GentoObox

simple manual for minimalistic Gentoo installation with Openbox environment

September 25, 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.

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

3 rest of it

GentoObox installation additional notes September 25, 2017

1. Preparing the discs and stage tarball

   1.1. Since using live distro use pre-installed gparted to partition the disk.[10]

   1.2. Best partitioning for me for the moment is to use MBR, **not** UEFI or GPT:

   - swap - size the same as RAM - swap filesystem
   - /boot - size around 512MB - ext4 filesystem [11]
   - / - size around 60GB - ext4 filesystem
   - /home - size around 80GB - ext4 filesystem

   1.3. After partitioning:

   - Starting the swap

   ```
   root #  swapon /dev/sdb{partition num}
   ```

   [12]

   - Mounting the root partition

   ```
   root #  mkdir /mnt/gentoo
   ```

   ```
   root #  mount /dev/sdb{root partition num} /mnt/gentoo
   ```

   1.4. Preparing the stage tarball

   1.4.1. Navigate to here

   1.4.2. Download amd64 multilib[13][14]

   1.4.3. Move the stage tarball to your root partition and unpack it

   ```
   root #  mv {PATH}/stage3-*tar.bz2 /mnt/gentoo
   root #  tar xvjpf stage3-*.tar.bz2 --xattrs --numeric-owner
   ```

   - Make sure the same options are used!!!!
   - **x** stands for extract
   - **v** fore verbose
   - **j** decompress with bzip2
   - **p** preserve permissions
   - **f** denote that we want to extract a file, not standart input
   - –**xattrs** is to include the extended attributes stored in the archive
   - –**numeric-owner** assures the user and groups IDs of the files are extracted

[10] Alternative is to use **fdisk** to partition the disc as **fdisk /dev/sdb** and use **mkfs.ext4 /dev/sdb#** to create filesystem for partition number

[11] Why? Because historical and tutorial reasons, also I have some memory that it is necessary if you want to encrypt

[12] 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.

[13] Since I have x86_64 (64 bit processor) I will proceed with amd64 multilib. If you have 32 bit architecture do x86. If you know what you are doing do whatever you want.

[14] I presume you are using live distribution, if not - use **lynx** to navigate or **wget** to download direct file.

2. Configuring the base system

2.1. First lets make sure we will inherit the internet connection from the live usb

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

2.2. Now lets chamber root ourself

2.2.1. Prepare necessary filesystems

```
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
```

2.2.2. since I usually am using non-gentoo installation media

```
root #  rm /dev/shm && mkdir /dev/shm
root #  mount -t tmpfs -o nosuid,nodev,noexec shm /dev/shm
root #  chmod 1777 /dev/shm
```

2.2.3. Lets get inside it

```
root #  chroot /mnt/gentoo /bin/bash
root #  source /etc/profile
root #  export PS1="(chroot)␣$PS1"
```

[15]

2.2.4. Now we are in, lets not forget we are having a boot and home partition

2.2.4.1. Create and mount boot partition

```
root #  mkdir /boot && mount /dev/sdb{boot partition num} /boot
```

2.2.4.2. Mount home partition

```
root #  mount /dev/sdb{home partition num} /home
```

2.2.4.3. Lets assure those partitions will automatically mount during boot by editing the **/etc/fstab** file. It usually is pre written and should look something like this. Remember, for the Gentoo system itself the disk is **/dev/sda**!!!

sample_configs/fstab_example

```
# <fs>        <mountpoint>  <type>    <opts>     <dump/pass>
#
# NOTE: Even though we list ext4 as the type here, it will work
    with ext2/ext3
#       filesystems.  This just tells the kernel to use the
   ext4 driver.
#
```

2.3. Now lets update the portage, so we can install whatever we need. This happens by installing a snapshot of the main ebuild repository as old as 24h old. It may complain about missing files or similar, but it will fix them by himself.

```
root #  emerge-webrsync
```

2.4. Lets make sure we are using the top 5 fastest mirrors. For this we will emerge **mirrorselect** and let it do the job - it will edit the **/etc/portage/make.conf** file for us, continue reading for more info on that file.

```
root #  emerge -av mirrorselect
root #  mirrorselect --servers 5
```

[15] I usually do not do this step since I have an different terminal on the live usb. Also usually it changes its colour scheme so pretty easy to spot where is my gentoo system.

2.5. Gentoo is a distro, which compiles all of its packages, so we would like to optimize the process as much as possible. For this we will edit the **/etc/portage/make.conf** file[16]. I will go line by line and briefly explain the meaning:

- The **CFLAGS** and **CXXFLAGS** variables define the optimization flags for GCC C and C++ compilers respectively, and thus to every package. MUCH OF RECOMMENDATION: use the same settings for both variables, because reasons.[17]

sample_configs/make.conf_example

```
# These settings were set by the catalyst build script that
    automatically
# built this stage.
# Please consult /usr/share/portage/config/make.conf.example for a
    more
# detailed 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 setting. 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=
    ```

  - o **-O2** option: This variable controls the overall level of optimization. -O0 will turn of optimization, -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.

- The **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

```
# WARNING: Changing your CHOST is not something that should be
    done lightly.
# Please consult http://www.gentoo.org/doc/en/change-chost.xml
    before changing.

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.[18]

sample_configs/make.conf_example

```
# These are the USE flags that were used in addition to what is
    provided by the
```

[16] more detail notes on make.conf
[17] additional info about gcc optimization
[18] more detail on accept_keywords

```
# profile used for building.

ACCEPT_KEYWORDS="amd64"
```

- **MAKEOPTS** variable is used to specify arguments passed to **make** when packages are built from source.

  sample_configs/make.conf_example

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

  - o –**jobs=4** option: The parallel jobs entry ensures that, when make is invoked, it knows how many parallel sessions it is allowed to trigger (when parallel sessions are possible of course). This is completely within the scope of that **make** command and has no influence on parallel installation. The recommended value is the number of logical processors in the CPU. In my case I have i5 with 2 physical cores, but thanks to hyper-threading they are 4.[19]
  - 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, in my case with i5 2 physical, but thanks to hyper-threading - 4.[20]

- **EMERGE_DEFAULT_OPTS** holds entries which are appended to the emerge command line[21] [22]

  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** option: 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. In my case I have i5 with 2 physical cores, but thanks to hyper-threading they are 4.[23]
  - o –**load-average=3.8** option: This option prevents starting new instance of emerge 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, in my case with i5 2 physical, but thanks to hyper-threading - 4.[24].
  - 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.[25]
  - 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. The features I list here are set because curiosity, I would be happy to see more discussion about if they are helpful for a regular user. [26]

[19] more detail on load-average, makeopts, emerge and jobs
[20] more detail on load average or see footnote 19
[21] more detail on emerge_default_opts
[22] man page of emerge
[23] See footnote 19
[24] See footnote 20 or 19
[25] more detail on emerge_default_opts with-bdeps option
[26] Man page of emerge all *Feature* variables with explanation

sample_configs/make.conf_example

```
#check for cgroup
```

- o **parallel-fetch** option: Fetch in the background while compiling. Run ‘tail -f /var/log/emerge-fetch.log‘ in a terminal to view parallel-fetch progress.
- o **parallel-install** option: Use finer-grained locks when installing packages, allowing for greater parallelization. For additional parallelization.
- o **userfetch** option: When portage is run as root, drop privileges to portage:portage during the fetching of package sources.
- o **userpriv** option: Allow portage to drop root privileges and compile packages as portage:portage without a sandbox (unless usersandbox is also used).
- o **usersync** option: Drop privileges to the owner of $repository_location for emerge(1) –sync operations. Note that this feature assumes that all subdirectories of $repository_location have the same ownership as $repository_location itself. It is the user’s responsibility to ensure correct ownership, since otherwise Portage would have to waste time validating ownership for each and every sync operation.
- o **usersandbox** option: Enable the sandbox in the compile phase, when running without root privs (userpriv).
- o **cgroup** option: Use Linux control group to control processes spawned by ebuilds. This allows emerge to safely kill all subprocesses when ebuild phase exits.
- o **clean-logs** option: Enable automatic execution of the command specified by the PORT_LOGDIR_CLEAN variable. The default PORT_LOGDIR_CLEAN setting will remove all files from PORT_LOGDIR that were last modified at least 7 days ago.
- o **collision-protect** option: A QA-feature to ensure that a package doesn’t overwrite files it doesn’t own. The COLLISION_IGNORE variable can be used to selectively disable this feature. Also see the related protect-owned feature.
- o **fakeroot** option: Enable fakeroot for the install and package phases when a non-root user runs the ebuild(1) command.
- o **merge-sync** option: After a package is merged or unmerged, sync relevant files to disk in order to avoid data-loss in the event of a power failure. This feature is enabled by default
- 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 **news** option: Enable GLEP 42 news support. See here
- o **preserve-libs** option: Preserve libraries when the sonames change during upgrade or downgrade. Libraries are preserved only if consumers of those libraries are detected. Preserved libraries are automatically removed when there are no remaining consumers. Run ‘emerge @preserved-rebuild‘ in order to rebuild all consumers of preserved libraries.
- o **sandbox** option: Enable sandbox-ing when running emerge(1) and ebuild(1).
- o **ebuild-locks** option: Use locks to ensure that unsandboxed ebuild phases never execute concurrently. Also see parallel-install.
- o **strict** option: Have portage react strongly to conditions that have the potential to be dangerous (like missing or incorrect digests for ebuilds).
- o **unmerge-orphans** option: If a file is not claimed by another package in the same slot and it is not protected by CONFIG_PROTECT, unmerge it even if the modification time or checksum differs from the file that was originally installed.

- **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”. This does not affect the unpacked source. See ’noclean’ below.

sample_configs/make.conf_example

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

- **CPU_FLAGS_X86** is an USE_EXPAND variable containing instruction set and other CPU-specific features both for x86/amd64 architectures on both Intel and AMD CPUs. The easiest way to determine those is by using the app-portage/cpuid2cpuflags package.[27]

```
root #  emerge -av app-portage/cpuid2cpuflags
root #  cpuinfo2cpuflags-x86 >> /etc/portage/make.conf
```

sample_configs/make.conf_example

```
AUTOCLEAN="yes"
```

- **PYTHON_TARGETS** is USE_EXPAND variables controlling support for various Python implementations (versions) in packages. These essentially control what version of Python the package will reference during and after installation.[28]

sample_configs/make.conf_example

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

- **L10N** is variable with which we decide which extra localization support will be installed. We would like **bg** for bulg and en.

sample_configs/make.conf_example

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

- **USE** is one of the most powerful variables Gentoo provides to its users. Several programs can be compiled with or without optional support for certain items. For instance, some programs can be compiled with support for GTK+ or with support for Qt. Others can be compiled with or without SSL support. Some programs can even be compiled with framebuffer support (svgalib) instead of X11 support (X-server). You can add/subtract use flags one by one or arrange them in user defined variables as the following example.[29]

sample_configs/make.conf_example

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

SOUND = "pulseaudio"
NETWORK = "networkmanager"
FONTS = "truetype type1 cleartype corefonts"
INTEL_MODESETTING="glamor"
```

  o **SOUND** is a variable for flags associated with sound. In later on we will need **PulseAudio** as sound server, so we want every program who has that property to install the appropriate packages.

  o **NETWORK** is a variable for the network stuff. Later on we will use **NetworkManager** as a network management software for Ethernet, WiFi and etc. So we want every program who has that property to compile it.

[27]Further details for cpu_flags_x86
[28]Further details for python_targets
[29]Further details for USE and its flags