mkg: a virtual build tool for Gentoo GNU/Linux

mkg arose from a simple practical need: to speed up my Gentoo recovery process when my platform ended up in tatters after too much tinkering with it. So here is a tool that meets this need, at least for core cases and without any claim to generality, completeness or achievement: making mistakes is better than faking perfection.

The building process is encapsulated using scripted VirtualBox machines to build the platform, clone it to block device or transform it into a CloneZilla bootable ISO.

I hope this tool will contribute to democratizing a very nice distribution by smoothing out some rough edges for newcomers.

Fabrice Nicol
Montpellier, France Feb. 2021

1. About MKG

A tool to build and install Gentoo

Purpose

This software is a set of simple scripts to automate the installation of the Gentoo GNU/Linux distribution. It follows the official AMD64 install handbook, with a limited number of configuration options and a few departures from the canonical building process, aimed at enhancing stability and robustness.
It creates a Plasma or Gnome desktop platform compiled out of the latest available sources in the stable branch.
Available init systems are OpenRC (default) and systemd. Supported Gentoo profiles are:

```
default/linux/amd64/[latest stable version tag]/hardened (stable)
default/linux/amd64/[latest stable version tag]/desktop/gnome (stable)
default/linux/amd64/[latest stable version tag]/desktop/gnome/systemd (stable)
default/linux/amd64/[latest stable version tag]/desktop/plasma (stable) *
default/linux/amd64/[latest stable version tag]/desktop/plasma/systemd (stable)
```

MKG can install Gentoo on a device, and/or create an installation ISO file that holds on a single DVD.
MKG also backs up any device into a recovery medium.

Core concepts

This project builds on several others, notably by using a virtual machine to encapsulate the building process.
Compared to other projects, MKG differs by the following special features:

- the `squashfs` file system is used throughout the project to wrap up the building scripts into the virtual machines.
- virtual machines are used in combination with the CloneZilla backup tool, with a view to closely integrating backup and building workflows.

- the project lays emphasis on robustness and reproducibility. This accounts for the intentionally limited set of (currently) supported platforms: MKG should (almost) always complete a successful build on the supported platforms. To help with reproducibility, an extensive Doxygen documentation of the bash scripts is included in the repository.

Output

The output is a direct install to disk and/or an installation medium (ISO file, DVD, or bootable USB stick). Currently the minimal build ISO size is 2.9 GB for Plasma. The standard build ISO size is 3.9 GB.

Target

This software is targeted at:
- users who would like to experiment with Gentoo but are somewhat discouraged by the amount of time that installing it the hard way takes; yet who are open to face the technical challenge to some extent,
- users who already know how to install and manage Gentoo, yet who forgot to back up their OS for some time and need a quick recovery process for want of free time,
- users who bump into intractable dependency conflicts in their portage tree and want a clean bootstrap into a new platform.

Think of MKG as a bootstrapping tool, not as an Anaconda installer! Please take a look at the Gentoo AMD64 manual for more details.

The standard MKG Gentoo build is more specifically geared towards data science, with R installed along with a reasonably complete set of R libraries and Emacs. Other software included are Libreoffice and Okular, a complete TeX Live distribution for PDF-report creation, and core libraries of the Qt5 platform for the Plasma version. A minimal build is also available with just the basic desktop software.

Prerequisites

This software is designed to work on *nix platforms. It has only been tested under GNU/Linux (Ubuntu-20.04, 21.04 and Gentoo itself, current stable AMD64 branch).

For the time being, supported configurations should include the following features:

- 64-bit processor of the AMD64 (x86_64).

- Intel or Nvidia-compatible video card.
- Pre-installed software: see FAQ on dependencies.
- A working, preferably wired, direct internet connection (firewalls are not supported).

- At least 55 GB of spare disk space (in some cases up to 100 GB, see FAQ).
- A removable USB storage device (USB stick or external USB drive) with at least 55 GB of reachable space if Gentoo is to be directly installed to an external device.

2. Installation guidelines and usage

Installation guidelines

- Clone or unpack

- Run in the local mkg directory
- Check possible options and option defaults by calling: `./mkg help`

In a nutshell

MKG can be run with standard user privileges (denoted as `$`), unless test mode is used, or an operating system is directly installed on a block device, or the default third-party workflows are not used (in these cases, we use the standard denotation `#`).

- To create a bootable ISO run (use the .iso extension):
  `$ ./mkg  filename.iso`
- To burn to DVD run:
  `$ ./mkg  filename.iso burn` (if there is only one optical disc writer to your platform, you may have to run as root in some cases).
- To directly install Gentoo on any block device (disk drive, USB stick..) mounted or not:
  `# ./mkg [iso file] [burn] hot_install ext_device=/dev/sdX`
- To install a minimal platform, without **LibreOffice** and data science tools add `# (...) minimal use_mkg_workflow=false` to command line.
- To run the process silently (without the VirtualBox graphical interface) add `gui=false`
- To run in the background, either add `gui=false &` to your command line or, if you want to keep a VirtualBox GUI and not run in headless mode, first launch in the foreground **then** use `bg %n` (where `n` is the corresponding shell job shown in the output of the `jobs` command). Starting MKG in the background with `&` and `gui=true` (default value) will result in a crash. When the virtual disk is completed, if your command line has an *.iso file in it, a second virtual machine will be fired to create the ISO installer. If MKG was first launched in the background and in GUI mode (using `bg`), this chained VM will crash for the same reason. Overall, it is preferable to run in the foreground if you wish to stick to `gui` mode.

An alternative Github Actions-based way of running MKG

Instead of using MKG in its above command-line version, you may prefer, for example for testing or security motives, to use it in its *virtual instantiation* only.
In this usage, MKG does not run on your platform but only within a virtual machine over which you keep full control.
This way of running MKG requests more input from the user but avoids all but potential security issues. Artifacts that are to be used are exclusively built by automated Github Actions workflows on a regular basis from GPG-signed commits in Github projects **mkg** and **clonezilla_with_virtualbox**.
User input is limited to creating one (or two) VirtualBox machines, setting their parameters and firing them. Please consult the comments and installation advice in the master Release section for Plasma desktops or in the gnome Release section for Gnome desktops.

Bandwidth-saving options

Spare Gentoo and Sourceforge servers, cut down on bandwidth and speed up the build!

I you already fetched the appropriate tools (the minimal install Gentoo ISO, a recent stage3 archive and a Ubuntu-based CloneZilla ISO), you may just reuse them without downloading, using the following options on command line :

- `download=false` : Reuse the Gentoo install ISO (named **install-amd64-minimal.iso**)
- `download_clonezilla=false` : Reuse Ubuntu-based CloneZilla ISO (named **clonezilla.iso**)
- `download_arch=false` : Reuse stage3 XZ archive (named **stage3-amd64.tar.xz**)
- `custom_clonezilla=path/to/ISO` : Reuse previously created custom Ubuntu-based CloneZilla ISO, with added VirtualBox and guest additions (named **clonezilla.copy.iso**). This augmented version of the CloneZillo distribution is automatically created in the process of transforming the virtual disk into a CloneZilla compressed image. It is also available independently for downloads in the Release section of the companion repository clonezilla_with_virtualbox

The files should be named as indicated in bold above.
These options will save a lot of bandwidth, some electric power, and noticeably cut down on processing time, especially if a high-speed internet connection is not available.

Other options

- The default user is **fab** with password **__user2021__**. Root password is **__dev2021__**. You can specify other choices by adding `nonroot_user=name_of_user` and `passwd=password_of_user` and/or `rootpasswd=password_of_root` to command line.
- To create a Gentoo installer on a USB stick (or any block device), for example on device `/dev/sdf`, add to command line: `device_installer ext_device=sdf` or alternatively `ext_device="model_name"`, where *model_name* is the first few letters the relevant output line of `lsblk -oMODEL | grep .` (use this if there is only **one** such model tag on your platform!). You can use this to create an installer stick as an alternative to a DVD install.
- To process a VM disk already created at a prior stage into an ISO file and/or external device installation, add: `from_vm`. In case of several virtual machines, you can specify `vm=name-of-virtual-machine` (*without* .vdi).
  If `vm` is not specified, the default VM name `Gentoo` will be used.
- To process an existing Gentoo operating system into an ISO file and/or a USB-stick Gentoo installer add: `from_device ext_device=sdX` and the install options (`device_installer=...`, ISO filename and/or `burn`)
- Likewise to process an already created ISO installer into a USB stick Gentoo installer and/or burn the ISO to DVD add `from_iso`. Direct installation to device from iso is currently not supported: please use the ISO (e.g. burned to DVD) to create the installed OS.
- The project comes in with a default kernel configuration file (.config) adapted from the Ubuntu 20.04 platform. This configuration may be overall too overloaded with unnecessary built-in drivers but will come in handy to many users. Should you wish a lighter, possibly more reactive kernel, add your custom configuration file with option:
  `use_mkg_workflow=false kernel_config=/path/to/cutom/config/file`
- If your PC was made prior to 2015, its processor may not enable the AVX2 register, which is set as a global compiling option by default. In this case you should tweak the building process by adding:
  `use_mkg_workflow=false cflags=\'[-core2,-O2]\'`
  (*note the list format enclosed within* ***escaped*** *single quotes*) to command line, if your processor is at least CORE2-compatible, otherwise just `cflags="-O2"`.
- You can optionally build VirtualBox by running:
  `# ./mkg build_virtualbox`

You will have to manually add the root directory to your PATH variable and either uninstall the vanilla version of VirtualBox or place the root directory and the bin subdirectory in the PATH so that this version gains precedence.

3. Command line options

USAGE:
mkg [1]
mkg [[switch=argument]...] filename.iso [2]
mkg [[switch=argument]...] [3]
mkg help[=md] [4]

Usage [1] and [2] create a bootable ISO output file with a current Gentoo distribution.
For [1], implicit ISO output name is **gentoo.iso**
Usage [3] creates a VirtualBox VDI dynamic disk and a virtual machine with name Gentoo.
Usage [4] prints this help, in markdown form if argument 'md' is specified.
Warning: you should have at least 55 GB of free disk space in the current directory or in vmpath if specified.

Arguments with white space (like `cflags="-O2 -march=..."`) should be written in list form with commas and no spaces, enclosed within single quotes: `cflags=\'[-O2,-march=...]\'`
The same holds for paths with white space.

As of March, 2021, part of the build is performed by *Github Actions* automatically. An ISO file of CloneZilla supplemented with VirtualBox guest additions will be downloaded from the resulting automated Github release. To disable this behavior you can add `use_clonezilla_workflow=false` to command line, or build the custom ISO file beforehand using the companion project **clonezilla_with_virtualbox**. In this case, add:
`custom_clonezilla=your_build.iso`
to command line.
Within containers, `use_clonezilla_workflow`, `build_virtualbox` and `test_emerge` are not (yet) supported and will fail.
A similar procedure also applies to the minimal Gentoo install ISO.
MKG scripts and the stage3 archive are added within its squashfs filesystem by the *Github Actions* workflow of the MKG Github site.
An ISO file labelled **downloaded.iso** is automatically released by the workflow. It will be downloaded from the MKG Github release section.
This preprocessed ISO has build parameter presets. It builds the full desktop.
In particular, the following command line options will be ignored:
`bios, cflags, clonezilla_install, debug_mode, elist, emirrors, kernel_config, mem, minimal, minimal_size, nonroot_user, passwd, processor, rootpasswd, stage3_tag, vm_keymap, vm_language.`
You can however use `ncpus` with values 1 to 6 included.

Memory will be automatically allocated depending on `ncpus` value.
To disable this behavior you can add `use_mkg_workflow=false`
to command line. You will need to do so if you do not use OS build presets.

GUI mode and background runs

To run in the background, either add `gui=false &` to your command line
or, if you want to keep a VirtualBox GUI and not run in headless mode,
first launch in the foreground then use `bg %n` (where n is the
corresponding shell job shown in the output of the `jobs` command).

Options:

Boolean values are either `true` or `false`. For example, to build
a minimal distribution, add to command line:
`minimal=true`
or simply: `minimal` as `true` can be omitted (unlike `false`).

Examples

```
$ ./mkg pdfpage
$ ./mkg debug_mode verbose from_vm vm=Gentoo  gentoo_small.iso     ext_device=sdc device_installer blank burn
cleanup=false
# ./mkg download_arch=false download=false download_clonezilla=false  custom_clonezilla=clonezilla_cached.iso
use_mkg_workflow=false nonroot_user=phil
# nohup ./mkg plot plot_color=red plot_period=10 plot_pause=7 compact minimal minimal_size=false use_mkg_workflow
gui=false elist=myebuilds  email=my.name@gmail.com email_passwd='mypasswd' &
# nohup ./mkg gui=false from_device=sdc gentoo_backup.iso &
# ./mkg dockerize minimal use_mkg_workflow=false ncpus=5 mem=10000 gentoo.iso
```

Type Conventions:
b: true/false Boolean
o: on/off Boolean
n: Integer
f: Filepath
d: Directory path
e: Email address
s: String
u: URL

When a field depends on another, a colon separates the type and
the name of the dependency. dep is a reserved word for dummy defaults of dependencies i.e.
optional strings that may remain unspecified.
Some options are incompatible, e.g. `test_only` and `use_mkg_workflow`

Option	Description	Default value	Type
debug_mode	Do not clean up mkg custom logs at root of gentoo system files before VM shutdown. Boolean.	[false]	b
verbose	Increase verbosity	[false]	b

Option	Description	Default value	Type
bios	Type of bootloader: BIOS or EFI (default)	[false]	b
blank	Blank rewritable optical disk before burning installation image.	[false]	b
build_virtualbox	Download code source and automatically build virtualbox and tools. Yields VirtualBox and vbox-img executables under main directory and subdirectory virtualbox with the build, including documentation.	[false]	b
burn	Burn to optical disc. Boolean.	[false]	b
cdrecord	cdrecord full path. Automatically determined if left unspecified.	[`which cdrecord`]	f:burn
cflags	GCC CFLAGS options for ebuilds	['[-march=native,-O2]']	s
check_vbox_version	Check that VirtualBox version within a container is the same as in the host platform	[ ]	s
cleanup	Clean up archives, temporary images and virtual machine after successful completion. Boolean.	[true]	b
clonezilla_install	Use the CloneZilla live CD instead of the official Gentoo minimal install CD. May be more robust for headless install, owing to a VB bug requiring artificial keyboard input (see doc).	[false]	b
cloning_method	Method used by hot_install ext_device=... to intall Gentoo to disk. One of: 'guestfish' (default) or 'qemu'	[guestfish]	s:ext_device
compact	Compact virtual disk after VM building. Caution: this may impede hot_install and ext_device.	[false]	b
cpuexecutioncap	Maximum percentage of CPU per core (0 to 100)	[100]	n
CRAN_REPOS	CRAN repository URL for downloading extra R packages	[https://cloud.r-project.org]	s
create_squashfs	(Re)create the squashfs filesystem. Boolean.	[true]	b
custom_clonezilla	Use this previously created custom CloneZilla ISO with added VirtualBox and guest additions.	[dep]	s
cut_iso	Cut ISO files created for Gentoo installers into 2GB chunks. Entails sums=true.	[false]	b

Option	Description	Default value	Type
device_installer	Create Gentoo clone installer on external device. **ext_device** value must be specified.	[ ]	
disable_checksum	Disable checksum verification after downloads. Boolean.	[false]	b
disconnect	Unmount guest virtual disk from host.	[true]	b
dockerize	Use a pre-built Docker image to run MKG into. Incompatible with e.g. 'gui', 'hot_install', 'plot', 'test_emerge'	[false]	b
docker_image_path	URL to ../'workflow_tag2'/mygentoo-'workflow_tag2'.tar.xz	[path5]	s:workflow_tag2
download	Download install ISO image from Gentoo mirror. Boolean.	[true]	b
download_arch	Download and install stage3 archive to virtual disk. Booelan.	[true]	b
download_clonezilla	Refresh CloneZilla ISO download. An ISO file must have been downloaded to create the recovery image of the Gentoo platform once the virtual machine has ended its job. Boolean	[true]	b
download_clonezilla_path	Download the following CloneZilla ISO	[path1]	u
download_only	Only download the Gentoo minimal install ISO and stage3 archive of the day, then exit.	[false]	b
efi_size	Size of EFI partition in MiB.	[512]	n
elist	File containing a list of Gentoo ebuilds to add to the VM on top of stage3. Note: if the default value is not used, adjust the names of the 'elist'.accept_keywords and 'elist'.use files	[ebuilds.list]	f
email	Email address to send a warning to upon build completion. Note: you will have to accept so-called *insecure software* with some providers. It is not insecure if you are using your private PC throughout.	[ ]	e
email_passwd	Email password	[ ]	s:email
emirrors	Mirror sites for downloading ebuilds	[path2]	u
exitcode	Print virtual machine final exit code (0 on success)	[false]	b

Option	Description	Default value	Type
ext_device	Create Gentoo OS on external device. Argument is either a device label (e.g. sdb, hdb), or a mountpoint directory (if mounted), or a few consecutive letters of the model name (e.g. 'Hitachi HDS5C3020BLE630', 'PNY CS900 960GB' or 'WDC WD30'), if there is just one such. Requires either **device_installer** or **hot_install** on commandline.	[dep]	s
force	Forcefully creates machine even if others with same same exist. Stops and restarts VBox daemons. Not advised if other VMs are running.	[false]	b
from_device	Do not Generate Gentoo but use the external device on which Gentoo was previously installed. Boolean.	[false]	b:ext_device
from_iso	Do not generate Gentoo but use the bootable ISO given on commandline.	[false]	b
from_vm	Do not generate Gentoo but use the VM. Boolean.	[false]	b
full_cleanup	Remove virtual disk, archives and ISO files on clean-up	[false]	b
github_release_path	URL to Github Release of clonezilla_with_virtualbox.iso	[path3]	s
github_release_path2	URL to Github Release of preprocessed_gentoo_install.iso	[path4]	s
gui	Binary: true corresponds to VBoxManage startvm ... –type=gui, false to –type=headless	[true]	b
help	This help	[ ]	
hot_install	Intall to Gentoo attached device (like /dev/sdb) possibly mounted (like /media/USER/567EAF). To be used with *ext_device=...*	[false]	b:ext_device
htmlpage	Create HTML help page	[false]	b
hwvirtex	Activate HWVIRTEX: on/off	[on]	o
interactive	Allow interaction with user. This may cause deadlock if process is detached from the console (*nohup* or other methods)	[true]	b
ioapic	IOAPIC parameter: on or off	[on]	o
kernel_config	Use a custom kernel config file	[.config]	f
livecd	Path to the live CD that will start the VM	[gentoo.iso]	f

Option	Description	Default value	Type
logging	Activate logging	[true]	b
manpage	Create manpage mkg.1	[false]	b
mem	VM RAM memory in MiB	[8000]	n
minimal	Remove *libreoffice* and *data science tools* from default list of installed software. Boolean.	[false]	b
minimal_size	Remove kernel sources to minimize packaging. Not advised for personal use but OK for deployment and distribution.	[true]	b
mirror	Mirror site for downloading of stage3 tarball	[path2]	u
ncpus	Number of VM CPUs. By default the third of available threads. You should not configure virtual machines to use more CPU cores than are available physically. This includes real cores, with no hyperthreads.	[count]	n
no_run	Thwart VM generation, only perform other operations (if valid).	[false]	b
nonroot_user	Non-root user	[fab]	s
pae	Activate PAE: on/off	[on]	o
paravirtprovider	Virtualization interface: kvm for GNU/Linux, may be tweaked (see VirtualBox documentation)	[kvm]	s
passwd	User password	[_user2021_]	s
pdfpage	Create PDF help page	[false]	b
plot	Plot VDI disk size using GNUPlot	[false]	b
plot_color	Plot line color between simple quotes (e.g: 'cyan','red' etc.)	['cyan']	s
plot_pause	Number of seconds of plot display per minute of processing. Maximum 50.	[5]	n
plot_period	Number of minutes elapsed between two successive plots	[10]	n
plot_position	Plot position of on monitor screen (top-left angle) in pixels e.g '0,0' or '500,500'	['0,0']	s
plot_span	Number of minutes of virtual disk processing to be plotted, back in time	[1200]	n
postpone_qemu	If using **share_root='r' or 'w'**, do not run the virtual machine but use the VDI disk corresponding to vm=... Otherwise start 15 minutes later.	[false]	b
processor	Processor type	[amd64]	s

Option	Description	Default value	Type
pull	Invoke 'git pull' in mkg local repository on startup.	[false]	b
quiet_mode	Silence output except for the most severe errors.	[false]	b
rootpasswd	Root password	[_dev2021_]	s
rtcuseutc	Use UTC as time reference: on/off	[on]	o
scsi_address	In case of several optical disc burners, specify the SCSI address as x,y,z	[ ]	s
shared_dir	Host mount point for guest virtual disk filesystem	[/vdi]	s
share_root	Mount guest virtual disk to host folder **shared_folder**. Argument is 'w' to enable write I/O or read-mode 'r'. May raise security issues, see Wiki.	[dep]	s:shared_dir
size	Dynamic disc size	[55000]	n
smtp_url	SMTP URL of email provider for end-of-job warning. Default: gmail SMTP	[smtps://smtp.gmail.com:465]	u
stage3	Path to stage3 archive	[stage3.tar.xz]	f
stage3_tag	Type of stage3 package [openrc, systemd, hardened-openrc]. Experimental.	[openrc]	s
sums	Output b2sum and sha512sum of ISO output (only) in file SUMS_[name of ISO output]	[false]	b
swap_size	Size of swap partition in MiB.	[1024]	n
test_emerge	Test whether emerge will be able to install packages in the VM before starting it.	[false]	b
test_only	Only test if portage will be able to install packages. Do not create a virtual machine.	[false]	b
timezone	Set timezone. See /usr/share/zoneinfo for formats.	[US/Eastern]	s
usbehci	Activate USB2 driver: on/off	[off]	o
usbxhci	Activate USB3 driver: on/off. Note: if on, needs extension pack.	[off]	o
use_bsdtar	Use 'bsdtar' (from libarchive) to extract CloneZilla ISO files if creating an ISO installer.	[true]	b
use_clonezilla_workflow	Use Github Actions workflow to add virtualbox guest additions to CloneZilla ISO.	[true]	b

Option	Description	Default value	Type
use_mkg_workflow	Use Github Actions workflow to preprocess minimal Gentoo ISO by adding MKG scripts inside the squashfs filesystem.	[true]	b
use_qemu	Use qemu instead of guestfish for installing Gentoo to block device with **hot_install**	[false]	b
vbox_version	Virtualbox version	[6.1.20]	s
vbox_version_full	Virtualbox full version	[6.1.20]	s
vbpath	Path to VirtualBox directory	[/usr/bin]	d
vm	Virtual Machine name. Unless 'force=true' is used, a time stamp will be appended to avoid registry issues with prior VMs of the same name.	[Gentoo]	vm
vm_keymap	Add support for non-US English keymaps. Use values in /usr/share/keymaps, e.g. fr, de, us.	[us]	s
vm_language	Add support for non-US English language as a default. Use standard abbreviations. Must be at least 5 characters: e.g. fr_FR, de_DE, fr_FR.utf8	[en_US.utf8]	s
vmpath	Path to VM base directory	[$PWD]	d
vtxvpid	Activate VTXVPID: on/off	[on]	o
workflow_tag	Tag version (vX.Y) of the release of file clonezilla_with_virtualbox.iso by Github Actions	[v2.0]	s
workflow_tag2	Tag version (vX.Y) of the release of file preprocessed_gentoo_install.iso by Github Actions (MKG site)	[release-master or release-gnome]	s

path1 https://sourceforge.net/projects/clonezilla/files/clonezilla_live_alternative_testing/20210505-hirsute/clonezilla-live-20210505-hirsute-amd64.iso/download
path2: http://gentoo.mirrors.ovh.net/gentoo-distfiles/
path3: https://github.com/fabnicol/clonezilla_with_virtualbox
path4: https://github.com/fabnicol/mkg/releases/download
count: nproc –all / 3 ## 4. Warnings and limitations

Warnings

- The install media will wipe out all data on the Desktop main disc **/dev/sda**. It leaves no choice of the target disk and runs **non-interactively** from beginning to end.
  Use it with care and only if you want to do a fresh install of your main PC disk. You have been warned.

- Building the platforms comes in with the third of available threads as a default.
  If resources are strained, rerun with N cores by adding `ncpus=N` to commandline.
  The `ncpus` number of jobs is used to implement the portage `CFLAGS` parameter in **make.conf**, so that the building process is in line with ressources granted to the virtual machine. User should review this parameter later on according to the characteristics of the target platform.

Limitations

- The ISO output should not be greater than 4096 MB. This limit is unchecked.
- Currently video card support is limited to Intel and Nvidia (Nouveau driver).
- Legacy BIOS bootloaders (command line option `bios`) have not been tested, if possible stick to EFI (default).
- Internationalization is untested except for French.
  To test support for another language, for example for German, use the command line option `vm_language=de_DE vm_keymap=de`
  Check the Gentoo localization guide.
- Gentoo profiles other than Plasma, Gnome and OpenRC Hardened (experimental) are not supported.

VirtualBox limitations

MKG uses Oracle VirtualBox (VB) to run a virtual machine Gentoo distribution builder.
This implementation choice implies a handful of VB-related limitations.

No hyperthreading support

VirtualBox does not support (Oct. 2021) hyperthreading. This means that you should set the value of `ncpus` at a maximum of your number of **physical** cores N (and more safely at N-1). Even if your CPU supports *hypertheading*, which presents 2 logical cores to the OS, you will not be able to put this feature to good use with MKG as VB will only *see* N cores, not 2N. By default, `ncpus` is set at `$(nproc)/3`. When hyperthreading is supported, this value will almost surely be two thirds of the number of physical cores, and one third of the number of logical cores. When hyperthreading is not supported, it will be one third of physical cores (with an obvious minimum of one core).

Unsupported CPU extensions

At the time of writing (March 2021), VB has known limitations w.r.t the implementation of CPU extensions for post-Ivy Bridge generations. Concerned CPU features are notably: **fma, f16c, bmi, bmi2**. However the AVX and AVX2 features are supported. When specifying `cflags=\'[...,-march=generation]\'` with `generation` equal or superior to `sandybridge`, build failures may result if configured packages depend on these CPU features. This is notably the case for the Qt5 libraries. Pending VB upgrades, the following policies should be followed when using the `cflags` option with a `-march` specification higher than `sandybridge` (or `ivybridge` if AVX2 is the only new CPU feature used by builds).
It is therefore advised, until these limitations are fixed by Oracle developers, either not to specify `-march=...` or to stick to:

```
cflags=\'[...,-march=native]\'
```

This will not, however, create a build with the full range of host CPU optimizations, unlike what usually happens when `-march=native` is used in the `make.conf` CFLAGS variable. If you wish to optimize your installed platform when fully operational, you should later

on rebuild it using:

```
emerge -e @world
```

The `cflags` option `-march=...` should therefore be limited to *downgrading* the CPU model to the following list of values:

`'x86-64', 'i386', 'i486', 'i586', 'pentium', 'lakemont', 'pentium-mmx', 'pentiumpro', 'i686', 'pentium2', 'pentium3', 'pentium3m', 'pentium-m', 'pentium4', 'pentium4m', 'prescott', 'nocona', 'core2', 'nehalem', 'westmere', 'sandybridge'`

5. Troubleshooting, Testing and Debugging

Troubleshooting

Foreword

Experience shows that many build failures result from:

- insufficient allocated RAM (parameter `mem`, 8GB by default)
- too many CPU cores allocated to the VM (parameter `ncpus`, a third of total CPU threads by default).

The two parameters are not independent and the defaults, for a modern PC with 16GB of RAM, is a relatively safe one. However, increasing even moderately the number of threads allocated to the VM should come with an associated increase of allocated RAM (2 GB by extra thread is a minimal requirement from experience). A VM that runs short of RAM will inexplicably crash without indicating adequate reasons. An occasional error message (if any) will refer to the build being killed by user interruption, while it was not, which may be confusing.

Another common cause of failure arises from changes in the portage tree, notably in the set of use or keyword parameters (more occasionally, licensing changes). Users are invited to closely check error messages related to these feature changes. It is a fairly easy fix to adjust files **ebuilds.list.use** and **ebuilds.list.accept_keywords** at the root of the source directory (or within the VM) to fix these issues. In this event, please report the fix in the Issues section of this site.

Stalled machines

Machine stalling is another issue that may arise sometimes from VirtualBox software limitations or bugs. Most occurrences of stalling happen at the end of the building procedure, sometimes on compacting (this is why `compact` is not the default value).
If stalling happens when all jobs have actually been completed, the fix is simple: stop the VM and let the script proceed:

```
# VBoxManage controlvm name_of_vm poweroff
```

Otherwise you will have to restart the VM using either the VirtualBox graphical interface or:

```
# VBoxManage controlvm name_of_vm reset
```

Then have a go at debugging techniques outlines below.

Testing

Testing portage dependencies is advised if it has not been performed recently.
By default, as of tag v2.3, MKG tests package dependencies as a *prior step* to all building operations. The job will not proceed if tests fail for some reason, avoiding most of the causes of time loss due to failing virtual machines.

This step is performed in about 3 to 10 minutes. It can be deactivated by adding `test_emerge=false` to command line.
It is also possible to just test-run MKG so that to make sure that the combination of use, keyword and package specifications in files **ebuilds.list.*** is coherent with the portage tree of the day. Once these sanity checks are performed, MKG exits with a diagnosis of possible inconsistencies and also stops if tests are passed. It is up to the user to fix possible portage tree issues using the messages displayed by the test run and the Gentoo Installation handbook (notably section When Portage is complaining).
To enable test runs, add option `test_only` to command line.
It is advised to do so if MKG has not be run for more than a few days:

```
# ./mkg test_only
```

As of tag v2.3, it is possible to remotely run-test MKG this way on Github Actions, using the workflows yaml script under .github/workflows in the repository, by triggering the `workflow_dispatch` event. Testing takes about half an hour and, if successful, ends as follows:

```
1197   * IMPORTANT: 6 news items need reading for repository 'gentoo'.
1198   * Use eselect news read to view new items.
1199
1200  <13>Mar 13 00:42:39 root: [MSG] Portage tests were passed.
1201  <13>Mar 13 00:42:39 root: [INF] Unmounting host filesystem
1202  <13>Mar 13 00:42:44 root: [INF] Removing mount directory
1203  <13>Mar 13 00:42:44 root: [INF] Cleaning up. mnt is a mountpoint
1204  <13>Mar 13 00:42:44 root: [MSG] Gentoo building process ended.
```

Figure 1: workflow

Note that you cannot test-run MKG using `test_only` and at the same time use the Github Actions preprocessed Gentoo install ISO file (see **usage section of this wiki**). So `test_only` sets `use_mkg_workflow` to `false` automatically.
Below is a commented excerpt of the console output for this invocation:

Output	Comment
Mar 11 22:35:18 fab: [INF] Fetching live CD...	Downloading the Gentoo minimal installation CD
Mar 11 22:35:30 fab: [INF] Fetching stage3 tarball...	Downloading stage3 archive
Mar 11 22:35:36 fab: [INF] Testing whether packages will be emerged...	Launching the test
Mar 11 22:35:36 fab: [INF] Unmounting host filesystem	Cleaning up mnt, mnt2 mount directories
Mar 11 22:35:42 fab: [INF] Moving stage3.tar.xz to mnt2/squashfs-root	Moving files to the chroot target directory
Mar 11 22:35:54 fab: [MSG] Using CFLAGS=-march=native -O2	Mentioning CFLAGS value
[INF] Merging portage tree...	Now testing for portage tree conflicts
* Generating 2 locales (this might take a while) with 12 jobs	Locale setting
app-misc/pax-utils: 1.2.9 1.2.6 none (...)	Some packages may occasionally be uninstalled
»> Unmerging (1 of 3) app-misc/pax-utils-1.2.9...	...
[MSG] Using profile=default/linux/amd64/17.1/desktop/plasma	Selected profile
[INF] Updating cmake...	cmake must first be built

Output	Comment
»> Installing (5 of 5) dev-util/cmake-3.18.5::gentoo	...
[INF] Updating python. Please wait...	Working around a circlular dependency involving Python
»> Installing (1 of 1) dev-lang/python-3.9.1-r1::gentoo	...
[INF] Testing update of world set...	Now on to updating @world set
Calculating dependencies ..... done!	...
[ebuild N] dev-qt/qtchooser-66 USE="-test"	
(...) list of ebuild updates	...
[Possible message about conflicts]	If none, @world update was passed.
[INF] Testing whether packages may be merged...	Now on to installing packages in file ebuilds.list.complete or minimal
These are the packages that would be merged, in order:	
Calculating dependencies ... . done!	
[ebuild N] dev-qt/qtchooser-66 USE="-test"	
(...) long list of ebuild dependencies	...
[Possible message about conflicting dependencies	If none, this test was passed.
Mar 11 22:43:59 fab: [MSG] Portage tests were passed.	All is fine
Mar 11 22:43:59 fab: [INF] Unmounting host filesystem	Cleaning up
Mar 11 22:44:02 fab: [INF] Removing mount directory	...
Mar 11 22:44:02 fab: [MSG] Gentoo building process ended.	End of test: OK.

Debugging

Quick check

When building has stopped, a quick check procedure should first be used to ensure that the building process exited with code 0:

```
# ./mkg exitcode vm=... no_run
```

if the exit code indicated by the output message of the command is non-null, then proceed as indicated below, choosing between VM debugging (you need how to handle VirtualBox) or shared root debugging (easier, but you will need a decent installation of `qemu` on your platform).

You can combine `exitcode` with any Gentoo OS building command line (so not with `from_device` or `from_iso`). In this case, the exit code of the VM process will be printed on exiting MKG. Use `no_run` in combination with `exitcode` in the reverse case, when you do not want to start a building process again.

Debugging in the VM

Some users may want to debug the running VM themselves. This is only possible if MKG is not run in silent mode (so without command line option `gui=false`). You may follow these steps:

- stop the VM if still running.
- check the parameters and launch it again.

- select your keyboard locale about 30 seconds after boot when requested.
- enter the appropriate replies in the network settings menus (usually OK if you are connected to a wired network)
- interrupt with `Ctrl + C` within 5 seconds after the last menu. Run `ifconfig` to check that your network connection.
- mount /dev/sda4 to /mnt/gentoo and chroot into it:

```
# mount /dev/sda4 /mnt/gentoo && cd /mnt/gentoo
# for i in proc sys dev dev/pts run; do mount -B /$i /mnt/gentoo/$i; done
# chroot /mnt/gentoo
```

- proceed with your debugging session. Preferably use `nano` as an editor (I've had occasional issues with `vim`)
- at the end of the session, run `exit` and (preferably) unmount the mount points before shutting down the VM.

Shared root debugging

You may prefer the convenience of debugging under a shared host directory, to which the guest virtual disk will be mounted.
Be aware that this may cause security issues, which are aggravated if the VM is running and/or write permissions are granted (normally the connection should crash in this case).
Add the following options to command line:

- `# ./mkg vm=your_vm_name share_root="r" ["w" for write permissions, "r" for read-only]`

Use this to allow mounting of the virtual machine VDI disk root to a host directory, with write permissions if "w" is specified and read-only if "r" is specified.

- `# ./mkg vm=your_vm_name share_root=... shared_dir=/path/to/host/shared/dir`

Use this to avoid using default `/vdi` value and use a custom mount point instead.
It is advisable to use the above options when the virtual machine has been stopped.
You may however use them on launch too (use `share_root=r` in this case). In this case, a **15 minute** delay will be enforced, which is the time requested by the machine to launch and partition the virtual VDI disk (plus a safety margin). Do not consider this delay to be a bug: the exact time of partitioning is not known and depends on hardware.
While running, the following logs are generated by the building process:

- While running `mkvm_chroot.sh:adjust_environment()`
  - partition_log: partitioning operations
  - emerge.build: basic tools, updating @world, keyboard, timezone, locale
- While running `mkvm_chroot.sh:build_kernel()`
  - /kernel.log: kernel source code install
  - /usr/src/linux/kernel.log: kernel build
- While running `mkvm_chroot.sh:install_software()`
  - log_install_software.log: installation of software listed in ebuilds.list.minimal or ebuilds.list.complete

  - Rlibs.log: installation of R libraries for the full distribution install (empty for the minimal one).
- While running `mkvm_chroot.sh:global_config()`
  - sddm.log: SSDM configuration for the Plasma desktop master branch.
  - grub.log: grub install log.
  - useradd.log: only if adding user was not possible.
- While running `mkvm_chroot.sh:finalize()`
  - log_uninstall.log: uninstalled software
- res.log: exit code of mkvm_chroot.sh.
  The following exit values may indicate where errors appeared:
  - odd number: issue with adjust_environment()
  - code & 2 == 1: issue with build_kernel()
  - code & 4 == 1: issue with install_software()
  - code & 8 == 1: issue with global_config()
  - code & 16 == 1: issue with finalize()

Filing an issue

Before filing an issue, please check that you are using the latest Github source code, unless repository changes since your version are clearly irrelevant to your case.
For issue reports to be useful for debugging, please follow the procedure indicated below as closely as possible:

- Minimally:
  - your command line
  - the date and version of the starge3 archive, the Gentoo install ISO and the CloneZilla ISO files
  - versioning references (tag or commit, date)
  - the output of:
    `# grep \\[...\\] /var/log/syslog` > log if `syslogd` was previously installed (which is often the case).
    Alternatively, rerun with `tee`:

```
# [your mkg command] 2>&1 | tee log
```

and attach log to your post.

- If one of your virtual machine fails, then first check that you have installed `guestfish`.

Please run again your job with `compact=false debug_mode verbose` added to command line, *without* an ISO output.
Once the new job has ended, plug in an **empty or freely erasable** USB key or external disk (minimum size: 55 GB).
Check the device label (say sdX) of this mass storage device using `fdisk -l`. Do not mount it.
Now run:

```
# guestfish --progress-bars  --ro -a [your VDI disk] run : download /dev/sda /dev/sdX && sync
```

This will take between 3 and 5 minutes.
Note: All data on your mass storage device will be erased and replaced with the downloaded virtual machine.

Mount the device for example to `/mnt` and `/mnt2`:

```
# mount /dev/sdX4 /mnt && mount /dev/sdX2 /mnt2 && cd /mnt
```

You will see a number of files at the root of `/mnt`. Report the full list of files:

```
# ls -al * > files.log
```

Join a tarball of the log and ebuild files, and the syslogs (if any):
```
# tar cJvf debug.tar.xz *log *build* [var/log/syslog*]
```

Now cd to `/mnt2` and report the state of kernel builds:

```
# cd /mnt2 && ls -al * > boot.log
```

Attach `files.log`, `boot.log` and `debug.tar.xz` to your post.

6. Packaging

To package a Gentoo installer using MKG for distribution or tests, you may want to reduce its size to a minimum, e.g. to keep bandwidth as low as possible.
MKG provides two options for packaging:

- `compact`
  This option zeroes the virtual disk free space then requests VBoxManage to compact the resulting disk. This may cut down the VDI disk size by a factor of 2.5 to 3. A side effect is that after xz-compression by CloneZilla, the resulting ISO file of the installer is about 10 % smaller than without this option.
  Use this option with care: it is not advised in combination with `hot_install ext_device=...` to directly install the distribution to a mass storage device (although this has worked in tests).
- `minimal_size`
  This options unmerges `gentoo-sources` from the VM portage tree and cleans up /usr/src. This is ill-advised if you want to be able to use your distribution out-of-the-box, as many ebuilds rely on kernel sources to install. However this cuts down the size of the package noticeably, so it may come in handy provided that you warn users about emerging `gentoo-sources` as a post-install step, followed by kernel selection (`eselect kernel set N`) and the minimal `make` steps (`make syncconfig`, `make modules_prepare`), which are occasionally necessary for a few ebuilds. As of v2.4, this option is activated by default and does not need to be invoked. It can be deactivated by adding `minimal_size=false` to command line.
  Note that if you do this, `xorriso` may likely fail to create the ISO output (if requested) as its size will for most configurations be higher than the limit size of 4GB. So `minimal_size=false` should be reserved for debugging, or when directly installing the OS to an external partition, without the intermediary ISO file output, using `ext_device=sdX hot_install`

7. Using MKG for backups

MKG as a backup tool

Although MKG has been written to quickstart a bootstrapped Gentoo platform, it can also be used as a backup tool.
The overall concept is flexible and should be adapted to particular user needs, yet roughly runs as follows:

- backing up user data is a fairly easy task that is left to the user's responsibility (a simple `rsync` script covers most needs)

- backing up the operating system (system-wide binaries, configuration files and bootloaders) is harder and hardly ever feasible while the platform is operating (so-called *hot* backups, which some commercial software boast on performing). When it is, it requests a lot of disk space and implies some computing overhead for compression and IO operations.
- MKG offers a simple alternative along the following lines for OS backups, which relies on the combination of the KISS principle and a *benign neglect* approach:
  - do no care about bootloaders and (most) configuration files, as they will be automatically recovered with standard specifications,

  - save a user-defined, flexible list of critical configuration files,
  - simplify the backup scheme so that backups can be performed in real time

  - save your ebuild list as a core backup target.

It is up to users to adapt these principles to their needs. Below is the core of the backup script I personally use:

```
#!/bin/bash
# Before starting this, it is better to sync-update-and-clean the package tree
# to avoid inconsistencies.
# Do this rather manually than scripting:
# emerge –sync
# emerge -auDN –with-bdeps=y @world
# emerge –ask depclean
# revdep-rebuild
#
updatedb
qlist -I | uniq > packages
# If you want to save package versioning, use: -Iv
for i in conf json yaml xml rc # [other formats here]
do
  locate -b *\.$i
done | sed -r '/\/(run|dev|proc|sys|tmp|var\/tmp)/d' > config.list
# You should tweak this to systemd if this is your OS loader
echo '/etc/init.d' » config.list
# You may want to save all of /etc in the above line # Another option is to save /etc using a local Git repo. # You
should tweak this to gdm or lightdm etc.
# depending on your desktop logging interface
echo '/usr/share/sddm' » config.list
# [At this stage I fine-tune config.list. ]
tar –xattrs -cJvf config.tar.xz -T config.list
tar –xattrs -cpJvf bash.tar.xz /etc/bash /etc/profile ~/.bash*
tar –xattrs -cpJvf portage.tar.xz /etc/portage
tar –xattrs -cpJvf kernel.tar.xz /boot/config*
```

And that's about it. Backed-up data consists of simple text files only with an average compressed size of 10 MB.
Backup can thus be done frequently through a `chron/rsync` job with almost no overhead and disk space usage. The recovery procedure then unravels as follows:

- create a USB key installer (I use a high-transfer rate stick):
  `# ./mkg [...] device_installer ext_device=...`

or download a binary release from the **Release** section of this site and copy it with `dd` as indicated there.

- boot your computer to the USB recovery medium. Reboot after the cloning has ended (about 3 minutes).

- when logged to Gentoo, connect another USB stick with the tarballs on and extract the config, portage, kernel and bash tarballs into their original locations. Copy the `packages` file to your home (about 2-3 minutes).
  Baseline recovery is performed in exactly 6 minutes using quality USB sticks and an SSD main disk. Now:

- reboot and run in your home:

  tar –xattrs -xpJvf portage.tar.xz
  # reset your portage specs to previous
  rsync -avr etc/ /etc
  # reset your kernel to previous
  # tar –xattrs -xpJvf kernel.tar.xz
  rsync -avr boot/ /boot
  emerge gentoo-sources
  # or your previous kernel version if not latest stable, see config files in kernel.tar.xz:
  # emerge =sys-kernel/gentoo-sources-(version)
  eselect kernel set 1 [or 2 if restoring a newer kernel]
  cd /usr/src/linux && cp /boot/config*[your kernel version] .config && make syncconfig
  make modules_prepare && cd -
  # only if you had overlays:
  # emerge app-portage/layman
  # layman -S # layman -a (your overlays)
  emerge -auDN –keep-going –with-bdeps=y $(cat packages)

or, if you saved package versioning:

```
# emerge -auDN --keep-going --with-bdeps=y $(while read pack; do echo =${pack}; done < packages)
```

In the event of a failure, you should try this first:
`# emerge --resume`

The time taken by the portage merge step is variable as it depends on how much software you installed on top of the baseline MKG Gentoo distribution.
On average this takes about 2 to 5 hours starting from the more complete version of the distribution.

If your kernel version was more recent than that of the MKG distribution, you will have to rebuild it, which is a simple task (please consult the Gentoo installation manual again). In a nutshell, with administrative rights:

cd /usr/src/linux
make -j4 && make modules_install # change 4 into the number of assigned cores
rm /boot/vmlinuz* && rm /boot/initramfs* # in case you are short of space under /boot
make install
make clean
genkernel –install initramfs
grub-mkconfig -o /boot/grub/grub.cfg

Now reboot. You can now proceed with user data restoration using your rsync procedure of choice.

Warning 1. Take extra care if restoring configuration files, especially from **/etc**. You should always restore the full kernel and software packages *before* restoring your previous **/etc** directory (except for portage config files). Note that your previous passwords will also be restored in this case. Tricky issues may arise if kernel or critical packages cannot be restored with the same versioning as in your backup. This is why it is preferable to backup config files and package lists on a daily basis. Do **not** perform this if you are not in a position to restore all critical system packages in their original version number. In any case do not forget to exclude **/etc/fstab** from your backup as copying it back may cause boot failures. This is an acknowledged limitation of the present approach: no partition design backup.
2. Restoring **/opt** and **/usr/local** is less risky. You should restore at least /usr/local *after* you have compeletely updated your @world set as indicated above, to avoid potential environment or versioning mismatches between similar software under /usr/local/ and /usr. Run `ldconfig` after this optional step.

Much of the above code can be put together into a custom install script that will also fine-tune how configuration files are reinstalled (this cannot be generalized as it is platform or user-specific).
This procedure has been tested numerous times and is overall robust. Expect some occasional rough edges though: you may have to resolve a small number of portage conflicts or use/keyword issues, which actually will reflect undetected inconsistencies in your original package tree.

Environmental issues All source-based OS distributions, not only Gentoo or Arch but also the BSD family, have an environmental weak point: building is a power-consuming operation that takes about 1 to 4 kWh for the baseline configuration. On top of this price tag, extra time and power must be added for user-installed software. As far as power expenses are concerned, source-based operating systems therefore seem to compare unfavorably to standard binary distributions, which cheaply duplicate and deploy their builds once they have been created.
My personal take on this issue is that all environmental footprints should be taken into consideration in the lifetime of a platform. The replacement of power-demanding, conventional backups to hard disk with the very lightweight, energy-saving procedure outlined above more than makes up for the extra amount of power requested by source code compilation, at least if users consent to refraining from overbuying backup disk space and replace it with optical media or cheap USB sticks for critical text file backups.
Over the lifetime of a Gentoo platform, the environmental footprint of the saved disk ware (including the extra power needed to keep it running) should be greater than the extra power costs incurred, by an order of magnitude if the saved footprint is evaluated based on the market value of disk hardware compared to electric power. The comparison should even be more favorable in the long run,

with the expected rise in renewable electric power production.

8. Running MKG in Docker containers

If you do not intend to experiment with MKG code, advanced options or build parameters, have a go at the companion project **mkg_docker_image**. Docker images are released automatically, signed and verified by third-party workflows. They are based on official Docker Gentoo images, supplemented with extra software using the repository Dockerfile. Within the docker container, the VirtualBox machines will be protected from possible external hazards, and reciprocally the host will be mostly immune from potential hazards affecting the nested machines.

8.1 The short story

You can simply use MKG option `dockerize` with administrative rights and let it go:

```
# ./mkg dockerize gentoo.iso
```

You may add other valid MKG options to this command line, provided that they do not imply graphic display or mounts (like `test_emerge`, `build_virtualbox`, `use_clonezilla_workflow=false` or `plot`).
Performance-wise, you may specify `ncpus=N`, N between 1 and 6 included, and currently no more. In this case, memory requirements (option `mem`) will be automatically adjusted (from 8 GB for 1 core to 15 GB for 6 cores).
Then wait for about a 15 to 24 hours, depending on your platform and build target (Plasma takes longer than Gnome).
The ISO installer will be fetched back from the container upon completion of the Docker job.
If this does not work, try to fetch back your ISO installer using the standard command line:

```
#  docker cp mygentoo:release-master/mkg/gentoo.iso .
```

(Replace tag `release-master` with `release-gnome` is you checked out the **gnome** branch rather than **master**.)

8.2 The long story

What follows is aimed at users who wish to keep control over how containers are created.

8.2.1 Image availability

Images are made available:

- in the Github Releases section, as automated output of Github Actions workflows,
- on Docker Hub, as *autobuilds*
- or pulled from Docker Hub, using the standard invocation:
  `$ docker pull fabnicol/mkg_docker_image/branch:[tag]`
  where `branch:[tag]` is for the image version tag corresponding to a given branch.

To check the availability of version tags, have a look at the Docker Hub repository. Usually latest images are tagged **master:latest** for Plasma desktops or **gnome:latest** for Gnome desktops.

8.2.2 Prerequisites

You will just need to install VirtualBox kernel modules (and their dependencies) on your host computer, which may or may not imply installing the whole VirtualBox package on the host, depending on your platform and package manager. On Gentoo itself, you will just have to merge:

```
app-emulation/virtualbox-modules app-emulation/docker
```

Installing a full virtualization toolchain on your host will not be necessary, all dependencies, including the nested VirtualBox toolchain, being delegated to the Docker container.

8.2.3 Limitations

Some limitations currently apply to MKG within Docker containers:

- `qemu` and `guestfish`-based options (`share_root`, `shared_dir` and `hot_install`) are not (yet) supported.
- all options that involve `chroot` are not available, except for `ncpus`: e.g. `use_clonezilla_workflow=false`, `test_emerge`, `mem` or `build_virtualbox`
- graphical interface display is not yet supported.

See MKG help for details.
Containers are created using a multi-stage build, from official Gentoo stage3 AMD64 Docker images. They are fully functional Gentoo distributions augmented with a handful of linux utilities and VirtualBox. For packaging purposes, kernel sources under `/user/src/linux`, and the ebuild database under `/var/db/repos/gentoo` have been removed to keep size down. They can be easily restored using the following command line sequence:

```
# emerge --sync
# emerge gentoo-sources
# eselect kernel set 1
# cd /usr/src/linux && make syncconfig && make modules_prepare && cd -
```

8.2.4 Building the Docker image

In what follows, replace `1.0` with the tag of choice. A list of valid tags can be obtained by clicking on the Github tags button on the mkg_docker_image main repository page.

Building Gentoo official portage and stage3 images first You will need to update docker to at least version 20.10, enable experimental docker features and add the `buildx` plugin if you do not already have installed it.

First build fresh official Gentoo portage and stage3 images, following indications given by the official site: download this repository and within it run:

```
# TARGET=portage ./build.sh
# TARGET=stage3-amd64 ./build.sh
```

You created gentoo/stage3:amd64 and gentoo/portage:latest with the above commands. Alternatively, you can pull then from Docker Hub:

```
# docker image pull docker.io/gentoo/portage
# docker image pull docker.io/gentoo/stage3:amd64
```

Then download or clone the present repository In the source directory, run:

$ sudo docker build -t mygentoo:1.0 .

Or using `buildx` (advised):

$ docker buildx build -t mygentoo:1.0 .

Adjust with the tag name you want (here mygentoo:1.0).
Allow some time (possibly several hours) to build, as all is built from source.

(Optional) Compress the image For packaging purposes it is advised to compress the resulting image using docker-squash. Optionally clean the container of kernel sources:

```
# docker run --entrypoint bash -it mygentoo:1.0
```

[Note the container ID on return.]

```
(container)# rm -rf /usr/src/linux && exit
```

Then commit the container and tag it:

```
# docker commit ID
# docker tag ID mygentoo:1.1
```

Then use docker-squash:

```
# docker save ID | docker-squash | docker load
```

Finally use **zip** or **xz** compression to archive the squash tarball.
The resulting compressed tarball is about 15 % the size of the Docker image created by the above build stage.

8.2.5 Using the Docker image

Running MKG within the container

- Say you just built **docker.io/gentoo/mygentoo:1.0**, and as for the other two base images, firt pull it from cache:
  `#docker image pull docker.io/gentoo/mygentoo:1.0`

- Now run the container using:

```
# docker run [--privileged [-v /dev/cdrom:/dev/cdrom -v /dev/sr0:/dev/sr0 (...)]] \
-it --entrypoint bash --device /dev/vboxdrv:/dev/vboxdrv -v /dev/log:/dev/log mygentoo:1.0
```

The `--privileged` option is only necessary if you are to create a CloneZilla installer as an ISO image within the container.
The `-v /dev/cdrom ...` option is only necessary if you wish to automatically start burning your ISO installer to optical disc after completion of the building process. It should be adjusted depending on your hardware and platform configuration; these defaults will work on most GNU/Linux platforms but may have to be changed on other *nix operating systems.

- Once in the container, note its ID on the left of the shell input line.
- If the image contains an **mkg** directory, run `git pull` within it to update the sources.
  Otherwise (depending on versions), clone the *mkg* repository:

```
# git clone --depth=1 https://github.com/fabnicol/mkg.git
```

and then `cd` to directory **mkg**.

- Run your `./mkg` command line, remembering to use `gui=false` and not to use `share_root`, `hot_install`, `from_device`, `use_clonezilla_workflow=false`, `mem` or `test_emerge` (but you can use `ncpus`).
- Preferably use:

```
# nohup (...) &
```

so that you can monitor the build in **nohup.out**

- Once the virtual machine is safely launched, monitor the run using:

```
# tail -f nohup.out
```

- Once the process is safely running, exit using Ctrl - p Ctrl - q.
- You may come back again into the container by running:

```
#  docker exec -it ID bash
```

- You may follow the build from your host by running:

```
# docker cp ID:/mkg/nohup.out . && tail -n200 nohup.out
```

Running MKG from the host Alternatively you can run your command line from the host, preferably in daemon mode (-d):

```
# docker run -dit [--privileged [-v /dev/cdrom:/dev/cdrom -v /dev/sr0:/dev/sr0 (...)]] \
--device /dev/vboxdrv:/dev/vboxdrv -v /dev/log:/dev/log mygentoo:1.0 [your mkg options]
```

A nice way to avoid long command lines is to add to your ~/**.bashrc**:

```
alias mkg="sudo docker run -dit [--privileged [-v /dev/cdrom:/dev/cdrom -v /dev/sr0:/dev/sr0 (...)]] \
--device /dev/vboxdrv:/dev/vboxdrv -v /dev/log:/dev/log $@"
```

so that after running `source ~/.bashrc`, you just have to call mkg as if it were an installed script:

```
# mkg [your image name first: here mygentoo:1.0] [your mkg argument names: gentoo2.iso ncpus=2 verbose
[...]]
```

[note the ID when the function returns]

Note that `gui=false` is already set in this launch mode, so it does not need to be specified (and should not be overridden).

You can check the container state by shelling back into it:

```
# docker exec -it ID bash
```

and within it examine **nohup.out** which logs the job. Then exit as usually (`Ctrl-P, Ctrl-Q`).

8.2.6 Switching from Plasma to Gnome and back

You should create one image for Gnome and another for Plasma.
Just checkout the **gnome** branch of this repository, then build your image and run as above without modification to obtain a Gnome

desktop rather than a default Plasma desktop. If you want to preserve both options, it is advised to tag your images accordingly. For example, checkout the **gnome** branch and run:

```
# docker build -t mygentoo:gnome-1.0 .
```

When completed checkout back to the **master** branch and run:

```
# docker build -t mygentoo:plasma-1.0 .
```

Then you can run either image using the same `mkg()` function in ~/.bashrc as above.

8.2.7 Reusing MKG Docker images

Images built as indicated above or released in the Release section can be reused in multi-stage builds as follows.
The following Dockerfile updates the image:

```
# name the portage image
FROM mygentoo:1.0 as build

# Use a current base stage3 image
FROM gentoo/stage3:amd64

WORKDIR /

# copy the entire root
COPY --from=build / .

# continue with image build ...
RUN emerge -auDN --with-bdeps=y @world
```

9. Safety and security

Building an operating system from scratch on a running computer is a process that may involve safety and security hazards. Several procedures and techniques have been developed to keep these hazards under control.

Keeping your running platform safe

Safety is generally thought of in terms of data integrity. Before running MKG on anything other than a container, a virtual machine or a live medium, it is generally a cautious move to backup the running platform.
Hundreds of successful (and some unsuccessful) builds show that the overall design of MKG, which delegates most of the building operations to VirtualBox machines, will keep your personal data away from potential risks, as most of the processes involved are encapsulated within virtual disks. This said, mistakes happen and damage to data can possibly be caused by altering the scripts or running them in inappropriate contexts.

Security: Protecting data from unauthorized access

Running software should request only the minimal rights necessary for execution. A good option is to leverage to power of Github Actions workflows and stay away from elevated rights. You can run MKG without elevated rights provided that you use the helper workflow releases (which are automatically downloaded by default) and avoid using options that involve `mount` or `chroot` operations

(like `hot_install`, `burn` or `test_emerge`).
You may wish to consider the following techniques:

- **Using Github Actions workflows**

  The present project workflow delivers a daily package, in the form of an ISO live CD, which comprises the minimal Gentoo install live medium together with the stage3 archive and the MKG build scripts. Releases are entirely automated by Github Actions from the public repository code, for both branches (master and gnome), and the output is signed and verified by Github. Checksums are available as release artifacts. By default, MKG uses Github Actions releases and checks control sums. The building process then unravels as indicated in the FAQ. VirtualBox machines can be run with user rights, so MKG can also be run with user rights in this case.
  To create a CloneZilla installer, you will likewise download an ISO live medium automatically released, signed and verified by the Github Actions workflow of companion site clonezilla_with_virtualbox.
  Although this may be performed automatically by MKG, users are allowed to keep control and replace the ISO live CD in the first VirtualBox machine with this one to create an ISO installer. This in turn can be burned to optical disk or copied to USB stick using `dd`. This last option is the only one that will require elevated rights to be achieved.

- **Rebuilding Docker images**

  You can crank up security levels (perhaps beyond what is necessary for most users) by rebuilding the above releases in a local fork of the above two repositories. You can also rebuild the current MKG Docker image using the Dockerfile in the **mkg_docker_image** repository.

You will not be able to create an installer ISO within the container, which is only possible if elevated access rights are granted to the docker engine (using option `--privileged`, see section 8.).
However with standard user rights, you should be able to create a valid Gentoo OS on a VDI virtual disk and fetch it back to the host to use it in a virtual machine or complete the processing.

- **Dropping Linux kernel capabilities**

  Even though it is necessary to run docker as root for MKG to work in containers, it is possible to better isolate container processes from the host by excluding a number of Linux kernel *capabilities* (see the corresponding man page), which grant fine-grained access to administrative rights.
  Tests have shown that all Linux kernel capabilities can be dropped, using option `--cap-drop=ALL` together with `docker run`. You can further restrict potential exploits by adding: `--security-opt=no-new-privileges`, which will bar containers from acquiring privileges that are not already set on launch:

```
# docker run -dit --security-opt=no-new-privileges --cap-drop=ALL \
-v /dev/log:/dev/log --device /dev/vboxdrv:/dev/vboxdrv  \
mygentoo:release-master
```

- Using **x11docker**

  Project `x11docker` enforces a handful of security features on top of docker and makes it possible to use graphic display within containers. Currently MKG does not make use of graphic display, yet installing `x11docker` may be worth considering if you wish to step up hardening. In addition to the above standard docker security options, `x11docker` adds a namespace remapping option, `--user=RETAIN` which conveniently remaps root processes within the container to host user (not root user) rights.

  Unlike S. Jordahl's original approach, which has inspired MKG dockerization, the current settings for MKG:

  – do not request sharing the host's network stack, i.e. do not rely on `--network=host`(sharing stacks may cause hazards);

- allow `x11docker` kernel capability dropping on startup, i.e. do not rely on `--cap-default`, thereby using the full range of `x11docker` hardening features;
- allow namespace remapping to host user, which lessens hazards associated with root processes within containers;
- do not request (re)building VirtualBox modules in the host, as both the host and container image have the same VB module configuration from the ground up (provided that the Gentoo host has been recently updated).

Using `x11docker` you may this way:

- log into your container (then run ./mkg and exit):

```
# x11docker --interactive --no-entrypoint --network=bridge --tty \
--user=RETAIN -- --device /dev/vboxdrv:/dev/vboxdrv \
-v /dev/log:/dev/log -- mygentoo:release-master /bin/bash
```

- or run the container in detached mode:

```
# x11docker --quiet --network=bridge --tty --user=RETAIN \
-- --device /dev/vboxdrv:/dev/vboxdrv -v /dev/log:/dev/log \
-- mygentoo:release-master >/dev/null 2>&1 &
```

To access the container created by the above command line, use:

```
`# docker exec -it $(docker ps -q | head -1) bash`
```

then within the container, change directory to **/mkg**.

Current limitations

Virtual Machines embedded within containers will currently refuse to start if docker is run in experimental rootless mode or even if namespace remapping is used to remap root processes within the container to a non-root ad-hoc test user.
With this in mind, combining containerization, nested virtualization, kernel capability-dropping and (optionally) `x11docker` hardening features should prevent most likely hazards and keep data integrity and platform security in line with reasonable expectations.

10. Frequently Asked Questions (FAQ)

- Can MKG be run with standard user privileges?

- I'm using the default third-party workflows and my command line options do not seem to be taken into account!
- On which platforms does MKG work?

- Do you support server versions? Gnome desktops? ~amd64 branches? other profiles?

- How to create a bootable ISO to install Gentoo to main PC disk?
- How to get some help?
- Are you advocating an "Install and forget" approach?
- So what kind of support should I expect?
- What is the difference between MKG and Ubuntu or Fedora installers?
- How much time does it take to build Gentoo?
- Waow, 22 hours is a lot of time: can I somehow be kept informed of job completion?
- Can you run MKG detached in the background and be warned of completion?

- Pending completion, how can I monitor if all goes well?
- After completion, how can I check if all went well?
- How much CPU power will I need?
- How much RAM and disk space will I need?
- Building Gentoo sometimes runs into failures: will mkg complete its build?
- Do you provide pre-built Gentoo ISO installers?
- What is the difference between a minimal build and a full build?
- How to create an install medium?
- My build failed but I fixed the issue and my VDI disk was created: how can I proceed?
- I have limited CPU power: what can I do?
- What dependencies should be installed?
- Is mkg safe for my platform?
- How can I be sure the released binary distributions are safe?
- Can I use workflows to build Gentoo using MKG?
- Are there other cloud computing options compatible with MKG?

- How many MKG jobs can be run on a given computer?

Can MKG be run with standard user privileges?

MKG can be run with standard user privileges (denoted as `$`), unless test mode is used, or an operating system is directly installed on a block device, or the default third-party **clonezilla_with_virtualbox** workflow is not used (`use_clonezilla_workflow=false`). In these cases, we use the standard denotation `#`.
More specifically, you can run MKG with standard user privileges if you do not use the following options:

- `burn` (in some rare, hardware-specific cases)

- `build_virtualbox`

- `from_device`
- `hot_install`
- `test_emerge`
- `use_bsdtar=false`
- `use_clonezilla_workflow=false`
- `use_mkg_workflow=false`

I'm using the default third-party workflows and my command line options do not seem to be taken into account!

Unless `use_mkg_workflow=false` is specified on command line, presets are used by the Github Actions workflow that creates the VirtualBox live CD used in the build. The following options are therefore deactivated in this case, at least for the part of the execution that takes place within the VirtualBox machines:

`bios, cflags, clonezilla_install, debug_mode, elist, emirrors, kernel_config, minimal, minimal_size, ncpus, nonroot_user, passwd, processor, rootpasswd, stage3, vm_keyboard, vm_language`

On which platforms does MKG work?

`mkg` has been developed under GNU/Linux.
It should run on all GNU/Linux platforms with appropriate installation of dependencies.
It should be portable to the BSD family of *nix platforms (including MacOs) with some amendments. A port should use GNU versions of common tools like `sed` and `grep` for example and so relink such tools.

Do you support server versions? Gnome desktops? ~amd64 branches? other profiles?

This is a desktop distribution for the stable branch (with a few ~amd64 packages).
MKG currently supports Plasma and Gnome: checkout the appropriate branch.
For hardened profiles you should use `stage3_tag=hardened` on command line, from the git branch corresponding to the desktop of choice (master for Plasma or gnome for Gnome). OpenRC is the default init system. Use `stage3_tag=systemd` for (non-hardened) systemd. Hardened profiles are OpenRC-only (there is no systemd hardened profile available on Gentoo).

How to create a bootable ISO to install Gentoo to main PC disk?

Simply run:

```
# ./mkg
```

to create **gentoo.iso**, or add the (relative or absolute) filename to command line.
You will need a working internet connection and a handful of dependencies (see below), which it is your responsibility to install.

How to get some help?

One of:

- `# ./mkg help` (console output)
- `# ./mkg help=md | pandoc - | /usr/local/bin/bcat` (html output, using `pandoc` and the Ruby gem `bcat`)
- `# ./mkg help=md | pandoc -V margin-right=1cm -o help.pdf && okular help.pdf` (pdf output, using `pandoc` and `okular`, replace with your pdf reader of choice)
- This Wiki, which documents the stable version. For git repository developments, prefer the above three ways, as they use the software help itself.
- For a full list of software options check text file **options** at the root of the git repository, with added details on option types.
- For a detailed documentation of the API check the Doxygen documentation in HTML format or PDF format.

Are you advocating an "Install and forget" approach?

Not in the least. This is a user-contributed bootstrapping tool that is there in the hope that it will be useful at least to some users, be they experienced or not. Inexperienced users are welcome to use the tool if it gives them a taste of Gentoo freedom, but they should be aware that:

- not all graphic cards can be supported,
- only a limited set of desktops will ever be (currently only plasma),

- it is their responsibility to tweak the kernel to their needs.

Users who need further help on how to install further packages or capabilities, or e.g. portage dependency conflicts, should first and foremost consult the Gentoo installation manual then, if they do not find their way out, browse the Gentoo forums and possibly request some (**non**-MKG related) help there.

So what kind of support should I expect?

Support will be restricted to MKG itself, if the targets do not build or install for the supported platforms and amd64 desktop profiles.

What is the difference between MKG and Ubuntu or Fedora installers?

MKG is more of a bootstrapping (or a recovery) tool than a fully-fledged installer, for the most common user platform (Intel x86_64, Haswell+). For other profiles, you are on your own.
Further, numerous parameters are set to common international C-locale values, with the one exception of the language and keyboard layout (parameters `vm_keyboard` and `vm_language`). To fine-tune locales, time zone, wifi and networks, and non-Intel graphic support, please follow the Gentoo installation manual.
This approach makes it possible to strike a balance between the need for a quick recovery tool and user freedom and experience building.
Also, it has one concrete and solid advantage: speed. Using a high-transfer speed USB stick as an installation medium, to an SSD main disk, installation completes in about 3 minutes on my Core-i7 laptop. And unattended. This is time saved by an order of magnitude compared to Ubiquity/Anaconda installers.

How much time does it take to build Gentoo?

It obviously depends on your hardware. A full platform builds in 15 to 22 hours on my Core-i7 (Haswell) x86_64 platform. You should not shut down or hibernate your platform in the meantime.

Waow, 22 hours is a lot of time: can I somehow be kept informed of job completion?

Yes, you can: just add, for example:

```
# (...) email=john.smith@emailprovider.domain email_passwd="my_password"
```

to your command line.
You will have to authorize so-called 'insecure' software with some providers like gmail.
Password is not encrypted. However, `curl`, the software used to send mail, is reasonably secure, provided that the message is sent:

- from a private computer to its owner,
- without connecting to a public network.

Can you run MKG detached in the background and be warned of completion?

In GUI-less mode you can run MKG in the background detached from your console.
Just run:

```
# nohup [your MKG command line] gui=false &
```

Currently you can only be warned using the email procedure indicated above.

Pending completion, how can I monitor if all goes well?

Before starting you should first clean up your logs and restart `sysklogd`. Under Gentoo with `open-rc`:

```
# rc-service sysklogd stop
# [backup selected /var/log/syslog* files and clean up]
# rc-service sysklogd start
```

You should also make sure that your `syslog` is not rotated too often: check your **syslog.conf** parameters and (if installed) `logrotate` configuration (see the Gentoo manual).
If you rotate daily, it is better-advised to switch to weekly; otherwise your plots might be pruned in the middle of the building process (see below).

If your syslog has already rotated, usually there will be backups under /var/log. Decompress these backups:

```
# gunzip /var/log/syslog*gz
```

and, in what follows, replace `/var/log/syslog` with `/var/log/syslog* | sort -g`.

A simple monitoring function can be defined in your ~**/.bashrc** as follows:

```
alias mong="tail -f -n10000 /var/log/syslog | grep -E  '\[[A-Z]{3}\]'"
```

Source your ~**/.bashrc** then call `mong` from time to time or in a monitoring console.

Below is an alternative monitoring method (there are many others around):

```
$ cat /var/log/syslog | logtool -o csv | emacs --insert <(cat)
```

You will need `syslogd` and `logtool`. Then in Emacs, I use `csv-mode` + `csv-align-mode`.
You can switch CSV columns with a TAB stroke and filter using Occur.
I usually check [...] tags first:

```
M-s o RET  \[[A-Z]\{3\}\]
```

These are special MKG tags.
Vim users will probably prefer opening `/var/log/syslog` in vim and, within it, strike:

```
/\[...\]
```

Alternatively you can view `/var/log/syslog` in a web browser:

```
$ cat /var/log/syslog | grep -E \[[A-Z]{3}\] | logtool -o html | bcat
```

You will need the Ruby gem `bcat`for this. If you'd rather not use it, dump to a temporary file:

```
$ cat /var/log/syslog | grep -E \[[A-Z]{3}\] | logtool -o html > tmp.html 2>&1 && (your-browser) tmp.html &&
rm -f tmp.html
```

Unless you want to check environment events that may have affected the job, the above monitoring methods are about useless if you ran the job using `nohup`.
In this case, just display the `nohup.out` file in the `mkg` root directory.
A nice way of processing the log data is to plot the VDI disk size as follows:

```
$ cat /var/log/syslog | awk '/\[[A-Z]{3}\]/ {print $11}' \
| grep '[0-9,]G' | sed 's/G//g' | tail -n 180 > tab \
```

```
&& Rscript -e 'X11();plot(scan("tab", dec=","), type="l");Sys.sleep(30)'; \
rm -f tab
```

You will need `awk` and `R` for this. For non-Latin locales, the decimal separator has to be changed into a dot as below:

```
$ cat /var/log/syslog | awk '/\[[A-Z]{3}\]/ {print $11}' \
| grep '[0-9.]G' | sed 's/G//g' | tail -n 180 > tab \
&& Rscript -e 'X11();plot(scan("tab"), type="l");Sys.sleep(30)'; \
rm -f tab
```

If using `nohup`, replace `/var/log/syslog` with `nohup.out`. Change 180 into the number of minutes of monitoring you want to plot. Below is the output of this (long) one-liner:

The lower part of the S-curve will be shorter with higher-end processors. Below is the curve obtained on an AWS EC2 c5.metal instance with 15 cores and 30GB of RAM:

MKG offers a quicker alternative to plotting disk size with the above commands, if GNUPlot is installed on your computer:

```
# [your MKG command line] plot
```

The plot may be customized for color, position, periodicity, display time and time span.
See command line help for details.

Hazards and failures can often be graphically detected this way.

After completion, how can I check if all went well?

After completion, you may run:

```
# ./mkg share_root=r vm=[name of your VM, without vdi]
# cd /vdi && cat res.log && cd -
# ./mkg disconnect
```

Or, more automatically:

```
# ./mkg vm=[name of your VM, without vdi] exitcode no_run
```

Alternatively, you may add `exitcode` to your command line before you run it.
There will be a message, towards the end of the tagged log, like:

```
[MSG] Virtual Gentoo build exited with code: 0
```

or another exit value if an issue occurred. See table of exit values in section 3 of this Wiki (bottom).

How much CPU power will I need?

Building can be performed with single-core, older-generation Intel x86-64 processors. Should you wish to use higher-end cloud computing platforms, bare in mind that this option may be neither cost-efficient nor computationally optimal. Computing time does not go down linearly with the number of cores. This is notably caused by the numerous bash script configuring steps that come in between compilation phases. Also, smaller software with a limited number of source code file will not use all available cores. Parallelism is mostly put to good use when building the kernel or larger software like gcc, clang, libreoffice or webkits. The graph below shows this quite well. It has been obtained while building 6 Gentoo desktops (3 Gnome and 3 Plasma each), as parallel jobs, using 90 virtual cores on an AWS EC2 c5.metal instance (196 GB of RAM). Only for short periods is the power of the platform fully utilized:

Compare this with a single job on a 2-core, 30GB virtual VM.Standard2.2 instance on Oracle cloud (free trial package):

Although the latter job took much longer to complete, balanced CPU utilization was on average much higher.
The bottom line is that it may be overall a better option to use clusters of machines with 2 to 4 cores each than higher-end multicore instances.

How much RAM and disk space will I need?

Building Gentoo comes at a price: you will need a minimum of 7GB of available RAM and 55 GB of free disk space (twice this amount if you directly install to a device using the safe-and-slow option `cloning_mode=with-raw-buffer`).

Building Gentoo sometimes runs into failures: will `mkg` complete its build?

`mkg` is routinely tested and comes with a `minimal` option that builds a lighter version of Gentoo, which may help if a recent ebuild tree breaks something in the full build.
In case of a build failure, please file a Github issue.
Please check the Debugging reports Wiki page

Do you provide pre-built Gentoo ISO installers?

ISO installers will be provided in the **Release** section of this Github site.
Currently, builds are restricted to the x86-64 platform.

What is the difference between a `minimal` build and a full build?

Using `minimal minimal_size` on command line you will cut down on installed software and remove the kernel sources (which should be reinstalled later on) to create a more compact installer.
A full build adds in the following packages and their dependencies:

- System: dev-vcs/git logger (
- Data science and editors:
  `dev-lang/R`
  `app-editors/emacs`
  `app-editors/vim`

- Qt:
  `qtwebengine`
  `qtpositioning`
  `qtopengl`

- Office
  `app-office/libreoffice`

- PDF creation
  `=dev-texlive/texlive-latex-2020`
  `texlive`

In addition, a set of commonly-used R libraries are installed out-of-the-box
(notably `data.table`, `ggplot2`, `rmarkdown`, `dplyr`, `devtools`,`bit64` and their dependencies).

How to create an install medium?

- directly install Gentoo to my high-transfer rate USB stick or external disk?

```
# ./mkg hot_install ext_device=sdX
```

where /dev/sdX is your external device (check `fdisk -l` or `blkid` to make sure).

- create a Gentoo ISO recovery medium out of my existing Gentoo platform?

You should not be running the platform (so-called *hot backups* are not supported). Supposing your Gentoo platform to be backed up is under **/dev/sdX**, enter:

```
# ./mkg from_device ext_device=sdX gentoo.iso
```

This will create an iso file named `gentoo.iso` which can automatically reinstall your Gentoo OS onto a target PC. Important note: the target partition **will always be /dev/sda**. Withdraw other disks from the target computer either using BIOS or (preferably) physically.
Always back up before installing a new OS!

- Burn my ISO installer to disk?

Use your favorite disk-burning tool or, if you have installed cdrecord (from cdrtools), just run:

```
# ./mkg from_iso gentoo.iso burn
```

- ... and simultaneously create a USB-stick recovery medium?

Run, if your USB stick has been mounted to `path`:

```
# ./mkg from_iso gentoo.iso burn device_installer ext_device=path
```

Instead of `path` you can use the device KNAME (say `sdd` for example) or the device VENDOR (like `SanDisk` or `Kingston`).

- What if several disk burners are installed?

If your computer has several optical disk burners, run:

```
# cdrecord -scanbus
```

then select the SCSI address x,0,0 of choice int the output of this command and run for example:

```
# ./mkg from_iso gentoo.iso burn scsi_address=6,0,0 device_installer ext_device=SanDisk
```

My build failed but I fixed the issue and my VDI disk was created: how can I proceed?

- to create an ISO installer?

Just run:

```
# ./mkg from_vm vm=Gentoo gentoo.iso
```

with your VM name (*without* .vdi) instead of `Gentoo`

- to create a USB-stick installer?

First check the output of `fdisk -l` to identify the device of choice, say `sdX`. Then run:

```
# ./mkg from_vm vm=Gentoo device_installer ext_device=sdX
```

with your VM name (*without* .vdi) instead of `Gentoo`

I have limited CPU power: what can I do?

You can still use `mkg`, provided that you have enough RAM. To keep allocated resources low, add the following to your command line (you may adjust parameters depending on your needs):

```
# (...) ncpus=1 cpuexecutioncap=50
```

This will force `mkg` to run on just 50% of one processor core.

What dependencies should be installed?

You can run `mkg` and wait for it to complain about missing dependencies in the first stages of the run. If you prefer to avoid any dependency issue, you should check that your platform has a GNU version of the following tools:

- `bash` (any recent version)
- `curl`
- `dos2unix`

- `findmnt`
- `grep`
- `lsblk`
- `md5sum`
- `mkisofs` (from cdrtools)
- `mksquashfs` with xz-compression support (`USE=lzma` on Gentoo)
- `mountpoint`
- `rsync`
- `sed`
- `sha512sum`

- `uuid` or `uuidgen` (build tarball in repository if a system version is not unavailable)
- `tar`
- `VBoxManage` (from virtualbox)
- `xz`

and optionally:

- checksum tools: `sha1sum`, `sh256sum`, `sha512sum` **and** `b2sum` (BLAKE2b sum), if an ISO installer is to be generated.

- `bsdtar` (same case, by default, unless `use_bsdtar=false`. This executable is packed with `libarchive`)

- `xorriso` (same case)
- `cdrecord` (from cdrtools, to burn a disk)

- `docker` and optionally `x11docker` (if running MKG within a container)

- `guestfish` (to directly install Gentoo to a device without creating an installer)
- `qemu` (for debugging by mounting the virtual disk to shared directory)
- `at` (same case)
- `gnuplot` (top monitor building)

To back up an existing partition into a Gentoo ISO installer, it is preferable to have the CloneZilla suite (`ocs-sr` and `drbl` notably) previously installed on the platform. Otherwise `mkg` will try its best but this may fail more easily.

Is `mkg` safe for my platform?

`mkg` builds a Gentoo image using a VirtualBox VM. So most of the potential hazards are encapsulated within a virtual disk away from your OS system files. However care should be taken when installing Gentoo to a device. Check that your device is free or may be overwritten without loss. It is a common user responsibility to check this. `mkg` will request two confirmations in a row. You have been warned.

How can I be sure the released binary distributions are safe?

If you are targeting maximal security, please allow for one day's processing and build from source by running `./mkg`.
Otherwise, builds are released with published checksums. Please verify these checksums before installing your distribution:

```
$ md5sum [b2sum|sha256sum] *.iso
```

Guaranteeing 100 % security of released binaries is an almost impossible objective but there are reasonable ways for you to obtain independent assurance that the releases are clean:

- by testing select critical system packages using Gitian.
- by rebuilding your core GNU applications (e.g. `gcc`, `binutils`, `coreutils`, `glibc`) from GNU source code.
- by rebuilding your platform from Gentoo sources: this is quicker than running MKG directly and offers a reasonable security middle ground:

```
# qlist -I > packagelist
```

[check and possibly adjust your `packagelist` file at your own risk]

```
# emerge -e $(cat packagelist)
```

This will completely rebuild your platform from scratch.
You make take advantage of this step to fine-tune /etc/portage/make.conf.
Notably, you may want to increase the MAKEOPTS value and adjust L10N, VIDEO_CARDS and USE values.

Can I use workflows to build Gentoo using MKG?

On Github, only partially as of March 2021, as nested virtualization is not supported by this platform.
However this is possible using Travis CI or Amazon EC2 (C5 bare metal instance). With 90 vCPUs and 180 GB split into 6 paralell jobs, a c5.metal instance takes 13 hours to complete 3 full plasma and 3 gnome desktops (with different `cflags` settings for each).

It is nevertheless possible to use *Github Actions* workflows to test whether there are no portage tree conflicts, considering the list of packages to be installed (file **ebuilds.list.complete** or **ebuilds.list.minimal**), and the USE and ACCEPT_KEYWORDS settings in files **ebuilds.list.use** and **ebuilds.list.accept_keywords**. Consult the Testing and Debugging wiki page for further details.

As of March 2021, MKG has been leveraging the power (and safety) of Github Actions by delegating part of its build process to the companion project clonezilla_with_virtualbox. The ISO created by *Github Actions* is automatically released under a tag in the Release section of this project. MKG downloads this automatic, third-party release in the course of building the VM that converts the virtual disk into a CloneZilla image. This default behavior can be disabled by adding `use_clonezilla_workflow=false` to command line.

A similar procedure also applies to the minimal Gentoo install ISO. MKG scripts and the stage3 archive are added within its squashfs filesystem by the *Github Actions* workflow of the MKG Github site. An ISO file labelled **downloaded.iso** is automatically released by the workflow. It will be downloaded from the MKG Github release section automatically. To disable this behavior you can add `use_mkg_workflow=false` to command line.

The use of *Github Actions* workflows, although partial, positively contributes to safety, convenience and speed. It also makes it possible for the user to run MKG in its *virtual instantiation* only. In this usage, MKG does not run on your platform but only within a virtual machine over which you keep full control.

This way of running MKG requests more input from the user but avoids all but potential security issues. Artifacts that are to be used are exclusively built by automated Github Actions workflows on a regular basis from GPG-signed commits in Github projects **mkg** and **clonezilla_with_virtualbox**. User input is limited to creating one (or two) VirtualBox machines, setting their parameters and firing them. Please consult the comments and installation advice in the master Release section for Plasma desktops and the gnome Release section for Gnome desktops.

Are there other cloud computing options compatible with MKG?

Free cloud computing alternatives are offered by Oracle cloud and successful builds have been obtained this way. You will need either a bare metal or a 2.2 instance of the following minimal characteristics (in the free trial package):

Image:
Canonical-Ubuntu-20.04-2021.02.15-0
Launch Mode:
NATIVE
(. . .) Shape Configuration
Shape:
VM.Standard2.2
OCPU Count:
2
Network Bandwidth (Gbps):
2
Memory (GB):
30
Local Disk:
Block Storage Only
(. . .)
Firmware:
UEFI_64

Also specify about 80 to 100 GB of boot disk.
Successful builds used an Ubuntu 20.04 image with the following preliminary upgrade:

```
# apt update && apt upgrade
# apt install uuid libguestfs-tools squashfs-tools curl mkisofs \
cdrecord util-linux xorriso xz-utils virtualbox virtualbox-ext-pack
```

Once the instance has been created and the platform software installed, run an MKG job with adjusted value for `ncpus`. Currently the number of virtual CPUs allowed in the free package is 2 for bare metal instances, so specify on command line:

```
# ./mkg (...) ncpus=2 cflags=\'[-O2,-march=core2]\'
```

Adjust the `march` parameter to your processor characteristics (*note the list format*). Do **not** use `native` as this would tune the build to your provider's processor.
This job takes about 6 days to complete a full plasma desktop.

How many MKG jobs can be run on a given computer?

Each MKG job will request at least as many CPUs as specified. By default the third of core threads are reserved for each MKG job. Otherwise, if this option is set on command line, the job will be allowed `ncpus` cores. Depending on your processor, these *cores* may actually be *threads* rather than physical cores. You may check this by comparing the number of physical cores with the output of `nproc --all`.

Supposing for example the target environment runs 6 cores allowing 12 threads, you may run 1 standard default MKG job (4 cores) and 2 default container-based jobs (1 core each). Beyond this limit, nested VirtualBox processes, after some running time, have been experienced to crash for want of available CPUs.

A way to circumvent this limit is to throttle CPU power allowed to each job by creating pods in a local Minikube cluster. Each pod workload will have its workload tuned and balanced according to memory and CPU caps set on startup. Using Docker images and firing up k8s using the Docker (or the bare metal) driver is then the better option, as this will avoid adding an extra k8s-specific vitualization layer on top of VirtualBox machines running within Docker containers.

11. MKG Cheat Sheet

Running MKG with custom options	
`# mkg use_mkg_workflow=false [...]`	Do not use preprocessed live install CD from Github Actions workflow. You may notably use: `ncpus=X, bios, cflags, clonezilla_install debug_mode, elist, emirrors, kernel_config, mem, minimal, minimal_size, ncpus, nonroot_user passwd, processor, rootpasswd, stage3_tag vm_language.` Main options: `minimal`: just build a minimal desktop. `cflags=\'[...,...,...]\'`: CFLAGS options in list form. `vm_language=..`: set platform language if non US-English (`fr`, `de`, etc.)

Running MKG with custom options

`# mkg use_clonezilla_workflow=false [...]`	Do not use preprocessed cclonezilla live CD from Github Actions workflow. Rebuild this CD again incorporating VirtualBox from current Ubuntu repositories.

Supported desktops, init systems and profiles

`$ git checkout [gnome or master]`	Switch to Gnome/to Plasma
`# mkg use_mkg_workflow=false stage3_tag=[hardened or systemd]`	Switch to stable hardened profile / to gnome or plasma systemd profile

Input/Output and Backup options

`$ mkg [...] burn`	Burn Gentoo installer to DVD when processed.
`# mkg [...] hot_install ext_device=sdX`	Install Gentoo onto partition **/dev/sdX** after completion of VM processes.
`# mkg from_device ext_device=sdX gentoo.iso`	Backup partition **/dev/sdX** into a CloneZilla installer **gentoo.iso**
`# mkg [...] from_iso gentoo.iso burn`	Burn **gentoo.iso** to disk.
`# mkg [...] from_iso gentoo.iso device_installer ext_device=sdX`	Create USB stick or any block device installer from **gentoo.iso**
`# mkg [...] from_vm vm=... gentoo.iso`	Create CloneZilla installer image from VM (after VM completed processes and stopped.)
`# mkg [...] from_vm vm=... hot_install ext_device=sdX`	Directly install Gentoo to partition **/dev/sdX** from VM (after VM completed processes and stopped.)

Running MKG in containers

`# mkg dockerize [gentoo.iso ...]`	Running MKG in a Docker container. Followed by command line. Fetches back ISO installer if any.
`$ git checkout gnome`	
`# mkg dockerize [gentoo.iso...]`	Running MKG in a Docker container for Gnome.
`# docker exec -it ID bash`	Check job log
`cont# tail -f nohup.out`	
or:	
`grep -E '\[\w{3}\]' /var/log/syslog`	Host log search. Also echoes container logs. Use `syslog.x` for older logs.

Running MKG in containers

`# docker cp ID:/mkg/gentoo.iso .`	Fetch back MKG installer from container.

Graphic display and Interaction

`$ mkg [...] gui=false`	Do not display VirtualBox guest in GUI.
`$ mkg [...] interactive=false`	Do not interact with user. To be used in scripts and containers, with caution.
`$ mkg [...] email=...@... \ email_passwd=...`	Send a meesage to email address with given user password upon completion. Not to be used in public networks.

Reusing previously downloaded artifacts

`$ mkg custom_clonezilla=file [...]`	Use this file as CloneZilla live CD.
`$ mkg download_clonezilla=false [...]`	Use cached CloneZilla live CD from prior downloads.
`$ mkg download_arch=false [...]`	Use cached stage3 archive from prior downloads.
`$ mkg download=false [...]`	Use cached Gentoo minimal install CD from prior downloads.