GentoObox

simple manual for minimalistic Gentoo installation with Openbox environment

September 27, 2017

Contents

1 Introduction

Thanks to my father, I am an Gentoo user since high school. Although 10 years have passed from my first installation, there is little difference with my last. I always go to the Gentoo handbook, always google additional information, always have new ideas to achieve, and always spend a week or two configuring the system to suit my needs. I cannot say that my needs are so special - I need a browser, video player, unified look and feel in the GUI, little programming tools. I like when things work as smooth and fast as possible, and of course as reliable as possible.

My initial choice of environment was Gnome2, but after the changes introduced in Gnome3 I was on a crossroad. I never liked how KDE looks and feels, so the next logical choice was Xfce. It truly was fast and easy to use, but somehow did not feel right and my search continued. Governed by the idea that everything should be as fast and light as possible I started experimenting with windows managers. From all of the ones out there Openbox got my heart.

Of course choosing to use window manager has its benefits and drawbacks. The pros are small dependencies, lightweight, fast, reliability, freedom to choose what you want to use. The cons are how much work and effort you have to put into building your system, but isn't this the Gentoo way after all?

Unfortunately there are not many guides how to build your system. Some honourable mentions are Urukrama openbox guide, Gentoo Openbox Wiki, Arch Openbox Wiki, but they do not contain all the answers. For example it took me embarrassingly long time to understand what is a dock, example plank, system tray and how they do not mix in all cases. Or how to create dual head configuration using ZaphodHeads method in xorg. This document is dedicated to all of those problems I faced and put numerous hours to resolve.

2 Basic Gentoo installation

2.1 Preparing the disks

This section is dedicated on the basic Gentoo installation. It follows for the most part Gentoo AMD64 Handbook and will include additional details.

I usually use some live distro for the installation and initial configuration, and thus presume this position for the rest of the text. Without further ado the first step, of course is preparing the disk. For the moment I have not met any need for GPT or UEFI and continue using MRB with BIOS. Almost all of the live distros I have encountered include *Gparted*, so I use it for the partitionin.

2.1.1 MBR with BIOS

Using *Gparted*

1. you have chosen the correct device, */dev/sdb*[1]
2. go to *Device -¿ Create Partition table* and make sure it is **MBR**

For all of my purposes I need no more than 150 GB so sufficient configuration for me is something like:

- **swap** - swap - as much as the physical RAM on the machine
- **boot partition** - ext4 - 512 MB - boot partition, for historical reasons and structural benefits, I have found article which discusses the differences between /boot, bootloader partition in greater detail, but at the time of writing I did not manage to find it again
- **root partition** - ext4 - 60 GB - the partition for root /
- **home partition** - ext4 - 120 GB - the partition for /home, reasons see above link

After the partitioning is done the live distro usually automaticly activates the swap, but if necessary you can do it by hand with.

```
root #  swapon /dev/sdb{swap partition number}
```

[1] I am presuming we are working from live usb distro, so usually **/dev/sda** is the usb drive, and **/dev/sdb** is the disk we are working on.

2.2 Installing a stage tarball

Now lets mount the root partition.

```
root #  mkdir /mnt/gentoo && \
     mount /dev/sdb{root partition number} /mnt/gentoo
```

Make sure the date is correct

```
root #  date
Sun Sep 24 10:58:24 EEST 2017
```

If it is not, set it using the format *MMDDhhmmYYYY*, so to set it as the above

```
root #  date 092410582017 && date
Sun Sep 24 10:58:24 EEST 2017
```

Navigate to Gentoo download section and choose **Stage 3** for your architecture, mine is *amd64*, and place the tarball in **/mnt/gentoo**, and unpack it

```
root #  cd /mnt/gentoo && \
curl -O http://distfiles.gentoo.org/releases/amd64/{path to file} && \
tar xvjpf stage3-*.tar.bz2 --xattrs --numeric-owner
```

2.3 Initial Gentoo configuring

2.3.1 Pre requires

Now we will **chroot** into */mnt/gentoo* and configure everything within there. For this to happen we need to

1. copy dns info
2. mount our partitions
3. mount the necessary filesystems

1. to ensure that networking still works even after entering the new environment. This is done by

```
root #  cp -L /etc/resolv.conf /mnt/gentoo/etc/
```

2. mount our partitions

   Firstly lets mount them

```
root #  mount /dev/sdb{boot partition number} /mnt/gentoo/boot
root #  mount /dev/sdb{home partition number} /mnt/gentoo/home
```

   Lets add them to **/etc/fstab**, which should look something like this

   sample_configs/fstab_example

```
/dev/sda6 /boot   ext4  defaults,noatime  0 2
/dev/sda5 none    swap  sw      0 0
/dev/sda7 /   ext4  noatime     0 1
/dev/sda8 /home   ext4  noatime     0 2
/dev/cdrom  /mnt/cdrom  auto  noauto,ro   0 0
```

   Quick reference by field, for more details for example see here or here

   (1) device/partition, can be specified with label, network id, device path
   (2) mount point
   (3) filesystem type
   (4) mount options, some of the used are:
       - defaults: Uses the default options that are rw, suid, dev, exec, auto, nouser, and async
       - noatime: fully disables writing file access times to the drive every time you read a file. This works well for almost all applications, except for those that need to know if a file has been read since the last time it was modified.
       - auto: noauto
   (5) dump field: single digit, sets whether the backup utility dump will backup filesystem, set to 0 to ignore or 1 to back up
   (6) pass field: single digit, **fsck** order to check the filesystem at boot
       - 0 means do not check recommended for network shares
       - 1 check this partition first, highly recommended only for root partition
       - 2 check this partition next, all partitions marked with it are checked in sequence and we do not need to specify an order

3. mount the necessary filesystems, again I presume using non-Gentoo installation media

```
root #  mount -t proc /proc /mnt/gentoo/proc
root #  mount --rbind /sys /mnt/gentoo/sys
root #  mount --make-rslave /mnt/gentoo/sys
root #  mount --rbind /dev /mnt/gentoo/dev
root #  mount --make-rslave /mnt/gentoo/dev
root #  test -L /dev/shm && rm /dev/shm && mkdir /dev/shm
root #  mount -t tmpfs -o nosuid,nodev,noexec shm /dev/shm
root #  chmod 1777 /dev/shm
```

2.3.2 Chroot at /mnt/gentoo

We need to enter the new installation environment by chrooting into it:

```
root #  chroot /mnt/gentoo /bin/bash
root #  source /etc/profile
root #  export PS1="CH${PS1}"
```

Which should result into following line, thus we will recognize if we are at the live medium or into the changed root of Gentoo.

```
CHroot #
```

2.4 Configure portage

2.4.1 Repository

First we need to configure the repository, if it does not exist.

```
CHroot # mkdir -p /etc/portage/repos.conf
CHroot # touch /etc/portage/repos.conf/gentoo.conf && nano /etc/portage/repos
```

It should look like this:

sample_configs/gentoo.conf_example

```
[gentoo]
location = /usr/portage
sync-type = rsync
sync-uri = rsync://rsync.gentoo.org/gentoo-portage
auto-sync = yes
```

Now lets fetch the latest snapshot, up to the last hour. The command might complain about some missing locations, but it is safe to ignore.

```
CHroot # emerge-webrsync && emerge --sync
```

2.4.2 Profile selection

Now lets choose our profile, each one comes with predefined use flags (if you are not sure come here)

```
CHroot # eselect profile list
Available profile symlink targets:
[1]   default/linux/amd64/13.0
[2]   default/linux/amd64/13.0/selinux
[3]   default/linux/amd64/13.0/desktop
[4]   default/linux/amd64/13.0/desktop/gnome
[5]   default/linux/amd64/13.0/desktop/gnome/systemd
[6]   default/linux/amd64/13.0/desktop/plasma
[7]   default/linux/amd64/13.0/desktop/plasma/systemd
[8]   default/linux/amd64/13.0/developer
[9]   default/linux/amd64/13.0/no-multilib
[10]  default/linux/amd64/13.0/systemd
[11]  default/linux/amd64/13.0/x32
[12]  hardened/linux/amd64
[13]  hardened/linux/amd64/selinux
[14]  hardened/linux/amd64/no-multilib
[15]  hardened/linux/amd64/no-multilib/selinux
[16]  hardened/linux/amd64/x32
[17]  hardened/linux/musl/amd64
[18]  hardened/linux/musl/amd64/x32
[19]  default/linux/uclibc/amd64
[20]  hardened/linux/uclibc/amd64
```

For my purposes I need *default/linux/amd64/13.0/desktop*

```
CHroot # eselect profile set 3
```

2.4.3 /etc/portage/make.conf

The **/etc/portage/make.conf** is used to define settings and options applied to every package that is emerged. In the following lines I will explain in detail my configs. More details click here

- **CFLAGS**, **CXXFLAGS** are variables who define the build and compile flags that will be used for all package deployments. CFLAGS are for C based applications, CXXFLAGS are for C++ based ones.

  sample_configs/make.conf_example

  ```
  CFLAGS="-march=native -O2 -pipe"
  CXXFLAGS="${CFLAGS}"
  ```

  - o **-march=native** option: If the type of CPU is undetermined, or if the user does not know what setting to choose, it is possible use the *-march=native* settings. When this flag is used, GCC will attempt to detect the processor and automatically set appropriate flags for it. *This should not be used when intending to compile packages for different CPUs!*
    You can find the kind of CPU you have by using

    ```
    user $  cat /proc/cpuinfo
    ```

    If you are interested in what flags a specific option, lets say **core2**, will activate check by

    ```
    user $  gcc -c -Q -march=core2 --help=target
    ```

    If you are interested how the flags of different options will differ check with:

    ```
    user $  diff \
              <(gcc -march=native -Q --help=target) \
              <(gcc -march=core2 -Q --help=target)
    ```

  - o **-O2** option: This variable controls the overall level of optimization. -O0 will turn it off, -O1 will do most basic. -O2 is a step up from -O1 and the recommended level. -O3 is the highest level, but does not guarantee to improve performance and in some cases can slow down system due to large binaries and increased memory usage.
  - o **-pipe** option: Is a common flag which makes compilation process much faster.

- **CHOST** variable is passed through the configure step of ebuilds to set the build-host of the system. Note that the Gentoo profile already sets the appropriate CHOST value, and updating it requires insight and experience in build chains.

  sample_configs/make.conf_example

  ```
  CHOST="x86_64-pc-linux-gnu"
  ```

- **ACCEPT_KEYWORDS** defines globally, for all packages, on one hand the architecture, in our case it is **amd64**, since we have x86_64, but it could have been **arm**; and if we are to use stable or unstable ∼ packages. In our case we want only stable packages for x86_64. more detail on accept_keywords

  sample_configs/make.conf_example

  ```
  ACCEPT_KEYWORDS="amd64"
  ```

- **MAKEOPTS** specify arguments passed to *make* when packages are built from source. More info here [2] [3] [4] [5]

  sample_configs/make.conf_example

  ```
  MAKEOPTS="--jobs=4 --load-average=3.8"
  ```

[2] more detail on load average link1
[3] more detail on load average link2
[4] more detail on load average link3
[5] more detail on load average link4

- o –**jobs=4** defines how many parallel sessions to trigger, if they are possible. The recommended value is the number of logical processors in the CPU. You can obtain that number with the following command

```
user $  nproc --all
```

- o –**load-average=3.8** option: This option prevents starting new installations if the load-average is more than 3.8. It is recommend your load-average to be as much as the number of logical CPU.

• **EMERGE_DEFAULT_OPTS** specify arguments passed to *emerge*. More info here See previous footnotes for details on load average and parallelism. [6] [7] [8] [9]

sample_configs/make.conf_example

```
EMERGE_DEFAULT_OPTS="--jobs=4 --load-average=3.8 --with-bdeps y --quiet-
    build y --keep-going --autounmask-write y"
```

- o –**jobs=4** sets the amount of parallel packages to emerge. Note that if you have set **makeopts -j N** and **emerge_default_opts -j K** you will end up with $N * K$ tasks! The recommended value is the number of logical processors in the CPU.
- o –**load-average=3.8** prevents starting new instance of emerge if the load-average is more than 3.8. Rule of thumb is set it as much as the number of logical CPU.
- o –**with-bdeps y** option: By default, the dependency graph may not include some packages. If you would like to include such build time dependencies even though they are not strictly required. more detail on emerge_default_opts with-bdeps option
- o –**quiet-build y** option: Redirect all build output to logs alone, and do not display it on stdout. If a build failure occurs for a single package, the build log will be automatically displayed on stdout (unless the –quiet-fail option is enabled)
- o –**keep-going** option: Continue as much as possible after an error. When an error occurs, dependencies are recalculated for remaining packages and any with unsatisfied dependencies are automatically dropped.
- o –**autounmask-write y** option: If –autounmask is enabled, changes are written to config files, respecting CONFIG_PROTECT and –ask. If the corresponding package.* is a file, the changes are appended to it, if it is a directory, changes are written to the lexicographically last file.

• **FEATURES** variable specifies options which affect how Portage operates and how packages are compiled. It has some predefined options, depending on what profile has been set, but this is an incremental variable, thus values can be added without directly overriding the default ones. Man page of emerge all *Feature* variables with explanation

sample_configs/make.conf_example

```
FEATURES="parallel-install multilib-strict candy"
```

- o **parallel-install** option: Use finer-grained locks when installing packages, allowing for greater parallelization. For additional parallelization.
- o **multilib-strict** option: Many Makefiles assume that their libraries should go to /usr/lib, or $(prefix)/lib. This assumption can cause a serious mess if /usr/lib isn't a symlink to /usr/lib64. To find the bad packages, we have a portage feature called multilib-strict. It will prevent emerge from putting 64bit libraries into anything other than (/usr)/lib64.
- o **candy** Enable a special progress indicator when emerge calculates dependencies.

• **AUTOCLEAN** enables portage to automatically clean out older or overlapping packages from the system after every successful merge. This is the same as running 'emerge -c' after every merge. Set with: "yes" or "no", recommended "yes" since can cause serious problems due overlapping packages.

[6] See footnote 2
[7] See footnote 3
[8] See footnote 4
[9] See footnote 5

sample_configs/make.conf_example

```
AUTOCLEAN="yes"
```

- **CPU_FLAGS_X86** enables specific instructions for the architecture/cpu. More info

sample_configs/make.conf_example

```
CPU_FLAGS_X86="aes avx avx2 fma3 mmx mmxext pclmul popcnt sse sse2 sse3
   sse4_1 sse4_2 ssse3"
```

The easiest way to obtain the flags is by using *cpuinfo2cpuflags-x86*

```
CHroot # emerge -v app-portage/cpuid2cpuflags && \
    cpuinfo2cpuflags-x86 >> /etc/portage/make.conf
```

- **PYTHON_TARGETS** controls support for various Python implementations in packages. More info

sample_configs/make.conf_example

```
PYTHON_TARGETS="python2_7 python3_4"
```

- **VIDEO_CARDS** sets the video drivers that you intend to use. Will talk about kernel configs later on. More info

sample_configs/make.conf_example

```
VIDEO_CARDS="intel i965"
```

- **INTEL_MODESETTING** sets the usage of GLAMOR to accelerate 2D graphical over Mesa. It is intel specific thing, since *Intel DDX driver* has been slowly deprecating, it is recommend to use *modesetting DDX driver*. More info

sample_configs/make.conf_example

```
INTEL_MODESETTING="glamor"
```

- **INPUT_DEVICES** sets drivers for input devices. Kernel configs are later. evdev is responsible for keyboards, mice, joysticks, etc.... synaptics is responsible for touchpads

sample_configs/make.conf_example

```
INPUT_DEVICES="synaptics evdev"
```

- **L10N, LINGUAS** are localization variables. *L10N* will replace the *LINGUAS* in future, but for compatibility reasons should set them both for the moment. More info for LINGUAS and More info for L10N and More info about the replacement

sample_configs/make.conf_example

```
L10N="bg en en-GB"
LINGUAS="bg en en-GB"
```

- **SOUND** variable is to define use-flags related to sound. Its only entry is the necessary use flag for *PulseAudio* - sound server that provides a number of features on top of ALSA(most importantly - youtube with sound) Kernel configs are for further.

sample_configs/make.conf_example

```
SOUND="pulseaudio"
```

- **NETWORK** variable is to define use-flags related to network. Its only entry is the necessary use flag for NetworkManager. More on kernel config later on.

sample_configs/make.conf_example

```
NETWORK="networkmanager"
```

- **FONTS** variable is to define use-flags related to fonts config. they are recommended from here, although I am not following the full article any more. More info later on.

sample_configs/make.conf_example

```
FONTS="truetype type1 cleartype corefonts"
```

- **GUI** variable to define use-flags related to GUI - such as gtk, qt and similar. In general I prefer qt over gtk, but above all I want all of them to look alike. Since my last installation I noticed that including both *qt4* and *qt5* I could not achieve that, but more details later.

sample_configs/make.conf_example

```
GUI="qt5 -qt4"
```

- **MISC** misc use flags

sample_configs/make.conf_example

```
MISC="hdaps xinerama"
```

- **USE** use variable is the keyword, which use flags. Since it permits the usage of variables I did so and the final result is

sample_configs/make.conf_example

```
USE="${SOUND} ${NETWORK} ${FONTS} ${INTEL_MODESETTING} ${GUI} ${MISC}"
```

- **GENTOO_MIRRORS** is setting the mirrors.

sample_configs/make.conf_example

```
GENTOO_MIRRORS="http://tux.rainside.sk/gentoo/ ftp://ftp.uni-erlangen.de
    /pub/mirrors/gentoo http://ftp.halifax.rwth-aachen.de/gentoo/ http://
    ftp.twaren.net/Linux/Gentoo/ ftp://ftp.twaren.net/Linux/Gentoo/"
```

Let *mirrorselect* to do the job for you with the following command:

```
CHroot # emerge -v mirrorselect && \
    mirrorselect --servers 5
```

2.4.4 Configure timezone and locales

1. Timezone

   Available timezones are placed into */usr/share/zoneinfo/*, and you need to write them in the */etc/-timezone* file and then you should reconfigure sys-libs/timezone-data Lets say your capital is Sofia, so

```
CHroot # echo $(find /usr/share/zoneinfo/ -name "Sofia" \
         | cut -d"/" -f5-) \
         > /etc/timezone \
         && emerge -v sys-libs/timezone-data \
         && emerge --config sys-libs/timezone-data
```

2. Locals

   specify not only the language, but also what the rules are for sorting strings, displaying dates and times etc. Available combinations of locale are placed at */usr/share/i18n/SUPPORTED*, for my case I want to see combinations of *bg*, so

```
CHroot # cat /usr/share/i18n/SUPPORTED | grep bg
bg_BG.UTF-8 UTF-8
bg_BG CP1251
```

   Now to specify wanted locals we shall edit */etc/locale.gen*. Add the ones we want, if missing, or comment/uncomment depending on our needs.

   sample_configs/locale.gen_example

```
# For the default list of supported combinations, see the file:
# /usr/share/i18n/SUPPORTED
#
en_US ISO-8859-1
en_US.UTF-8 UTF-8
bg_BG CP1251
bg_BG.UTF-8 UTF-8
#ja_JP.EUC-JP EUC-JP
#ja_JP.UTF-8 UTF-8
```

   Last step is to generate the locals with

```
CHroot # locale-gen
```

   Finally lets set the system-wide locale. In near future will set link to a nice article about all those variables and why I chose *C*

```
CHroot # eselect locale list
Available targets for the LANG variable:
[1]   C
[2]   POSIX
[3]   bg_BG
[4]   bg_BG.cp1251
[5]   bg_BG.utf8
[6]   en_US
[7]   en_US.iso88591
[8]   en_US.utf8
CHroot # eselect locale set 1
```

2.5 Kernel configuration

Despite the method you choose run

```
CHroot # emerge -va sys-kernel/gentoo-sources sys-kernel/linux-firmware
```

2.5.1 Easy way - genkernel

Alternative to manual configuration is using genkernel,click the link for more info.

```
CHroot # emerge -va sys-kernel/gentoo-sources sys-kernel/linux-firmware
```