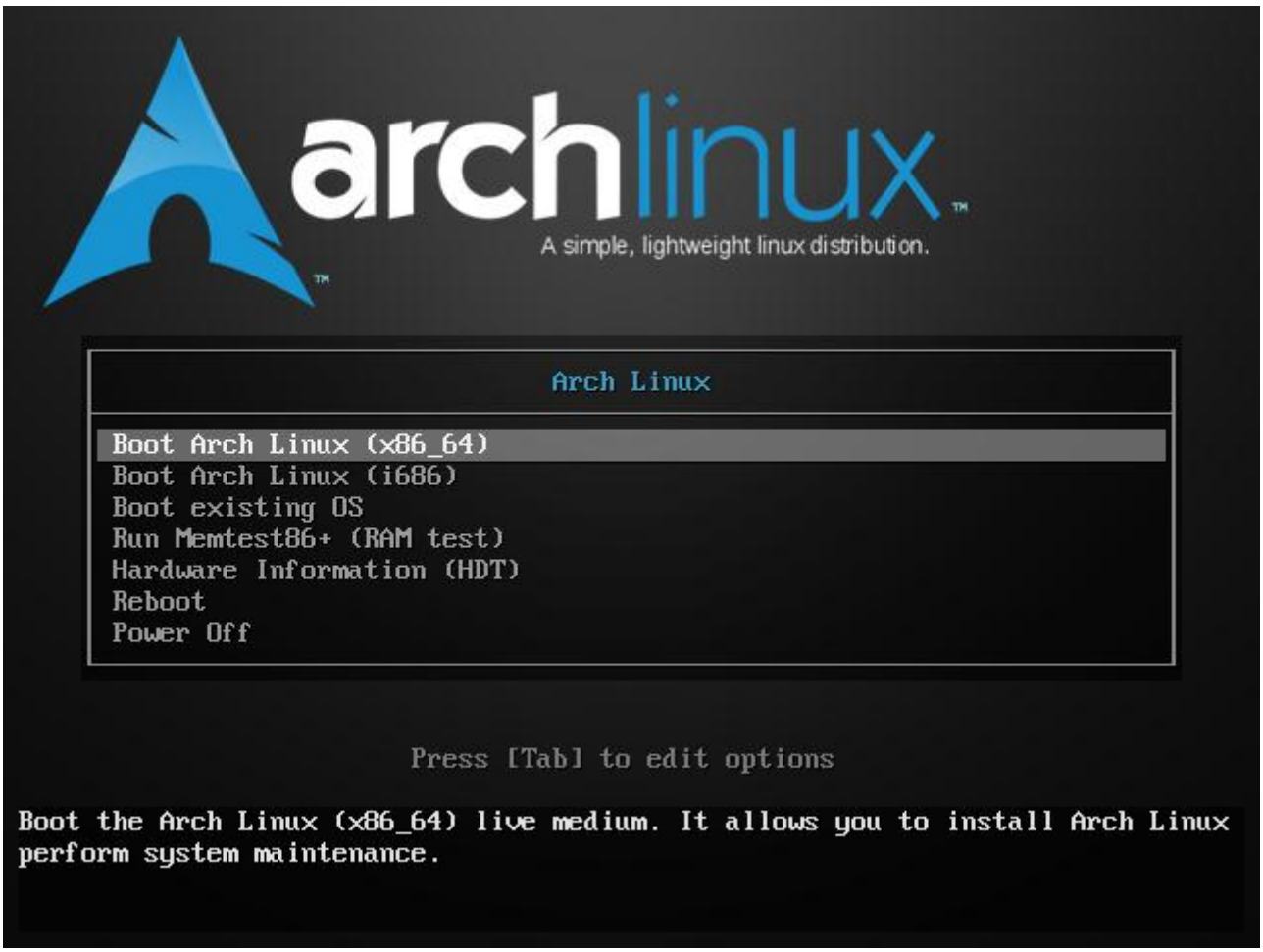
```
# dd bs=4M if=archlinux-2013.02.01-dual.iso of=/dev/sdx
```

где **/dev/sdx** — это USB накопитель, на который вы хотите записать образ(обычно **/dev/sdb**).

После записи образа на внешнее устройство вам необходимо настроить загрузку компьютера(в BIOS или EFI) в режим загрузки с вашего дисковода или USB Flash и запустить компьютер с подключенным устройством. Теперь можно приступить к установке.

Установка

Первым делом мы увидим меню загрузки



Выбираем архитектуру, я выбираю x86_64 и ждём пока система загрузится.

Русский язык

После загрузки в первую очередь настроим русский язык. Это делается следующим образом:

Установим русскую раскладку:

```
# loadkeys ru
```

Изменим консольный шрифт на тот, который поддерживает кириллицу:

```
# setfont cyr-sun16
```

Добавим русскую локаль в систему установки:

В файле **/etc/locale.gen** раскомментируйте (уберите # вначале) строку **#ru_RU.UTF-8 UTF-8**

```
# nano /etc/locale.gen
```

(закрыть файл: **Ctrl + X**)

Обновим текущую локаль системы:

```
# locale-gen
# export LANG=ru_RU.UTF-8
```

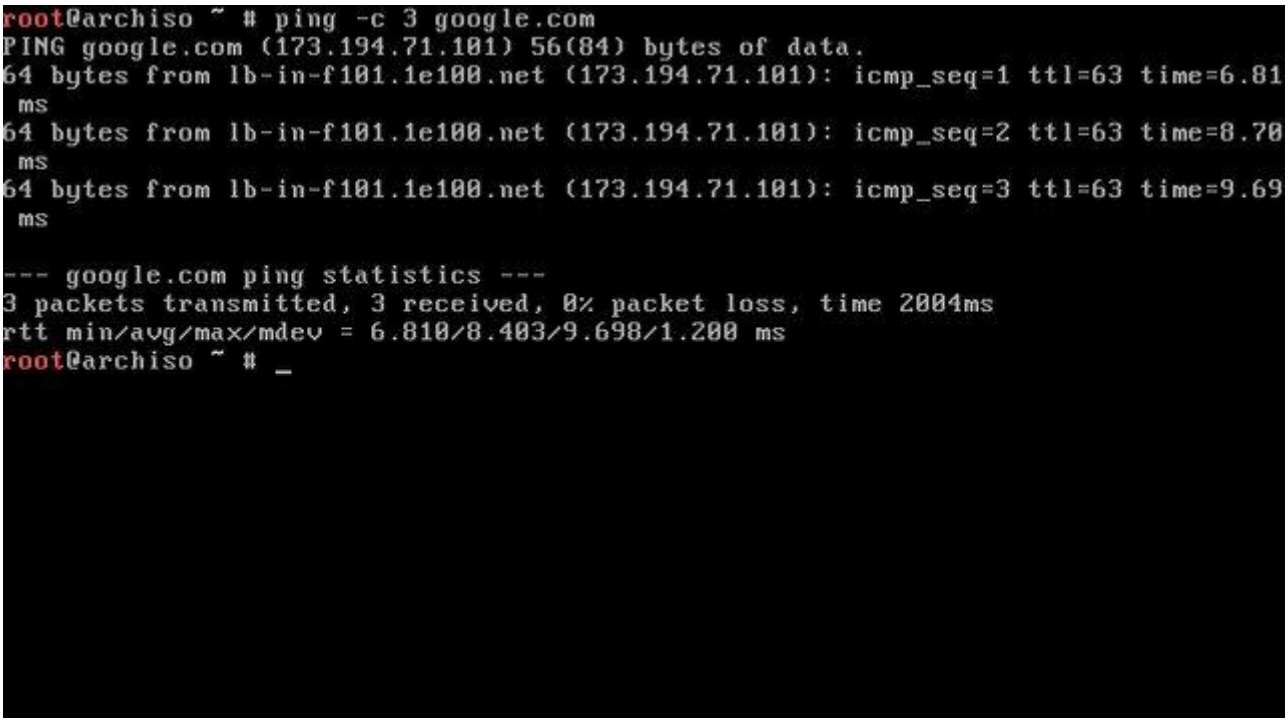
Настройка сети

Сейчас при установке дистрибутива наличие сети обязательно, поэтому её настройка необходима.

Проверить подключение к сети можно так:

```
ping -c 3 google.com
```

Результатом должно быть что-то вроде этого:



Если подключения нет, то приступим к настройке.

Для беспроводного подключения используйте программу **wifi-menu**

Примечание: *Если в результате выходит ошибка о не существовании wlan0, то узнайте как называется ваш сетевой интерфейс с помощью* ***iwconfig*** *и введите* ***wifi-menu <интерфейс>***

Для PPPoE: используйте для настройки программу **pppoe-setup**, для запуска — **pppoe-start**

Создание разделов на жестком диске

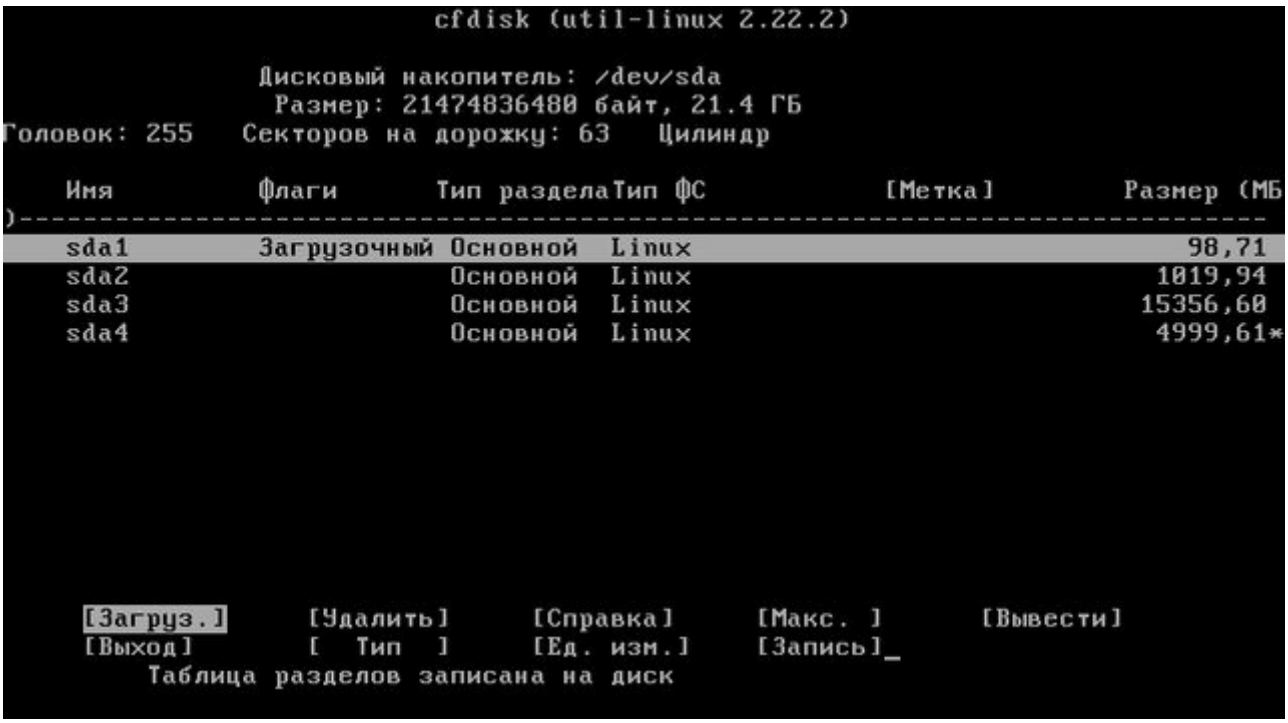
Для управления разделами на жестком диске в процессе установки рекомендую использовать программу**cfdisk**.

Рекомендую создать следующие разделы:

- 100 Мб с флагом Загрузочный(Boot) — для загрузчика.
- Раздел, на 1 Гб больше, чем объем оперативной памяти — для раздела подкачки(swap), если он вам нужен.
- 15 Гб (15360 Мб) для корневого раздела системы.
- И всё оставшееся для домашнего раздела.

Жмём кнопку Запись.

В итоге должно получиться что-то вроде этого:



Форматирование разделов

Для загрузочного раздело будем использовать файловую систему **ext2**. Также мы будем использовать метки для удобства.

```
# mkfs.ext2 /dev/sda1 -L boot
```

Для раздела подкачки (swap):

```
# mkswap /dev/sda2 -L swap
```

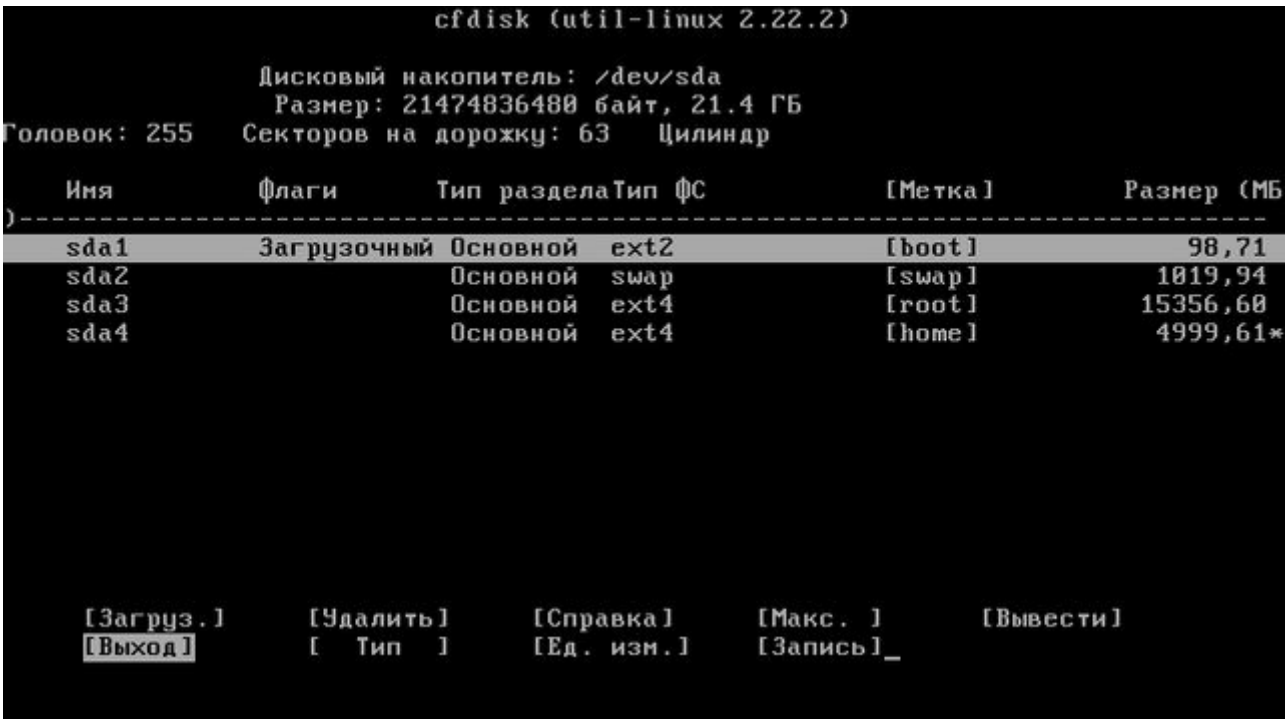
Для корневого раздела используем ext4:

```
# mkfs.ext4 /dev/sda3 -L root
```

Для домашнего раздела также используем ext4:

```
# mkfs.ext4 /dev/sda4 -L home
```

В итоге получим это(cfdisk):



Монтирование разделов

Смонтируем корневой раздел:

```
# mount /dev/sda3 /mnt
```

Создадим каталоги для монтирования boot и home разделов:

```
# mkdir /mnt/{boot,home}
```

Смонтируем загрузочный раздел:

```
# mount /dev/sda1 /mnt/boot
```

Смонтируем домашний раздел:

```
# mount /dev/sda4 /mnt/home
```

Подключим раздел подкачки(swap)

```
# swapon /dev/sda2
```

Выбор зеркал для pacman

Для более быстрой скачки пакетов нужно настроить зеркала. Поставим российское зеркало выше всех остальных:

```
# nano /etc/pacman.d/mirrorlist
```

Впишите эту строку вверх:

```
Server = http://mirror.yandex.ru/archlinux/$repo/os/$arch
```

Для закрытия нажмите **Ctrl + X** и согласитесь на изменения.

Установка пакетов

Установим базовые пакеты системы. В этот список вы можете добавить какие-нибудь нужные вам пакеты для установки.

```
# pacstrap -i /mnt base base-devel
```

После выполнения команды и выбора всех пакетов, скачаются(около 150 Мб) и установятся(около 130) последние версии необходимых пакетов.

***Важно:** если вы ведёте установку по **Wi-Fi**, то вам необходимо установить пакеты netctl, dialog и wpa_supplicant:*

```
# pacstrap -i /mnt netctl dialog wpa_supplicant
```

Установка пакета GRUB в устанавливаемую систему

```
# arch-chroot /mnt pacman -S grub
```

Если вы используете EFI, то установите efibootmgr:

```
# arch-chroot /mnt pacman -S efibootmgr
```

Если вы устанавливаете 32-х битную систему, то используйте пакет **grub-efi-i386**.

Первичная настройка системы

Сгенерируем **fstab**, для этого используем следующую команду:

```
# genfstab -p /mnt >> /mnt/etc/fstab
```

Перейдём в установленную систему:

```
# arch-chroot /mnt /bin/bash
```

Добавим русскую локаль в систему:

В файле **/etc/locale.gen** раскомментируйте(уберите # вначале) строку **#en_US.UTF-8 UTF-8** и строку**#ru_RU.UTF-8 UTF-8**

```
# nano /etc/locale.gen
```

(закрыть файл: **Ctrl + X**)

Обновим текущую локаль системы:

```
# locale-gen
```

Добавим русскую локаль в консоль:

В **/etc/mkinitcpio.conf**, в разделе **HOOKS**, должен быть прописан хук **keymap**.

В разделе **MODULES** нужно прописать свой драйвер видеокарты: **i915** для Intel, **radeon** для AMD, **nouveau**для Nvidia. Пример, как это может выглядеть: http://pastebin.com/xknvDX33

Создадим загрузочный RAM диск:

```
# mkinitcpio -p linux
```

Установим загрузчик (для **BIOS**):

```
# grub-install /dev/sda
```

Установим загрузчик (для **EFI**):

```
# grub-install --target=x86_64-efi --efi-directory=/boot --bootloader-id=grub
```

Обновим grub.cfg:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

Внимание: *если при создании* ***grub.cfg*** *у вас были ошибки, то попробуйте добавить* ***GRUB_DISABLE_SUBMENU=y*** *в* ***/etc/default/grub****.*

Установим root пароль:

```
# passwd
```

Выйдем из установленной системы:

```
# exit
```

Отмонтируем ранее монтируемые разделы:

```
# umount /mnt/{boot,home,}
```

Сейчас следует перезагрузить систему.

```
# reboot
```

И нужно зайти в **root** пользователя с помощью ранее введённого пароля.

Изменим имя компьютера(замените **myhostname** на своё):

```
# hostnamectl set-hostname myhostname
```

Установим временную зону:

```
# timedatectl set-timezone Europe/Moscow
```

Локализуем систему:

```
# localectl set-keymap ru
# setfont cyr-sun16
# localectl set-locale LANG="ru_RU.UTF-8"
# export LANG=ru_RU.UTF-8
```

Добавим строку **FONT=cyr-sun16** в **/etc/vconsole.conf**.

Обновим загрузочный RAM диск:

```
# mkinitcpio -p linux
```

Обновим grub.cfg(для локализации):

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

Настроим pacman (только для **x86_64**):

```
# nano /etc/pacman.conf
```

Для работы 32-битных приложений в 64-битной системе нужно раскомментировать (удалить # вначале) репозиторий multilib:

```
#[multilib]
#Include = /etc/pacman.d/mirrorlist
```

Внимание: *если возникли какие-то проблемы при загрузке пакетов, попробуйте создать файл* ***/etc/sysctl.d/40-ipv6.conf*** *и записать в него это:* ***net.ipv6.conf.all.disable_ipv6 = 1***

Добавим пользователя (замените **myusername** на своё) и сразу поместим его в нужные группы:

```
# useradd -m -g users -G
audio,games,lp,optical,power,scanner,storage,video,wheel -s /bin/bash
myusername
```

Установим ему пароль:

```
# passwd myusername
```

Изменим ему информацию GECOS:

```
# chfn myusername
```

Настройка системы

Настройка сети

Для проводной сети

```
# systemctl enable dhcpcd
# systemctl start dhcpcd
```

Для беспроводной сети

```
# wifi-menu
```

Обновим базы данных пакетов:

```
# pacman -Syy
```

Обновим все пакеты:

```
# pacman -Su
```

Поставим и настроим sudo

```
# pacman -S sudo
```

В файле **/etc/sudoers** раскомментируем строку **# %wheel ALL=(ALL) ALL**

Теперь мы можем использовать **sudo** для выполнения команд администратора.

Теперь выйдите из **root** пользователя с помощью команды **exit** и зайдите в пользователя, которого вы создали.

Графическое окружение

Установим X:

```
sudo pacman -S xorg-server xorg-xinit xorg-server-utils mesa xterm
```

(если вы используете тачпад или тачскрин, то установите **xf86-input-synaptics**)

Установим драйвер для видеокарты:

Пакеты **lib32-*** нужно устанавливать только на **x86_64** системы.
Intel:

```
sudo pacman -S xf86-video-intel lib32-intel-dri
```

Nvidia:

```
sudo pacman -S xf86-video-nouveau lib32-nouveau-dri
```

AMD:

```
sudo pacman -S xf86-video-ati lib32-ati-dri
```

Если вы устанавливаете систему на виртуальную машину:

```
sudo pacman -S xf86-video-vesa
```

Xfce + SDDM

Приступим к установке и настройке графического окружения Xfce с менеджером входа SDDM:

```
sudo pacman -S xfce4 xfce4-goodies sddm
```

Добавим **sddm** в демоны:

```
sudo systemctl enable sddm.service
```

Шрифты

Рекомендую установить следующие шрифты:

```
sudo pacman -S ttf-liberation ttf-dejavu opendesktop-fonts ttf-bitstream-vera
ttf-arphic-ukai ttf-arphic-uming ttf-hanazono
```

Конец

Теперь можно перезагрузить систему командой

```
sudo systemctl reboot
```

Важно: *если у вас не вводятся символы в поле ввода, то попробуйте изменить раскладку с помощью комбинации клавиш:* ***левый shift + правый shift****. Чтобы установить английскую раскладку по умолчанию откройте файл* ***/etc/X11/xorg.conf.d/00-keyboard.conf*** *и в строке* ***Option «XkbLayout» «ru,us»*** *поменяйте* ***ru*** *и* ***us*** *местами.*

После перезагрузки и первом входе в систему, выберите пункт настроек по умолчанию, затем я советую зайти в **Меню приложений -> Настройки -> Внешний вид -> Шрифты** и выбрать любимый шрифт, я предпочитаю **Liberation Sans**, размера 10. После этого добавьте на панель элемент раскладки клавиатуры и настройте его(смените комбинацию клавиш, модель клавиатуры).

В следующих статьях я покажу как сделать работу с системой максимально удобно, путём тонкой настройки каждого элемента под себя.

Запуск при загрузке с помощью systemd на примере NTP и SSH

Поскольку наша система будет общаться с другими компьютерами, нам потребуется синхронизировать время. Если время на сервере и клиенте будет отличаться, то существует большая вероятность того, что они вообще не смогут соединиться друг с другом. В свою очередь sudo может начать просить пароль после каждого действия, думая, что таймаут авторизации давно истёк. И кто знает, с чем нам ещё предстоит столкнуться? Перестрахуемся.

Чтобы синхронизировать время с серверами через Интернет по протоколу NTP, нам нужно установить недостающие пакеты. Можно воспользоваться arch-root, но но мы обойдёмся ключами, которые сообщат новое место для установки менеджеру пакетов:

```
pacman --root $root --dbpath $root/var/lib/pacman -S ntp
```

Настроим получение точного времени с российских серверов:

```
mv $root/etc/ntp.conf $root/etc/ntp.conf.old && cat $root/etc/ntp.conf.old | sed 's/\(
[0-9]\).*\(.pool.ntp.org\)/\1.ru\2/g' | tee $root/etc/ntp.conf
```

Нам достаточно синхронизировать время один раз при загрузке. Раньше мы бы записали запуск службы точного времени в файл rc.local, но сейчас появился менеджер системы и служб systemd, который старается запускать службы (в оригинале они называются unit) параллельно для уменьшения времени загрузки системы. Естественно, что работоспособность одной службы может зависеть от функционирования другой. Например, нам бесполезно пытаться синхронизировать время через Интернет до того, как у нас на компьютере заработает сеть. Чтобы описать все эти взаимосвязи, уже недостаточно простого указания имени исполняемого файла, поэтому запуск посредством systemd стал весьма нетривиальным занятием. Для этой цели были созданы специальные файлы с расширением ".service". В них указаны зависимости, имена исполняемых файлов и другие параметры, которые нужно учитывать для успешного запуска. В частности, для управления этапами загрузки в systemd используются цели (target), которые по возлагаемым на них задачам схожи с уровнями запуска (runlevel). Подробности читайте в вики.

К радости новичков, вместе с пакетом ntp поставляется уже готовый ntpdate.service. Все файлы, описывающие запуск служб, находятся в папке $root/usr/lib/systemd/system/, и их можно открыть в любом текстовом редакторе или посмотреть обычным образом. Вот, например, $root/usr/lib/systemd/system/ntpdate.service:

```
[Unit]
Description=One-Shot Network Time Service
After=network.target nss-lookup.target
Before=ntpd.service
```

```
[Service]
Type=oneshot
PrivateTmp=true
ExecStart=/usr/bin/ntpd -q -n -g -u ntp:ntp

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

В блоке [Unit] в строке Description указывается краткое описание службы, и при каких условиях она должна быть запущена (в данном случае, после запуска сети, но до перед запуском сервера NTP, который мы вообще не планируем запускать). Запрос точного времени происходит единственный раз во время загрузки, и за это отвечает строка Type=oneshot из блока [Service]. В этом же блоке в строке ExecStart указаны действия, которые необходимо выполнить для запуска сервиса. В блоке [Install] в нашем случае указано, что запуск нашей службы необходим для достижения цели multi-user.target. Рекомендуется использовать такое же содержание блока [Install] для запуска самодельных служб.

В качестве первого практического примера мы немного расширим функциональность ntpdate.service, попросив его дополнительно исправлять время на аппаратных часах. Если после этого, на этом же самом компьютере вы загрузите Windows, то увидите время по Гринвичу, так что не пугайтесь.

Изменение стандартного поведения любой службы systemd производится следующим образом: сначала в папке /etc/systemd/system/ создается новый каталог с полным именем службы и расширением ".d", куда добавляется файл с произвольным именем и расширением ".conf", и уже там производятся нужные модификации. Приступим:

```
mkdir -p $root/etc/systemd/system/ntpdate.service.d && echo -e '[Service]\nExecStart=/usr/bin/hwclock -w' > $root/etc/systemd/system/ntpdate.service.d/hwclock.conf
```

Здесь просто говорится о том, что во сразу после запуска службы выполнить команду "/usr/bin/hwclock -w", которая переведёт аппаратные часы.

Добавляем службу ntpdate в автозагрузку (синтаксис стандартен для всех служб):

```
arch-chroot $root systemctl enable ntpdate
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpdate.service to /usr/lib/systemd/system/ntpdate.service.
```

Как видите, в каталоге multi-user.target.wants создалась обыкновенная символическая ссылка на файл ntpdate.service, а упоминание о цели multi-user.target мы видели в блоке [Install] этого самого файла. Получается для того, чтобы система достигла цели multi-user.target, должны быть запущены все службы из каталога multi-user.target.wants.

Теперь устанавливаем пакет SSH аналогичным способом (в ArchLinux он называется openssh):

```
pacman --root $root --dbpath $root/var/lib/pacman -S openssh
```

Но на этот раз для автозапуска мы будем использовать сокет, чтобы сервер SSH стартовал только после поступления запроса на подключение, а не висел мёртвым грузом в оперативной памяти:

```
arch-chroot $root systemctl enable sshd.socket
```

Мы не поменяли стандартный 22-й порт и не включили принудительное использование Protocol 2 — пусть это останется на моей совести.

Забегая вперед или знакомимся с обработчиками (hooks)

Чтобы мы могли не глядя подключиться к нашему будущему серверу, нам нужно знать его IP адрес. Будет намного проще, если этот адрес — статический. Обычные способы, о которых говорится в вики, нам не подходят. Проблема в том, что сетевые адаптеры в современном мире именуются согласно своему физическому расположению на материнской плате. Например, имя устройства enp0s3 означает, что это сетевой адаптер ethernet, который расположен на нулевой шине PCI в третьем слоте (подробности здесь). Сделано так для того, чтобы при замене одного адаптера другим, имя устройства в системе не поменялось. Такое поведение нам не желательно, т. к. на разных моделях материнских плат положение сетевой карты может быть разным, и когда мы попытаемся перенести наш загрузочный сервер из VirtualBox на реальное железо, нам скорее всего придётся загружаться с клавиатурой и монитором, чтобы правильно настроить сеть. Нам нужно, чтобы имя сетевого адаптера стало более предсказуемым, например, eth0 (это место зарезервировано смайликом).

Почему будем делать так?

Устанавливаем пакет mkinitcpio-nfs-utils, и у нас появится обработчик (hook) под названием «net»:

```
pacman --root $root --dbpath $root/var/lib/pacman -S mkinitcpio-nfs-utils
```

По-умолчанию, все файлы обработчика попадают в /usr/lib/initcpio/. Обычно это парные файлы с одинаковым названием, один из которых окажется в подкаталоге install, а другой — в hooks. Сами файлы являются обычными скриптами. Файл из папки hooks обычно попадает внутрь файла initramfs (позже мы о нём всё узнаем) и выполняется при загрузке системы. Второй файл из пары попадает в папку install. Внутри него есть функция build(), в которой находятся сведения о том, какие действия нужно выполнить во время генерации файла initramfs, а также функция help() с описанием того, для чего предназначен данный обработчик. Если запутались, то просто читайте дальше, и всё сказанное в этом абзаце встанет на свои места.

Папка initcpio также присутствует в каталоге /etc, и в ней тоже есть подкаталоги install и hooks. При этом она имеет безусловный приоритет над /usr/lib/initcpio, т. е. если в обеих папках окажутся файлы с одинаковыми названиями, то при генерации

initcpio будут использоваться файлы из /etc/initcpio, а не из /usr/lib/initcpio.

Нам нужно немного поменять функциональность обработчика net, поэтому просто скопируем файлы из /usr/lib/initcpio в /etc/initcpio:

```
cp $root/usr/lib/initcpio/hooks/net $root/etc/initcpio/hooks/ && cp $root/usr/lib/init
cpio/install/net $root/etc/initcpio/install/
```

Приводим файл hooks/net к следующему виду:

```
cat $root/etc/initcpio/hooks/net
```

```
# vim: set ft=sh:
run_hook() {
    if [ -n "$ip" ]
    then
        ipconfig "ip=${ip}"
    fi
}

# vim: set ft=sh ts=4 sw=4 et:
```

Теперь откроем файл $root/etc/initcpio/install/net и увидим, что в функции help() отлично написано, что из себя должна представлять переменная «ip»:

```
ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>
```

Останется просто установить значение переменной, чтобы задать статический IP адрес и название сетевого устройства, например так «192.168.1.100::192.168.1.1:255.255.255.0::eth0:none» (здесь и далее используйте подходящие для себя настройки сети). В следующем разделе вы узнаете, где именно задаётся значение.

А пока уберём всё лишнее из файла $root/etc/initcpio/install/net. Оставляем загрузку модулей сетевых устройств, программу ipconfig, которую использовали выше, и, естественно, сам скрипт из папки hooks, выполняющий всю основную работу. Получится примерно следующее:

```
cat $root/etc/initcpio/install/net
```

```
#!/bin/bash

build() {
    add_checked_modules '/drivers/net/'
```

```
    add_binary "/usr/lib/initcpio/ipconfig" "/bin/ipconfig"

    add_runscript
}

help() {
    cat <<HELPEOF
This hook loads necessary modules for a network device.
Manually configures network and freezes network device name.
HELPEOF
}

# vim: set ft=sh ts=4 sw=4 et:
```

Когда во время загрузки менеджер устройств systemd-udevd попробует переименовать наше сетевое устройство в привычное ему predictable network interface name, например, в enp0s3, то у него ничего не получится. Почему — читайте дальше.

Как происходит загрузка системы

Для простоты рассмотрим обычные BIOS. После включения и инициализации, BIOS начинает по порядку идти по списку загрузочных устройств, пока не найдет загрузчик, которому передаст дальнейшее управление загрузкой.

Как раз такой загрузчик мы записали в MBR нашего накопителя. Мы использовали GRUB, в настройках которого (файл grub.cfg) указали, что корневой раздел находится на диске с меткой HABR. Вот эта строка целиком:

```
linux   /boot/vmlinuz-linux root=LABEL=HABR rw  quiet
```

Здесь упомянут файл vmlinuz-linux, который является ядром системы, а указатель на корневую систему является его параметром. Мы просим искать корневую систему на устройстве с меткой HABR. Здесь также мог бы быть уникальный для каждого накопителя UUID, но в этом случае при переносе системы на другой диск нам несомненно пришлось бы его изменить. Если бы мы указали положение корневой системы привычным для линуксоидов образом: /dev/sda1, то не смогли бы загрузиться с USB накопителя, т. к. это имя USB накопитель бы получил только будучи единственным накопителем в компьютере. Маловероятно, что в компьютере окажется ещё один накопитель с меткой HABR, но не стоит об этом забывать.

Здесь же устанавливается значение глобальной переменной «ip» для нашего обработчика «net» (не забудьте поменять адреса на используемые в вашей сети):

```
linux   /boot/vmlinuz-linux root=LABEL=HABR rw  quiet ip=192.168.1.100::192.168.1.1:25
5.255.255.0::eth0:none
```

В соседней строке есть упоминание файла initramfs, с которым я обещал разобраться:

```
initrd  /boot/initramfs-linux.img
```

Далее при загрузке происходит следующее: загрузчик GRUB получает файлы vmlinuz и initramfs, сообщает им, где искать корневую файловую систему и передаёт им управление дальнейшей загрузкой.

Название initramfs образовано от initial ram file system. Это на самом деле обычная корневая файловая система Linux, упакованная в архив. Она разворачивается в оперативной памяти во время загрузки и предназначена для того, чтобы найти и подготовить корневую файловую систему нашего linux, который мы пытаемся загрузить в итоге. В initramfs есть всё необходимое для этих целей, ведь это настоящий «маленький линукс», который может выполнять многие обычные команды. Его возможности расширяются с помощью обработчиков (hooks), которые помогают сформировать новую корневую файловую систему нашего linux.

После того, как программы из initramfs выполнят свою работу, управление дальнейшей загрузкой передается процессу init подготовленной корневой файловой системы. В качестве процесса init Archlinux использует systemd.

Менеджер устройств systemd-udevd является частью systemd. Он, как и его старший брат, старается обнаруживать и настраивать все устройства в системе параллельно. Он начинает свою работу одним из первых, но уже после того, как наш обработчик net инициализирует сетевую карту ещё на на этапе работы initramfs. Таким образом, systemd-udevd не может переименовать используемое устройство, и имя eth0 сохраняется за сетевой картой в течение всего времени работы.

Готовим initramfs

Для создания файла initramfs используется программа mkinitcpio, которая входит в пакет base, установленный нами в самом начале. Настройки находятся в файле $root/etc/mkinitcpio.conf, а пресеты лежат в папке /etc/mkinitcpio.d. От нас требуется сделать initramfs таким, чтобы он смог найти и подготовить корневую файловую систему, с которой впоследствии начнёт работать systemd. Нам совершенно необязательно учитывать все возможные варианты, достаточно только самого необходимого, чтобы не увеличивать размеры файла initramfs. Более подробная информация находится здесь wiki.archlinux.org/index.php/Mkinitcpio

Обязательно убираем обработчик autodetect. Он проверяет устройства установленные в данном конкретном компьютере, и оставляет только необходимые для них модули в initramfs. Нам этого не нужно, поскольку мы изначально рассматриваем возможность дальнейшего переноса системы на другой компьютер, который аппаратно скорее всего будет отличаться от используемой виртуальной машины.

Достаточный для наших целей список обработчиков включая созданный нами net выглядит следующим образом:

```
HOOKS="base udev net block filesystems"
```

вставляем эту строку в файл mkinitcpio.conf, а старую комментируем:

```
nano $root/etc/mkinitcpio.conf
```

На базе стандартного пресета linux создаем свой пресет habr:

```
cp $root/etc/mkinitcpio.d/linux.preset $root/etc/mkinitcpio.d/habr.preset
```

И приводим его к такому виду:

```
cat $root/etc/mkinitcpio.d/habr.preset
```

```
ALL_config="/etc/mkinitcpio.conf"
ALL_kver="/boot/vmlinuz-linux"

PRESETS=('default' )

default_image="/boot/initramfs-linux.img"
```

Нам не нужна ветка 'fallback', которая удаляет из обработчиков autodetect, ведь мы его уже сами убрали, и нам не нужно дважды генерировать одинаковый файл initramfs с разными названиями.

Генерируем новый initramfs с помощью пресета habr:

```
arch-chroot $root mkinitcpio -p habr
```

Пишем службу обновления DNS для использования с systemd

Наша сетевая карта получает все настройки для того, чтобы работала сеть и Интернет. Но названия сайтов переводиться в IP адреса не будут, т. к. наша система не знает, какие серверы DNS следует для этого использовать. Напишем собственную службу для этих целей, которую при загрузке будет запускать systemd. А чтобы узнать что-то новое и не заскучать от однообразия, передадим информацию о названии сетевого устройства в качестве параметра, а список DNS серверов сохраним во внешнем файле.

Обновлением информации о DNS серверах занимается resolvconf. Нам идеально подходит синтаксис:

```
resolvconf [-m metric] [-p] -a interface <file
```

В импортируемом здесь файле IP адрес каждого сервера указывается в новой строке после ключевого слова nameserver. Можно указать сколько угодно серверов, но использоваться будут только первые 3 из них. В качестве примера воспользуемся серверами Яндекс. В этом случае файл, передаваемый в resolvconf, должен выглядеть вот так:

```
nameserver 77.88.8.8
nameserver 77.88.8.1
```

Нам нужно получать информацию о DNS серверах до того, как система будет уверена, что сеть полностью работает, т. е. до достижения цели network.target. Будем считать, что информацию о серверах нам достаточно обновлять один раз во время загрузки. И стандартно скажем, что нашу службу требует цель multi-user.target. Создаём файл запуска службы в каталоге со следующим содержанием:

```
cat $root/etc/systemd/system/update_dns@.service
```

```
[Unit]
Description=Manual resolvconf update (%i)
Before=network.target

[Service]
Type=oneshot
EnvironmentFile=/etc/default/dns@%i
ExecStart=/usr/bin/sh -c 'echo -e "nameserver ${DNS0}\nnameserver ${DNS1}" | resolvcon
f -a %i'

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

В строке ExecStart мы выполняем команду echo, на лету генерирующую файл со списком серверов, который через конвейер передаем resolvconf. Вообще, в строке ExecStart нельзя использовать несколько команд и тем более нельзя использовать конвейеры, но мы снова всех обманули, передав эти команды в качестве параметра -c для /usr/bin/sh.

Обратите внимание, что в названии файла update_dns@.service используется символ @, после которого можно указать переменную, и она попадёт внутрь файла, заменив собой "%i". Таким образом строка EnvironmentFile=/etc/default/dns@%i превратится в EnvironmentFile=/etc/default/dns@eth0 — именно это название внешнего файла, мы будем использовать для хранения значения переменных DNS0 и DNS1. Синтаксис как в обычных скриптах: «название переменной=значение переменной». Создадим файл:

```
nano $root/etc/default/dns@eth0
```

И добавим следующие строки:

```
DNS0=77.88.8.8
DNS1=77.88.8.1
```

Теперь добавляем службу в автозагрузку не забывая указать имя сетевой карты после @:

```
arch-chroot $root systemctl enable update_dns@eth0.service
```

Только что мы написали универсальный файл, обеспечивающий запуск службы. Универсальность заключается в том что, если в нашей системе окажется несколько сетевых адаптеров, то для каждого из них мы сможем указать свои собственные DNS серверы. Нужно будет просто подготовить набор файлов со списком серверов для каждого из устройств и запускать службу для каждого адаптера в отдельности указывая его имя после @.