MANUAL

Table of Contents

cinac(1) – Generic infrastructure installation and management system

ACKNOWLEDGMENTS

This project is in BETA. It is not intended for continues use in production. Neither has this projects gone through a rigorous testing or security audit.

With that said it should be stable enough and safe to use in a lab or offline environment. Use at your own risk and don't hold me responsible.

I continuously implement new features and change the code-base. So differences in releases will probably break functionality. I will release a stable version once I'm happy with the code-base, testing and have good documentation.

DESCRIPTION

To make your life easier, here is a dictionary of terms used in this manual:
Servermonkeys IT-dictionary for dummies

The goal of CinaC is to be a generic IT-infrastructure installation, configuration and management system with focus on immutable hosts. It is mainly build up on Debian, libvirt and Ansible. It takes ideas from Ansible Tower, ESXi, OpenStack and DebianLAN. The basic concept is to use Infrastructure as code in form of 'blueprints'. CinaC tries to follow these philosophies found here:
ServerMonkey Software Philosophy

CinaC is a recursive backronym that stands for 'CinaC is not a Cyberrange'.

It focuses on minimalism, modularity and documentation. It makes extensive use of the shell for execution, simple CSV and INI style text files for configuration, SSH for management and the Debian packagemangement system for installation. CinaC actively avoids web applications, agents, databases, microservices and a monolithic software architecture. CinaC is pretty strict on how to do things. For example CinaC only uses libvirt, you can't choose between libvirt or VirtualBox. The reason is simple: Less configuration overhead. Having fewer choices is a feature here.

CinaC consists of several APT packages that are currently distributed via my private server and can also be found on GitHub. CinaC bundled together with Debian can also be considered a Derived Debian Distribution.

What you can do with it:

- Build IT-networks. Examples: Home lab, School network, Virtual training platforms for IT-security that can be build and destroyed on demand. You can build and manage networks that are entirely virtual, entirely physical or mixed.

- Remote kiosk applications. Examples: Thin clients that connect to remote virtual-host, like Storefront display monitors or a teacher-student IT-education platform.

FEATURES

- In CinaC, the environment (IT-infrastructure) to install and configure is called a 'blueprint' and is defined only by code inside a single folder. All blueprint code is made of CSV, INI style configurations, Ansible Playbooks and libvirt XML files.
- All installation and configure of software (aka. deployment and provisioning) happens via SSH. This makes it possible to combine physical and virtual machines in the same environment. This also enables management of an existing infrastructure.
- Modular. CinaC is made of mostly CLI applications. Which makes it easy to build your own solution. For example: Only installing the APT package 'cinac', creates a virtualization platform with management capabilities. First by installing the 'autokiosk' package it turns into a graphical kiosk application.
- Kiosk mode gives easy access to machines via a simple front-end and also allows you to integrate your own web applications.
- Virtual image deployment happens by using disk chains. Meaning that you can use a single Golden Image disk file and deploy it to multiple machines. Only new disk data will be written to a separate disk file. This increases deployment time. No need to copy disk images.
- Third party software and file sourcing/installation is provided via the fastpkg package manager. This includes Golden Images, custom software for Windows or any other kind of files from web-sources, like Wallpapers or zip files.
- Offline usage. Once a blueprint has been deployed, it can be destroyed and redeployed offline. No need for an internet connection.
- Some Golden Images and Demo blueprints are included. This includes playbooks, configurations and configuration scripts.

CinaC combines the following programs to realize its goals:

- fastpkg : file and software sourcing and installation
- inventorymaker : SSOT inventory management
- vmh : VM deployment
- wildwest : command and control
- ansible-roles : provisoning and configuration
- autokiosk : end user front-end
- spice-web-client : Connect to VMs via web-browser

COMPARISON

A comparison with other tools can be found at: github.com/ServerMonkey/cinac/docs/comparison.md or in /usr/local/share/cinac/docs/comparison.md

REQUIREMENTS

For first time users a single PC, Laptop or server is recommended.
You can even try nested virtualization, but you need a full virtualization capable hypervisor.
Minimum hardware requirements for the Demo blueprints:

- CPU : single 2 GHz x86 CPU with 4 cores (8 cores for larger demo environments and faster deployment)
- RAM : DDR3 with 4-8 GiB min. (16 GiB if you want to test all demo environments)
- DISK : Single SSD with 200 GB of free space
- x86 virtualization : CPU should be capable of VT-x or AMD-V and the x64 instruction set

All other hardware requirements are based on what blueprints you want to develop and use.

For the OS, install a fresh copy of Debian 11 or 12
Recommend following the Stable or Proposed release cykle.

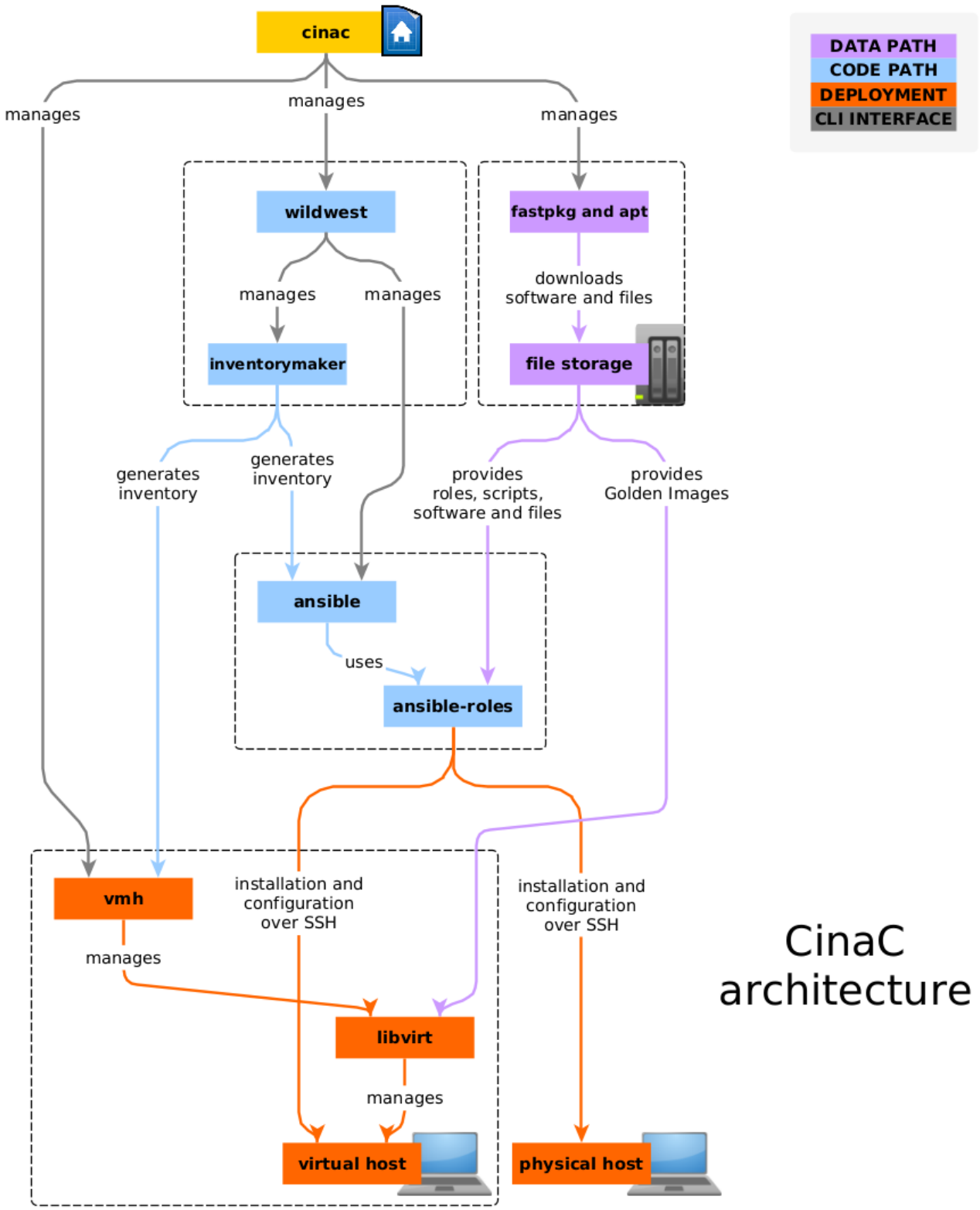

Figure 1: Architectural overview:

INSTALLATION

Single instance

You can compile all code yourself, but I recommend using my public Debian 11 repo: github.com/ServerMonkey Add this repo first. Then install the basic headless server with:

```
$ sudo apt install cinac
```

Or to install everything, including a full desktop environment, you can install the meta package:

```
$ sudo apt install cinac-full
```

This will install:

- cinac
- autokiosk
- spice-web-client
- nginx
- pulseaudio

Then run the following command and follow the graphical terminal instructions. For a quick-start, first time users in a lab environment can just push enter on all questions.

```
$ cinac-init
```

To run 'cinac-init' unattended, export some or all of the following variables before you run 'cinac-init':

```
# These are the default settings (same as pressing enter on each question)
export CINAC_PYARG=true  #false
export CINAC_GPGAGENT=merge  #ignore,replace
export CINAC_KEY=existing  #new,skip
export CINAC_MAIL=cinac@localhost
export CINAC_NAME=CinaC
export CINAC_PASSWORD=none
```

Firewall

cinac-init will also enable certain firewall rules.
The rules allow SSHD, HTTP, HTTPS, SPICE-WS/WSS on all physical and WI-FI interfaces but not on any virtual interfaces. This is to prevent accidental or malicious access to the virtualization platform from within the virtual network.

DNSMASQ is enabled on all interfaces. This is to allow DHCP and DNS resolution for the virtual machines.

The allowed SPICE ports are 7200 to 7300 for unencrypted (WS) and 8200 to 8300 for encrypted (WSS) connections.

To see all firewall rules run:

```
sudo ufw status numbered
```

Every time you run cinac-init the firewall rules will be reset. So if you want to add your own rules, you need to add them each time after running cinac-init.

SPICE web client

To connect to the virtual machines via a web-browser, you need to install the spice-web-client and a webserver.

```
$ apt install spice-web-client nginx
```

To enable encryption for SPICE over HTTPS+WSS, you need to create a certificate for the CinaC host. You can create a simple self-signed certificate but this will not work wil all browsers. For example Firefox requires a full root-authority chain. Chromium based browser can just add a self-signed certificate as exception.

Put your certificates in the folloing location:

Certificate: /etc/ssl/certs/cinac.crt
Private-key: /etc/ssl/private/cinac.key

Autokiosk

To enable the autokiosk front-end, you need to install the autokiosk package:

```
$ sudo apt install autokiosk
```

After installation reboot and rerun 'cinac-init' to enable autokiosk.

To open a VM in autokiosk (virt-viewer), run:

```
$ autokiosk <VM_NAME>
```

In virt-vier mode the VM will hog you entire keyboard and mouse. To exit virt-viewer mode, press 'CTRL+ALT' once and then press another of the following key combinations to leave virt-viewer mode:

'ALT+TAB' to switch between windows.
'CTRL+SHIFT+F9' to minimize only all VM autokiosk windows.
'CTRL+SHIFT+F10' to minimize all autokiosk windows.
'CTRL+SHIFT+F12' to minimize all autokiosk windows and start 'cinac tgui'.

It is important that you press the correct key combination after 'CTRL+ALT'. Else the virt-viewer assumes you want to send the key combination to the VM.
This can be trycky if you push the keys too fast, slow or in the wrong order.
Try it a couple of times to get the hang of it.

The next step requires the package spice-web-client and nginx.
To open a VM in autokiosk over HTTP+WS, run:

```
$ autokiosk http://<CINAC_HOST>/<VM_NAME>
```

The next step requires a fully working root certificate chain. A single Self-signed certificate will not work with the package simple-kiosk.
To open a VM in autokiosk over HTTPS+WSS, run:

```
$ autokiosk https://<CINAC_HOST>/wss.<VM_NAME>
```

Multiple hosts / cluster / server farm

If you have only a couple of hosts, you can install via an automated bootable USB-stick. Follow this guide: Ventoy with Debian Preseed and fastpkg

BASIC USAGE

After installation and cinac-init you need to get some blueprints.
Demo blueprints can be installed via:

```
$ cinac install-demo-blueprints
```

List all available blueprints:

```
$ cinac list-available
```

Load a blueprint:

```
$ cinac-load <BLUEPRINT_NAME>
```

List currently running blueprint:

```
$ cinac list-available
```

You are only allowed to load one bluprint at a time.
To unload any blueprint:

```
$ cinac unload
```

Or run the Terminal GUI for a graphical interface:

```
$ cinac tgui
```

UNDERSTANDING BLUEPRINTS

Once you have installed the demo blueprints, you can open the readme file in ~/.cinac/demo-tiny/README.md . If you haven't installed them yet, they can be found in /usr/local/share/cinac/demo-blueprints/. The demo blueprints contain additional documentation that you can explore further. First try to understand 'demo-tiny' then 'demo-home' and last 'demo-company'. Each of the demo blueprints will teach you a bit more.

ROLES

If you work as a team, seven basic work roles can be associated with the development and usage of CinaC. The roles can be assigned to different people or be done by a single individual. This list exist to get a better understanding of how CinaC works and what skills are required to use each part of it.

Backend administrator

Builds and maintains the hardware and software required to run CinaC.

- Required skills:
  - Debian 11/12
  - APT
  - GPG
  - SSH
  - Server hardware
- Recommended literature and courses:
  - Debian Administrator's Handbook
  - Debian Reference
  - Debian Wiki
  - SSH Mastery (M. W. Lucas)
  - udemy.com - Linux Administration: The Complete Linux Bootcamp for 2023

Repo maintainer

Maintains a list of software and files that will be used in blueprints

- Adds new software or files to the repository
- Required skills:
  - Debian package system
  - How to build Debian packages
  - Set up a Debian repo server
  - basic understanding of sha256 hash
  - basic understanding of SSL/TLS
  - fastpkg
  - target OS specific knowledge (example: Windows or Ubuntu)
    - Silent installer command-line arguments
    - File and Executable file formats

Golden Image developer

Creates custom libvirt operating system images.

- Required skills:
  - libvirt XML format
  - qcow2 format
  - virtio
  - Cygwin
  - SSH, SSHD
  - virsh
  - virt-manager
  - vmh
  - target OS specific knowledge

Blueprint developer (inventory)

Creates and maintains a list of hosts

- Required skills:
  - libvirt XML format
  - target OS specific knowledge

- target OS authentication structure of different OS:es, like: sudo, su or SYSTEM/Administrator roles on Windows
- CSV format
- LibreOffice Calc

Blueprint developer (configuration)

Writes scripts and playbooks that installs software and configures the target hosts.

- Required skills:
  - libvirt XML format
  - Ansible
  - udemy.com: Dive Into Ansible - Beginner to Expert in Ansible
  - Shellscript
  - Python
  - target OS specific knowledge

Blueprint developer (network)

Develops virtual networks (switches, routers, Ethernet).

- Required skills:
  - libvirt XML format
  - extensive TCP/IP stack knowledge including:
  - DNS, DHCP, IPv4/v6, MAC addresses, VLAN...

Blueprint Tester

Test deploys blueprints and reports bugs to the blueprint developers. Checks the blueprints for quality, compliance, correctness and basic security before they are allowed to run in production.

- Required skills:
  - Same basic knowledge as all the Blueprint developers
  - Has 'meticulous' as a personality trait

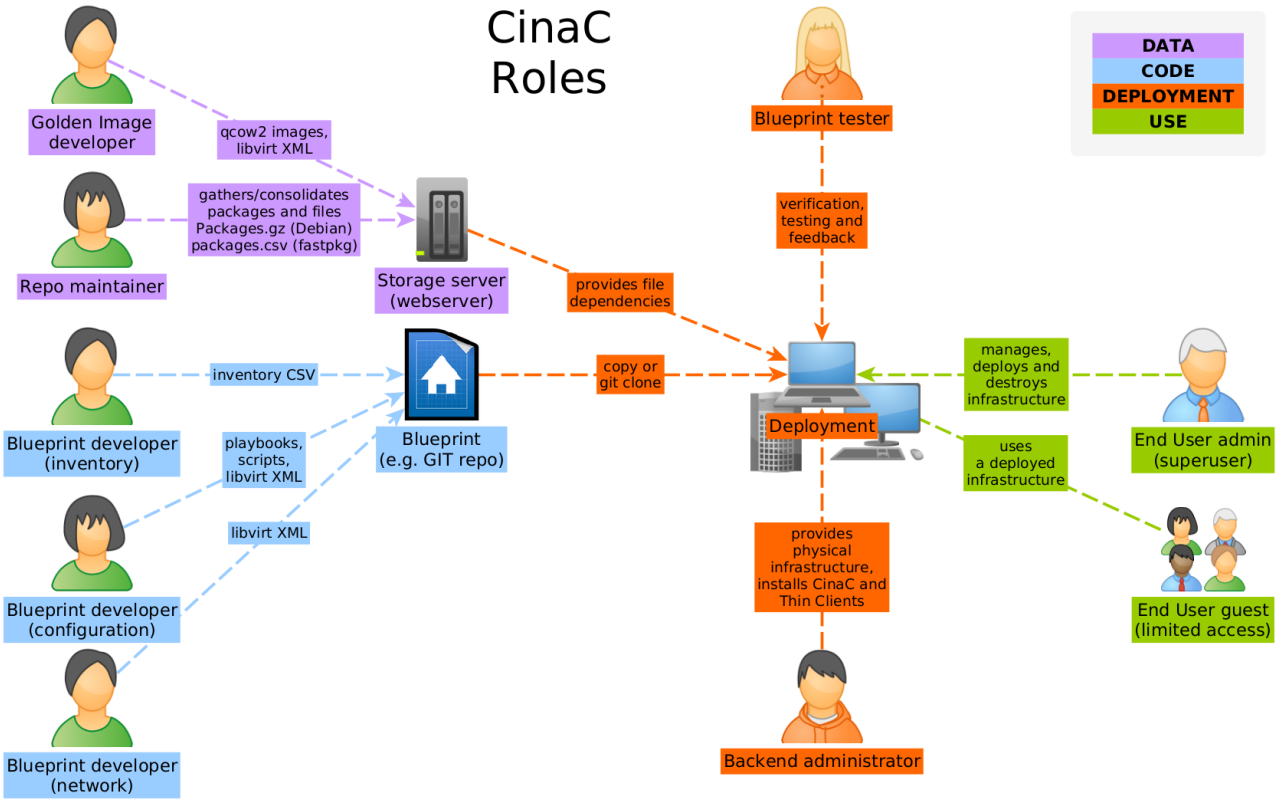

Figure 2: Roles overview:

DEVELOPING BLUEPRINTS

Workstation installation

Blueprint development is done on a CinaC workstation. The workstation can be installed on a physical machine or in a virtual machine.

First you need to follow the 'Single instance' installation guide above. After that install the development tools via:

```
$ sudo apt install \
    servermonkeys-devtools \
    servermonkeys-templates \
    ansible-role-servermonkey-cinac \
    isoremixer \
    cinac2deb
```

If you are on a machine that already has a Desktop environment, you are done. If not, you can automatically set up a complete development workstation by running the following command. Be mindful this is only recommended on a fresh Debian 11 installation.

```
$ ww -t localhost servermonkey.cinac -b devstation
```

Development flags

You can use the following flags to speed up the development process. This is helpful when you are experimenting and changing parts of a blueprint and don't want to reload the whole blueprint.

```
$ cinac-load <BLUEPRINT_NAME> <FLAG>
$ cinac load <FLAG>
$ cinac load-fast <FLAG>
```

- `v` - Print more debug information, up to four levels, like: 'vvvv'
- `s` - Skip ha**s**h verification of fastpkg downloads, assumes they are correct
- `i` - Skip VM **i**nstallation step in [auto-config]
- `c` - Skip the **c**onfiguration step in [auto-config]

cinac 'load-fast' always includes the 's' flag.

Tutorial

First you need to understand what the above tools do. Read up on them here:

servermonkeys-devtools
servermonkeys-templates
ansible-role-servermonkey-cinac
isoremixer
cinac2deb

To enable more Windows software in the fastpkg repo, please submit installation arguments here: github.com/ServerMonkey/servermonkeys-templates/packages_args.txt

SUPPORT

I develop this project in my spare time. So please respect if I do not answer immediately or ignore questions that seem stupid or unrelated to this project. That being said you can get support via Discord, mail dev(at)muspekaren.se or GitHub. I am usually available UTC+2 (Sweden) on mo-fr after 18:00 PM. Preferably via mail or GitHub-issues.

KNOWN ISSUES

- **fastpkg does not install / download a package** : Fastpkg downloads packages from all over the Internet. Sometimes servers are down or there are connection issues. Running fastpkg again or with '-f' usually fixes the issue. If not, the culprit is usually an outdated link. In that case please contact me, and I will fix it. To get around that problem yourself, you can mirror and host a local copy of any fastpkg repo yourself.
- **There is no sound from the VM in virt-viewer/autokiosk** : This seems to be a bug or a virt-viewer dependency issue. The workaround is to install the package 'pulseaudio' and reboot, on the machine you are running virt-viewer on (not the VM).

COPYRIGHT

See license file

SEE ALSO

fastpkg(1), inventorymaker(1), vmh(1), wildwest(1), autokiosk(1), spice-web-client(1), cinac2deb(1), ansible2deb(1), isoremixer(1)

github.com/ServerMonkey/cinac
github.com/ServerMonkey/cinac/docs

Servermonkeys software philosophy

This is just my personal opinion on how to develop software and build IT-infrastructure. Mostly in the context of automation. These are guidelines, not rules, so chill. It is called 'philosophy' for a reason.

Basic philosophies

Now bank these philosophies into your head, if necessary, with a hammer.

- **Follow Eric Raymond's 17 Unix Rules**
  en.wikipedia.org/wiki/Unix_philosophy#Eric_Raymond's_17_Unix_Rules
  medium.com/programming-philosophy/eric-raymond-s-17-unix-rules-399ac802807
- **Only use Phoenix configurations** (Martin Fowler / Kornelis Sietsma)
  martinfowler.com/bliki/SnowflakeServer.html
  martinfowler.com/bliki/PhoenixServer.html
  martinfowler.com/bliki/ImmutableServer.html
  This will avoid Configuration drift and specific staff dependency.
- **Write amazing documentation.** Programs without good documentation are incomplete programs and should be treated as Alpha versions. You want others to use your programs? Write a top to bottom style manual. The reason why to write 'amazing' documentation and not 'good enough' documentation is because good enough documentation is what we have now, and it is still not enough. You don't want to write a manual? Maybe you should work somewhere else. You are not allowed to write a manual? Tell your boss that documentation is PART of the code, not separated from it, documentation must be a part of every project budget. Computer programs without documentation are absolutely worthless. Nobody cares that your bicycle fell into the river if you can't tell them where.
- **Prefer stable software over shiny software.** As much as you want that new feature. Do you really need it right now? New features are usually there to increase productivity. But then, what is really hindering productivity? Is it that you don't have access to that one new shiny function, that you think you need? Or is it rather that things are not working just because they are too new? Do you have more problems because you lack functionality or do you have problems because someone wrote code that you can't run because you are missing a library that can only be installed by following some obscure guide on the internet? Are you unproductive because everyone is developing on the same old system or is it because everyone in your team uses a different version, because they happened to install that latest python pip package on a particular day?
- **Don't be clever.** Before you were even born there have been other developers and sysadmins that had the same challenges you have today. All these people could solve all those problems before you. They left behind a legacy of documentation, Open source programs, forum posts and more. If you have a problem there is a 99.9 % chance someone else already solved it long before you. - Don't be clever and think you are in a special situation. If, after month of research and learning, you still don't have a solution, then maybe (a big maybe), you can give yourself permission to develop a new software or doohickey, that solves your specific problem. And even then. Do not reinvent the whole doohickey, try to build your solution on someone elses, well tested, doohickey.
- **Code like it's the 1980s.** Modern computing is made of layers on top of layers of previous technologies. The technologies of the 70s and 80s has not disappeared, nor has it been replaced, but is mainly hidden underneath all those other layers and front-ends. Did computers work in the 80s? Yes! Did we have Docker, snapd and web applications back then? No! So do we really need them then? No. All computer systems are someone's personal garbage opinions layered on top of other garbage opinions. It is simply easier to dig trough one garbage bin than a dozen. So, develop with the pretense that the target system is from the 80s:
  - Software installation should always be possible fully offline. Do not make your software require internet access to install or to run.
  - Software should not require bizarre amounts of computing resources to function.
  - Software should be backwards compatible as much as possible. Your software should be able to run on the previous two stable releases of any operating system.
  - Software should be as portable as possible.
  - Stick to *NIX standards or at least the philosophy behind it.
  - Web applications did not exist in the 80s. So don't write any. If you have to, think long and hard about why you want to make your life more miserable.
  - Write a CLI application before you write a GUI for it.
  - Don't use weird libraries that are not part of the standard library.

Automation philosophies

- **Take your time automating.** Initial development of automation tasks takes more time than hands-on setups. But is much more worth it in the long run.
- **Abstract away, don't add complexity.** If you have two buttons and after your automation you have three buttons, you are doing it wrong. Turning a CLI interface into a graphical interface, is not automation. It is just turning the interface into another interface. It is the amount of possibilities that defines complexity. Automation means to take lots of possibilities and turning them into a minimal set of the right possibilities. Human error in computing is caused by information overload, not by the information itself.
- **Fewer choices.** In software architecture having fewer choices can be better because it simplifies the decision-making process and makes it easier to maintain and scale the software. When there are too many choices, it can be overwhelming and time-consuming to decide which options to use. It can also lead to compatibility issues and make it harder to update or change the software in the future. By limiting the number of choices, developers can focus on building a solid foundation for the software and avoid unnecessary complexity.
- **Write code that is easily comprehensible** by other human beings. Do not try to be smart and write complicated code that looks cool or crunch all code into one line, just because you can. Impress your colleges by making is easy for them to understand your program.
- **Avoid multidimensional data**. Humans have a very hard time understanding more than three spacial dimensions. Imagine a JSON file with four or more dimensions in comparison to a spreadsheet with only two dimensions. And compare that to a simple text file with a list, that is only one dimension. Now it is pretty clear that working with the one-dimensional list is easiest. So when you develop, try to avoid multidimensional data and instead split your data into multiple files. Sometimes multidimensional data is unavoidable, but think long and hard before you make others and your own life miserable. Do you have terra-bytes of data? Does your program really need a database?
- **Write Ansible playbooks** in favor of shell/python-scripts where you can. Playbooks are more readable and reliable. The same goes for any other automation tools of your choice.
- **Write none-interactive programs.** All automation programs/scripts should be strictly none-interactive. Meaning don't make your program suddenly stop and wait for user input. Use variables and 'export' functions. It is called automation for a reason.
- **Avoid parametrization** where you can. Parameters (arguments) gives room for human error. It is almost impossible not to use parameters in programs and scripts. But when writing code, think about if your program really needs parameters.
- **Avoid Feature creep**. Do not add useless things to your program because someone 'might' need them in the future. Only add features that are really used.
- **Adhere to the default behavior of standard streams.** Meaning don't write scripts that spit out errors where there are none. And the other way around. Adhere to return-codes and print errors to std-err.
- **Avoid unnecessary output.** Only use silent confirmation. Meaning if the requests action/task worked, do not print out unnecessary confirmations, exceptions can be made for tasks that take a long time. This will keep log-files readable. Especially if you run lots of tasks. Only print information if something went wrong in your script/program. Remember that Ansible can properly parse standard streams and return-codes.
- **Use Static Code Analysis.** Also known as linting. Use it for every programming language you use. For example when writing Shell-Script in PyCharm add #!/bin/sh to the first line and install the Shell Script coding assistance plugin. Now fix all issues the coding assistance points out. If you use another IDE look at ShellCheck, Shfmt and Explainshell. The same goes For Ansible playbooks (ansible-lint) and any other programming language.
- **Humans are lazy and stupid.** Write software in a way that a five-year-old can use and understand it. Nobody is interested in how cool or complex your software is. They just want to achieve a specific goal and then go on with theyr lives. Make others live easier, not more complex.
- **Be meticulous.** Because everything that is mediocre does not last. If you have even minor errors in your software, people will very fast loose interest.

servermonkeys Ventoy templates

DESCRIPTION

A collection of my personal Ventoy templates for fully automated installation of Debian and Windows. Includes Debian Preseed files and Windows Answer Files.

This guide will also show you how to create a USB-installation stick, for fully automated/unattended installation. It will also show you how to use fastpkg to download the necessary files.

TEMPLATES

Debian 11/12

- `generic.cfg` : Default Debian installation with SSHD enabled USER/PASS: *ansible* . Automatic DHCP discovery on the first Ethernet port. Swedish timezone/keyboard/locales. Includes proprietary firmware. Lets the user choose disk layout (semi automatic).
- `generic_laptop.cfg` : Same as generic, also installs 'laptop' package.
- `generic_nvme.cfg` : Same as generic, but automatically installs to `/dev/nvme*`
- `generic_nvme_laptop.cfg` : Same as `generic_nvme.cfg` plus `generic_laptop.cfg`
- `generic_sda.cfg` : Same as generic, but automatically installs to `/dev/sda`. Be careful this can target and erase the Ventoy USB-stick itself. If you are unsure choose `generic.cfg` instead.
- `generic_sda_laptop.cfg` : Same as `generic_sda.cfg` plus `generic_laptop.cfg`

Windows

Incomplete/untested

Prepare USB stick

Get a USB-flash-drive with at least 8 GB of disk space.

Format the USB-stick with Ventoy:

```
bash /opt/Ventoy_<VERSION>/Ventoy2Disk.sh /dev/<DISK>
```

To understand how to include the Pressed file in Ventoy, read this manual:
ventoy.net/en/plugin_autoinstall.html

Copy this entire folder to your USB sticks root partition and give it the name 'ventoy'.

Copy all ISO files required by the ventoy.json file to the flash drives root partition.

You can use the tool fastpkg to download the correct disk iso's github.com/ServerMonkey/fastpkg

Open the file ventoy.json file and lok at the lines that contain the key ' image'. The run for example:

```
sudo fastpkg -p debian-11-firmware-dvd1_11.1.0.x64 download
```

Then copy onto the mounted USB-stick.
On some Linux distributions it is a good idea to run the 'sync' command to make sure all files are fully written to the USB-stick.

```
rsync -aP /var/lib/fastpkg/downloads/debian-11-firmware-dvd1_11.1.0.x64.iso /mnt/usbstick
sync
```

If you have problems downloading Windows ISO's read here: github.com/ServerMonkey/servermonkeys-devtools/windows-isos.md

Now you should have a disk layout similar to this:

```
/
- ventoy
   - debian/
   - windows/
   - README.md
   - ventoy.json
- debian-11-firmware-dvd1_11.1.0.x64.iso
- ...
```

Unmount your flash-drive and you are done.

BIOS/UEFI settings

The templates are configured to automatically select UEFI over BIOS but should work with both depending on how you boot into the Ventoy USB-stick.

USAGE

Just put the flash-drive into your PC or server and select the menu option you want.

YOUR OWN DEBIAN PRESEEDS

To create your own Debain Pressed file you can download a template with

```
sudo fastpkg -p debian-11-preseed-example install
```

Or get the latest version here:
www.debian.org/releases/stable/example-preseed.txt

For older Debian releases use Archive.org (check your Debian release date):
web.archive.org/web/*/https://www.debian.org/releases/stable/example-preseed.txt

Modify the Pressed file to your liking and include it in Ventoy.
To understand how to modify the Pressed file, read this manual:
wiki.debian.org/DebianInstaller/Preseed

fastpkg(1) – A packageless package manager

SYNOPSIS

`fastpkg [<OPTIONS>] <ACTION> [<SUB_OPTIONS>]`

`fastpkg-template [-h, --help] <FILE_NAME>`

DESCRIPTION

A complementary packageless package manager, turn any URL-file into a Linux like package.

Managing software and files from third party sources can be a pain. This is especially true for Windows users. Fastpkg is a complementary packageless package manager that can be used to manage software and files from third party sources.

It differs from a traditional package managers in that packages don't come in a specific packaged format like .deb, .nupkg or .exe, that are specific to the package manager. Instead, packages are downloaded directly from a network and then installed. Unlike traditional package manager, in fastpkg - how packages are installed is defined in the repos packages.csv file, not in the package itself. This means that any package can be used on any system that has fastpkg installed. As long as the target system support that file type. This means you can host any kind of file, not just software. Like disk ISOs, website templates, Windows exes, Windows exes that come as zip files or .deb files that are hosted on GitHub. It can even be used as a wrapper for other package managers like APT.

This gives the additional bonus of not having to repackage software and host large amounts of packages on your own. This also enables distribution of software, that is not open source or that is otherwise not allowed to be repackaged. Fastpkg basically enables you to distribute any URL-file as a package without permission and without infringing on trademarks.

Target requirements:

- POSIX compatible OS (e.g. Linux or Cygwin)
- Python 2.7 or 3

Optional requirements:

- axel (available via shell)
- dtrx (available via shell)
- jigdo-lite (part of jigdo-file) (available via shell)

dtrx can also be installed via fastpkg itself. In the official repo, it is available as 'fastpkg-dtrx'.

Tested on: Debian 10, 11, 12, Windows XP with Cygwin

OPTIONS

- `-h`, `--help` : show this help message and exit
- `-a`, `--all` : Use instead of '-p'. Selects all packages in the repo(s).
- `-b`, `--blackwhite` : Don't colorize output
- `-c`, `--continuedl` : Continue an aborted jigdo download
- `-f`, `--force` : Re-download package. This can fix broken downloads.
- `-p`, `--package <PACKAGE_NAME>[_<VERSION>]` : Name of a package or packages. Separate multiple packages by a space and in quotation marks. Specify a version with underscore: `<PACKAGE>_<VERSION>`
- `-q`, `--quiet` : Suppress all confirming output, show only errors and warnings.
- `-s`, `--skip` : Skip hash verification, only use if you already know that all files are correctly downloaded.
- `-r`, `--repo <REPO_URL>` : Specify a single repo to target. Only for use with 'enlist'
- `-v`, `--verbose` : Get extra information. -v is recommended for users, -vv is basic debugging, -vvv is for full debugging

ACTIONS

- `download` : Download a package or packages, without installing. Uses parallel download.
- `enlist` : Append a repo URL to the repo list file /etc/fastpkg/fastpkg.list .
- `exportcache` : Export a custom package-cache to be used on an offline host. Define packages with -p or -a to narrow down the package-cache.
- `init` : Create basic folder structure.

- `install` : Download and install a package. Expects that you have a package-cache. E.g. ran 'update' beforehand or have copied download-cache and package-cache from another host.
- `list` : List all packages. Use -v to see if the packages are downloaded and/ore installed.
- `show` : Show package details
- `update` : Update cache for every repo found in /etc/fastpkg/fastpkg.list .

SUB OPTIONS

For action `show`

- `-d`, `--downloadpath` : Only show the Download-Path variable.

For action `install`

- `-i`, `--installpath <PATH>` : Override where to install packages with a custom installation path.
- `-o`, `--overwrite` : Install files, even if the target folder already exists.
- `-r`, `--rename` : Override the automatic naming of files and paths. Choose your own name. Only works in conjunction with the –installpath flag.
- `-n`, `--noversionrename` : Override the automatic naming of files and paths. Will use the automatic name but without the '_VERSION'

COMMON USAGE EXAMPLES

Add a repository (this example adds my personal repo)

```
$ fastpkg -r https://muspekaren.se/fpkg enlist
```

Add a repository from a local file

```
$ fastpkg -r file:///opt/my-local-repo/packages.csv enlist
```

Update all repositories

```
$ fastpkg update
```

List all packages

```
$ fastpkg -a list
```

List all packages, more detailed

```
$ fastpkg -va list
```

Show package details

```
$ fastpkg -p wordpress show
```

Only download a package

```
$ fastpkg -p wordpress download
```

Download and install a package

```
$ fastpkg -p wordpress install
```

REPO MAINTAINANCE

To create your own repo, all you need is an HTTPS web-server and a packages.csv file. The packages.csv file is a simple text file that contains information about the software, such as name, version, download URL, hashes and more. To create a template for the packages.csv file run 'fastpkg-template '. You can open the CSV file with LibreOffice or similar. The packages.csv file can be hosted locally or on any HTTPS web-server. HTTPS is mandatory for the repo but not the packages themselves.

To see what a packages.csv file looks like, just open the one used by the official repo:
muspekaren.se/fpkg/packages.csv

Because the package repo is a simple CSV file you are not allowed to use commas in variables. This is a limitation but this also makes management very simple.

Supported archive formats: bz2, gz, lzma, tar, tbz, tgz, tlz, xz, zip

The packages.csv file is made of a table with 8 columns and an unlimited number of rows. Each row represents a package. Each column represents a variable. The first line is the header and the following lines are the values. There are 8 variables:

- NAME : name of the application
- VERSION : version number
- EXTENSION : file extension
- INSTALLER : how to install the package
- SCRIPT : customize the installation process
- DESCRIPTION : short description of the package
- SHA256 : hash of the package
- SOURCE : URL to the package

Here is a more in depth explanation of the variables:

NAME Globally unique name of the package, you can use capital letter to make it more readable. When choosing names think long and hard about the name. Try to use Linux package compatible names. For example the program QTodoTxt is not available in APT and has a unique name all over the Internet. So it is safe to use the name 'QTodoTxt' as is. If you use fastpkg as a wrapper for other package managers like APT, use the naming scheme of the target package manager. For example for the program 'debcvescan' use the name 'debcvescan'. Do not use name like 'debian-debcvescan' because it will be clear from the extension .deb, that this is an APT package. If you need different OS versions of the same package, it is better to create a separate fastpkg repo for your packages instead of using a name like 'ZDoom-win' or 'ZDoom-Ubuntu'. When installing none-executable files, that don't have a native fastpkg installer, it is recommended to use prefixes like 'wallpaper-' for .jpg wallpapers or 'wadfile-' for DOOM .wad files. It is ultimately up to the package maintainer, to choose a meaningful naming scheme that works for the users. If a package requires multiple files, all you have to do is to give it the same NAME and the same VERSION. Fastpkg will automatically install all files with the same NAME and VERSION.

VERSION Version of the package, use dot separated strings. Fastpkg tries to handle all available versioning schemes to figure out the latest version. See https://en.wikipedia.org/wiki/Software_versioning . Versions like 2.03 and 2.04 will work just fine. Prioritization is based on separation by dot and the order from left to right. Using a date is a good method to version packages that don't have a release. For example, use the file creation date YYMMDD. If a files does not have a version, like a wallpaper or a file that never changes, just use the number '1' as version. Other examples are 1.2.3.beta, 1.2.3.alpha, repack-1, repack-2.1, 3.5.x32, 3.5.x64 . Even only letters can be used as a version number. To debug if the latest version is detected correctly run:
`fastpkg -vvvp myapplication show | grep 'latest version'`
It is ultimately up to the package maintainer, to choose a meaningful versioning scheme that works for the users.

EXTENSION File-extension of the file to download. E.g. 'zip', 'exe', 'cfg', 'jpg' and so on. It is not allowed to use an empty string. If you have a package that does not have an extension, use for example 'data' or 'run' as extension. This will rename the downloaded file to this extension. This string is also uses to tell the installer how to behave and detect archive or jigdo files.

If the extension is '.jigdo' fastpkg will use jigdo-lite to download the image. Usually you want to use the extension '.iso.jigdo' in your repo. Make sure your URL points to the .jigdo file and not the .iso or .template file.

INSTALLER A string that defines what installer to use. For example 'winapp' will install the program on all systems as a Windows application. Even on Linux. In that way you can install Windows applications on Linux for use with, for example, WINE. What usually happens is that the package will only be extracted/copied to a specific location. It is then up to the user how to execute that program. See in the list below 'Supported installer types'. Some more examples: 'ansible-role', this will fetch a ZIP file from the SOURCE URL and extract it to /etc/ansible/roles. Nothing more.

SCRIPT specify custom installer arguments to modify the installation. The following arguments are supported:

- `ARGS: <SILENT INSTALLER ARGS>` : When installing, will execute the SOURCE binary with this specified arguments. For example, if you have an .exe file that supports silent install, you can specify the arguments here. For example, to install Firefox silently on Windows you can use `ARGS: -ms -ma`
- `extract` : This will pre-extract archive files. For example, you have an .iso file that is packaged as a .zip file. You can specify 'extract' to extract the .zip file before the .iso file is installed. This is also useful to pre-extract nested archives.

- `background` : Will be installed as a wallpaper. In Posix this will be /usr/share/backgrounds and in Windows this will be WIN_ROOT/WINDOWS/Web/Wallpaper

DESCRIPTION A short, one sentence, description of the package. Like for the package ' WordPress' this could be 'Content management system written in PHP'.

SHA256 The complete 64 characters long SHA 256 hash string of the source file.

SOURCE URL pointing to the file to download. Make sure to avoid redirects and use persistent direct download URLs. If you have a file that is hosted on for example SourceForge, use the direct download URL. Use links that point to the exact version of a package like https://github.com/MyUser/MyApp/archive/refs/tags/20180505.zip and not https://github.com/MyUser/MyApp/archive/refs/heads/main.zip . If you have a file that is constantly changing but has the same URL, the best way is to use archive.org to create a snapshot of that file. Or host that file on your own server. The same goes for URLs that are only available for a limited time or are cookie/token based. If you need proprietary software, that is only available for a limited time, you have no choice but host them internally, of course.

Supported installer types:

Installer tag	Install directory	Description
ansible-coll	/etc/ansible/collections/ansible_collections	Ansible Galaxy collection
ansible-role	/etc/ansible/roles	Ansible role
app	/opt	Generic Posix compatible application
data	/var/opt/data	Generic none executable files
disk	/var/opt/disk_images	*Disk images, like .iso
firefox-ext	/var/opt/firefox_addons	*Firefox addon installer
java	/opt	Java applications, like .jar
libvirt	/var/lib/libvirt/templates	*Libvirt templates, like .xml and qcow2
vm	/var/opt/vm_templates	*Other virtual machine templates
web	/var/opt/www_templates	Website templates, like .html and so on
winapp	**WIN_ROOT/opt	Windows executables, like .exe or .msi
windata	**WIN_ROOT/data	Generic none executable files

*These, usually large, static files will be linked from the downloads directory to the installation directory.
**WIN_ROOT is the root of the Windows installation, e.g. C:

OUTDATED LINKS

By providing a collection of links from the internet, the chance that one of those links will be outdated very quickly is high. You are welcome to use my repo https://muspekaren.se/fpkg in your projects, but There is a high change that links in my repo will stop working. Please contact me if this happens and I do my best to update them. A simple solution is to just copy my repo and replace the broken links with the URL of your own internal file-host. In that way you don't need to rewrite scrips that use the same package name from my repo on muspekaren.se/fpkg.

Also see my notes on Miscrosoft ISO files: Windows ISOs

Another good reason to host your own repo is speed. Several packages in my repo are provided by archive.org which have very slow download speeds.

COPYRIGHT

See license file

SEE ALSO

axel(1), dtrx(1), jigdo-file(1), jigdo-lite(1)

fastpkg example packages.csv

NAME	VERSION	EXTENSION	INSTALLER	SCRIPT	DESCRIPTION	SHA256	SOURCE
# Demo repo							
ansible-role-geerlingguy-ntp	2.3.1	zip	ansible-role		Ansible role: Installs NTP on Linux	993388799f	https://github.com/geerlingguy/ansible-role
Burpsuite-community	2022.3.7	jar	java		Platform for performing security testing of web applications	bee9e17777	https://portswigger-cdn.net/burp/releases/c
CQTools	20190510	zip	data		CQURE Team penetration testing toolkit from BlackHat Asia 2019	8413033aa7	https://github.com/ServerMonkey/cqtools/a
QTodoTxt	1.7.0	tgz	app		UI client for todo.txt files from http://todotxt.org/	eb7a9b7ce8	https://github.com/QTodoTxt/QTodoTxt/arc
Virtio-drivers-Windows	0.1.190	iso	disk		KVM drivers for Windows	dc6044e02f	https://fedorapeople.org/groups/virt/virtio-w
wallpaper-LoopingViolet	1	jpg	data	background	Desktop Wallpaper	192b454ad9	https://wallpapers.com/images/hd/simple-p
wkhtmltopdf	0.12.6.1.r2	deb	dpkg		Replace the default Debian 11 version of wkhtmltopdf	50a3c5334d	https://github.com/wkhtmltopdf/packaging/
Debian-11-Preseed-Example	20220526	cfg	data		DebianInstaller Preseed	4d5ea8f6b0	https://web.archive.org/web/20220526090
Debian-11-nocloud	20220121.894.x64	qcow2	vm		libvirt VM: No cloud-init but allows root login without a password	1702518df1	https://cloud.debian.org/images/cloud/bulls
Wordpress	5.8	zip	web		Content management system written in PHP	e4d78cc309	https://wordpress.org/wordpress-5.8.zip
Wordpress-Theme-2021	1.4	zip	web		Wordpress Theme	6906b1cf88	https://downloads.wordpress.org/theme/tw
clamwin	0.103.2.1	exe	winapp	ARGS: /sp- /silent /norestart	OpenSource Antivirus software with virus scanning and virus defi	ac9d76d879	https://netix.dl.sourceforge.net/project/clan

inventorymaker(1) – CSV to Ansible inventory file converter

Handle your complete Single Source of Truth (SSOT) using CSV files that are both readable and writable by humans.

Tested on Debian 11

FEATURES

- Generates Ansible inventory file
- Generates vmh environment file
- Uses 'pass' to manage passwords: www.passwordstore.org

SETUP

Inventorymaker uses the tool 'pass' to get passwords for the Ansible inventory file. The 'pass' program requires a working GPG setup.

Typical setup:

Create a GPG password and key for the user:
`gpg --full-gen-key`

Then init pass tool with that email:
`pass init your.name@example.org`

Correctly configure gpg-agent with:
`inventorymaker-init`

If you have issues with GPG permissions, this might help:
`find ~/.gnupg -type d -exec chmod 700 {} ; find ~/.gnupg -type f -exec chmod 600 {}`

GPG Manual: wiki.archlinux.org/title/GnuPG

BASIC USAGE

Syntax:

`inventorymaker -h, --help`

`inventorymaker <FOLDER_WITH_CSVS | template> <OUTPUT_FILE>`

Inventorymaker will parse the CSVs and apply custom parsing rules that will result in a directly usable Ansible inventory file. Do not edit the final Ansible inventory file manually. During conversion the following things happen:

- Inventorymaker will import passwords from the 'pass' tool. This is a password manager that uses GPG to encrypt passwords. inventorymaker will look for passwords in: `~/.password-store/ansible/<PASS_PATH>.gpg` If the GPG password is set to a none-empty string, inventorymaker will ask for your GPG password. For automated servers it is recommended to set the GPG password to an empty string.
- Due to speed considerations, passwords will be stored in plain text in the output Ansible inventory file (attributes are set to 600).
- Will automatically add 'local' and 'localhost' to the inventory file.

CSV FORMAT

- Each folder represents a confined SSOT source.
- Each CSV file in that folder represents a group of hosts.
- The CSV file must use commas as separators.
- Filenames must end in .csv
- The first line always contains the Ansible variable names.
- The second line always contains group values for all hosts below.
- All following lines contain individual host values.
- Individual host values will override group values but try to avoid using group values and individual host values together. Choose one or the other.
- Inline comments can be added by starting a cell with '#'.

VARIABLES

Variables in the CSV file are not exactly the same as Ansible variables. They are simplified by removing the prefix 'ansible_'. Meaning the variable 'HOSTNAME' will result in the output variable 'ansible_hostname'.

- The minimum required variables are: HOSTNAME and USER or CHILD
- The CHILD variable can be used instead of HOSTNAME, to group host together. This can simply point to another CSV file in the same directory. Just use the name of the file without the .csv extension. For example node1.csv node2.csv can be grouped together in the file rack.csv. Ansible can now target the name 'racks', see the example below. This can be very useful when defining a large number of hosts that are similar.

CHILD
node1
node2

- There are new variables that extend Ansible's functionality. They start with 'im_' or 'tag_', see list below.
- Ansible variables that are not included in the list below, can be utilized by adding the complete variable with the 'ansible_' prefix.
- You can use your own custom variables, they will automatically get the prefix 'im_' in the final Ansible inventory .ini file.
- Recommended to only use static variables that don't change during the lifetime of a host. Otherwise, it is better to define variables in a playbook.

IM VARIABLE	ANSIBLE VARIABLE
BECOME	ansible_become
BECOME_METHOD	ansible_become_method
BECOME_PASSWORD	ansible_become_password
CHILD	tag_child
DESCRIPTION	im_description
DNS	im_dns
FACTORY_HOST	im_factory_host
FACTORY_PASSWORD	im_factory_password
FACTORY_USER	im_factory_user
HARDWARE	im_hardware
HOST	ansible_host
HOSTNAME	ansible_hostname
HPILO_HOST	im_hpilo_host
HPILO_PASSWORD	im_hpilo_password
HPILO_USER	im_hpilo_user
IPMI_HOST	im_ipmi_host
IPMI_PASSWORD	im_ipmi_password
IPMI_USER	im_ipmi_user
LOCATION	im_location
MAC	im_mac
ORDER	im_order
OS	im_os
PASSWORD	ansible_password
PORT	ansible_port
PYTHON_INTERPRETER	ansible_python_interpreter
STATE	im_state
USER	ansible_user

VALUES

To see what each variable's value does it is recommended to generate a template and read the comments in that file. See further down under 'EXAMPLE USAGE'.

Because some values are more complicated, they are explained here:

For the ORDER variable : This integer value sets the order in which the program vmh creates virtual

machines.
vmh can be found here: github.com/ServerMonkey/vmh
Settings a value here ultimately marks the host as a VM. Inventorymaker will automatically generate a custom vmh environment file. Be mindful that the value for OS and HOSTNAME also must be specified. In this case the value of the OS variable represents the VM image.

It will translate like this:

IM VARIABLE	VMH VARIABLE
ORDER	ORDER
HOSTNAME	DOMAIN_NAME
OS	DOMAIN_SOURCE

For the PASSWORD and USER variable : If this value is set to 'LOCAL' inventorymaker will assume that this is a physical-access-only system, that can be reached only via a physical terminal. This host will be excluded from the exported Ansible inventory file.

For the STATE variable : The value for the STATE variable is used to determine the systems state of a host. It is up to your playbooks to handle the different states. Recommend using the purposes from the list below. When using DOWN and EXC, the host will automatically be excluded from the exported Ansible inventory file. This is nice if you want to manage hosts in your source inventory but not in Ansible.

VALUE	PURPOSE
UP	host is supposed to always be online / booted
DOWN	permanently offline, will be excluded from inventory
MAN	manual, either booted or offline
MAINT	maintenance, under repair or broken
EXC	exclude, will be excluded from inventory
LIQ	liquidate, is supposed to be removed soon
UNKNOWN	unknown state, existing but unknown host

`$USER` value : This value will be replaced with the current users Posix username. This is not a shell-script variable. Example:

HOSTNAME	OS	USER	PASSWORD
winbox	Windows 11	peter	PASS
ubuntubox	Ubuntu	$USER	PASS

`PASS` value : This value will be replaced with the password from the password manager 'pass'. It will automatically retrieve the password from the password storage by searching for the matching group and username. Works only for the variables in the list below.

VARIABLE	MATCHING USER
PASSWORD	USER
BECOME_PASSWORD	USER
FACTORY_PASSWORD	FACTORY_USER
IPMI_PASSWORD	IMPI_USER
HPILO_PASSWORD	HPILO_USER

Let's demonstrate with this example file named 'myhosts.csv':

HOSTNAME	USER	PASSWORD
centosbox	jack	PASS

Inventorymaker will look for the password in `~/.password-store/ansible/myhosts/centosbox/jack.gpg`.

When using value for the variable CHILD in the same row, the password path will be the name of the child group, not the hostname.

`PASS:<PATH>` value : Custom path, instead of matching the username automatically. You can specify the relative search path. For example: `PASS: pcs/jacks-pc`, will retrieve the password from `~/.password-store/ansible/pcs/jacks-pc.gpg`.

If you don't want to use plain-text passwords in your Ansible inventory you can ommit pass and let Ansible handle passwords like this instead: `"{{ lookup('passwordstore', 'ansible/myusername', errors='strict')}}"` Be aware that when you have hundrets of passwords this will significantly slow down your plays. Or use vaults.

EXAMPLE USAGE

Create a folder that contains all of your CSV source files. In this example we use the folder '~/lab/inventory_src'

Create a CSV file from the template in that folder:

```
$ inventorymaker template homelab.csv
```

Now open the CSV file with libreoffice or visidata. Or any other CSV editor of you choice.

- Add your hosts to the CSV file under 'HOSTNAME', remember to skip the second row because it is used for group values.
- Add passwords and other variables

It should look something like this:

HOSTNAME	HOST	USER	PASSWORD	DESCRIPTION
# group variables				Test machine
my-workstation	192.168.40.2	peter	aplaintextpass	
ftp.example.com		user34	PASS	

To convert the source inventory to an Ansible inventory file, run:

```
$ inventorymaker "$HOME/lab/inventory_src" "$HOME/lab/homelab.ini"
```

Now inventorymaker will abort and complain that the password for 'user34' is missing from the pass store. Add it by running:

```
$ pass insert ansible/homelab/ftp.example.com
```

Then run inventorymaker again. You should now have a working Ansible inventory file. It is a good idea to open and verify it.

KNOWN BUGS

Avoid using group passwords and individual host passwords together. In some cases sudo authentication will not work.

SEE ALSO

ansible(1), wildwest(1), vmh(1), pass(1)

wildwest(1) – Turn Ansible into a better automation management tool

Wildwest is a wrapper for Ansible, the goal is to make Ansible easier to use. It also adds several new features to Ansible which makes it easier to use Ansible as a general purpose automation tool.

- Turn Ansible into a parallel-ssh like tool
- Less verbose output via custom Anstomlog
- Colorized output / logs and save to an HTML file
- TAB Autocomplete playbooks, tasks and scripts
- Extensive help system as part of the CLI
- Speed optimized configuration with Mitogen
- Run Ansible tasks and roles directly without Playbook
- Run shell scripts directly without Playbook, no need to know Ansible

At runtime parameters, do not exist in wildwest. This is a design choice because automation should be predictable and only be defined by code. Without runtime parameters there is less room for input errors. It forces the user to write proper playbooks or scripts instead.

INSTALLATION

Tested on Debian 11

To enable TAB complete for ww, install python3-argcomplete and then enable it:

```
$ sudo activate-global-python-argcomplete || sudo activate-global-python-argcomplete3
```

CONFIGURATION

Default configuration can be found in:
/usr/local/share/wildwest/ww_default.cfg
The first time ww is run it will copy this file to ~/.ww/ww.cfg Open this file to get a better understanding of how wildwest facilitates namespace and inventory management.

The current user configuration is stored in ~/.ww/ww.cfg

The default Ansible configuration can be found in:
/usr/local/share/wildwest/ansible_default.cfg
The first time ww is run it will copy this file to ~/.ww/ansible.cfg Open this file for more information and to add your custom settings.

The default settings are optimized for any generic workstation, and connecting up to around 1000 hosts simultaniously.

If there is not ~/.ansible.cfg file, wildwest will link to ~/.ww/ansible.cfg instead.

Put your global Ansible roles and collections in:
/usr/share/ansible/roles or **/etc/ansible/roles**
If you want to override above roles/collections or develop your own, put them in: **~/.ansible/roles** or **~/.ansible/collections**

USAGE

HELP

`ww -h` : This will list the global help for wildwest and also list all available namespaces and what they do.

`ww <WILDWEST_NAMESPACE> -h`, `ww <WILDWEST_NAMESPACE>` : Help for a specific namespace. This will list what each playbook, task or script does inside a specific namespace. Running `ww -t <HOST_OR_GROUP> <WILDWEST_NAMESPACE> -h` Works as well.

Wildwest will look for a file named 'ww.txt' in the main directory of your role, collection or folder. Add a very short description of your play to this file. This will then be displayed in this help interface.

To enable information about each script or task to the help interface, please add the tag '#info: <TEXT>' to your files. All playbooks and shell-scripts should always have this basic code in the header:

Playbook, task or role:

```
---
#info: Short description of your script here
rest of the code here...
```

Shell script:

```
#!/bin/sh
#info: Short description of your script here
rest of the code here...
```

SYNTAX

`ww [<OPTION>] [<COMMAND>] [-t <WILDWEST_NAMESPACE> [-b, --become] <ACTION>]`

- `WILDWEST_NAMESPACE` : Extended version of the Ansible namespace system. To see how the extended namespaces works, open the file:
  /usr/local/share/wildwest/ww_default.cfg
- `-b`, `--become` : Force to run the play as superuser. This is useful because the BECOME directive is usually not defined in tasks or scripts. Useful when developing tasks or scripts.

Commands:

If no command is given, wildwest assumes you want to use -h or -t.

- `edin` : Edit the inventory file of Inventorymaker with Visidata.
- `gen` : Force to regenerate the inventory file.
- `exe` : Directly execute a shell command. Will execute with the default sh shell on the target system. Only use this command for testing and development. It is not a good idea to use ad-hock commands in production. When running 'exe' with the '–become' flag on 'localhost', wildwest will always ask for password.

Options:

- `-t`, `--target <HOST>` : Is the name of the host or host group in the Ansible inventory. If you use inventory-maker, the group name can also be the same as the inventory CSV file name. You can also use Ansible patterns like ' host3,host10' or 'host*', see: https://docs.ansible.com/ansible/latest/inventory_guide/intro_patterns.html
- `-d`, `--dryrun` : Simulate the Ansible run.
- `-f`, `--format` : Pipe and redirect friendly output. No hidden special characters or colors in output.
- `-n`, `--notify` : Play a notification sound when done. Uses aplay.
- `-v`, `--verbose` : Get extra information. Useful if you have to iterate over lots of targets. -v is recommended for users, -vv is basic debugging, -vvv and -vvvv is for full debugging. This will also show the duration of a play.
- `-w`, `--watch <SECONDS>` : Repeat the same play and watch the output. In seconds.
- `-x`, `--export` : Save the script output to a colorful HTML file. Will be saved to ~/.ww/reports. Only works with roles 'servermonkey.sh' and ' servermonkey.ww_logger'.
  See: github.com/ServerMonkey/ansible-role-servermonkey-sh
  and github.com/ServerMonkey/ansible-role-servermonkey-ww-logger

CRONTAB

`ww-task <TARGET> <NAMESPACE> <ACTION>` : Wrapper for ww. Starts wildwest actions in background, use this in crontab. Always Uses become. Will write a logfile to "~/.ww/cache/logs_task"

ADDITIONAL FEATURES

wildwest comes with an extra logging system that can be used via this role:
github.com/ServerMonkey/ansible-role-servermonkey-ww-logger

When you run a play with servermonkey.ww_logger you will get additional output to your shell after Ansible has been run. This is useful when you use wildwest as an administration tool on systems that already have been deployed/installed.

servermonkey.ww_logger is automatically enabled when you run shell scripts directly. This is also true when running command via the 'exe' command. See:
github.com/ServerMonkey/ansible-role-servermonkey-sh

Wildwest uses ccze internally to colorize these logs.
Read the code here to see what words will be highlighted in different colors:

github.com/cornet/ccze/blob/master/src/ccze-wordcolor.c
or use my CLI tool ccze-test.

When using the extra logging system. Wildwest will automatically try to make the output readable by adding extra line-breaks when each host gives more than one line of output. Practically this means when you run “echo hello” on 10 hosts, you will get 10 lines of output neatly stacked under each other. When you run a command that gives more than one line of output on each host, wildwest will automatically add extra line-breaks.

EXAMPLES

First edit the file ~/.ww/ansible.cfg and add the namespace to enable. As an example we use my Ansible role from here:
github.com/ServerMonkey/ansible-role-servermonkey-ww

Run a playbook on a single hosts in the inventory:

```
$ ww -t debianbox servermonkey.ww info_test
```

Run a playbook on multible hosts and show the duration of the play:

```
$ ww -vt debianbox,windowsbox servermonkey.ww info_os
```

Run a shell command and export the result to an HTML file:

```
$ ww -xt debianbox exe "echo Hello World"
```

Run a shell command as superuser and play a sound when done:

```
$ ww -nt debianbox exe -b "sleep 2 ; echo Hello World"
```

Regernate the inventory because you changed a password:

```
$ ww gen
```

SEE ALSO

ansible(1), inventorymaker(1), pass(1)

```
# This file defines what Ansible roles, collections and none-ansible scripts
# will be usable by wildwest. To understand this file you first need to
# understand how the Ansible namespace structure works:
# https://galaxy.ansible.com/docs/contributing/namespaces.html
#
# Ansible uses Galaxy-collections or roles as part of a namespace.
# Wildwest builds upon Ansible namespaces and adds a new functionality,
# which enables the use of ansible-tasks, scripts and individual playbooks
# together with namespaces. These are now usable as a namespace structure.
# This means that tasks, scripts and playbooks, that are not part of a
# role or Galaxy-collection, can now be used via the namespace structure.
#
# Default Ansible namespace structure is:
#  namespace.[ role | collection ]
# wildwest namespace structure is:
#  namespace.[ role | collection | task | script | playbook ]
#
# When talking about a namespace in wildwest, it's important to use the term
# 'wildwest namespace' because wildwest uses a different namespace.
#
# A role, task, script or playbook of a wildwest namespace is called 'action'.
# With other words, Ansible's <NAMESPACE>.<ROLE OR COLLECTION> is turned into
# <WILDWEST_NAMESPACE>.<ACTION>

[settings]
inventorymaker_src = default
# inventorymaker_src points to a custom directory where inventorymaker's
# SSOT CSV source files are located. The default is ~/.ww/inventory_src/
# If this line is removed, inventorymaker will not be used at all.

# The next 5 sections are used to enable ansible roles, collections, tasks,
# scripts and playbooks, for the wildwest namespace structure.
# In these sections you need to point wildwest to these existing folders.
# You only need to enter the name of the respective namespace, not the full
# path (with the exception of the last section). Wildwest will automatically
# search the following folders:
# "~/.ansible/collections/ansible_collections/" and "~/.ansible/roles/"

[galaxy-playbooks]
# namespace of a galaxy collection
# placing the name of a galaxy collection here will import all playbooks from
# that collection.
# Will then be available as: <GALAXY_NAMESPACE>.<COLLECTION_NAME>

[galaxy-roles]
# namespace of all roles inside a galaxy collection
# placing the name of a galaxy collection here will import all roles from
# that collection.
# Will then be available as: <GALAXY_NAMESPACE>.<ROLE_NAME>

[role]
# namespace of a role
# placing the name of a role here will import the role as is.
# Will then be available as: <ANSIBLE_ROLE_NAMESPACE>.<ROLE_NAME>

[role-tasks]
# namespace of a role with tasks
# placing the name of a role here will import all tasks from that role.
# Will then be available as: <ANSIBLE_ROLE_NAMESPACE>.<TASK_NAME>

[scripts]
# Path to a folder where shell scripts (*.sh) and playbooks (.yml) reside.
# Use this for scripts that are not part of a role or collection.
# Will then be available as: <PARENT_FOLDER>.<FOLDER_WITH_SCRIPTS_OR_PLAYBOOKS>
# Recommend to use a username or organization name for the PARENT_FOLDER.
# The only variable available is '$HOME'. Hidden folders that start with '.'
# will be used without the dot.
```

vmh(1) – Advanced libvirt manager for smart mass and parallel deployment

DESCRIPTION

Advanced libvirt manager for smart mass and parallel deployment. Is a wrapper for libvirt and virsh.

- Mass and parallel deployment of domains
- Automatic deployment of virtual networks
- Create/compress/import/export images that use disk chains
- More verifications and better error handling than plain virsh
- More automation, e.g. auto shutdown and then force shutdown after a timeout
- Wait for SSH to become available
- Logging system (rsyslog)
- Versioning of libvirt templates
- Useful for automating kiosk applications that use VMs

SYNOPSIS

`vmh <ACTION> <DOMAIN|DESTINATION|DISK_NAME|ENVIRONMENT|CONFIG_TAG> [DOMAIN]|<CONFIG_VARIABLE>` : (virtualmachinehandler) Advanced actions for libvirt domains

`vmhc <DOMAIN> <DISPLAY_OPTION>` : (vmh connector) Simple wrapper for 'virt-viewer'. Connect to a domain. DISPLAY_OPTION can be 'k' for kiosk and 'f' for fullscreen.

`vmh-screenshot <DOMAIN> <OUT_FILE>` : Take a screenshot of a domain in ppm format. If the path OUT_FILE is ommited, the screenshot will be saved to `$HOME/shot_$DOMAIN.ppm`.

vmh will always install domains to the 'default' libvirt pool. This document will refer to this folder as 'POOL'.

The IMMUTABLE folder contains all templates. Is used to store disk images that are not supposed to be changed. This folder is also used to store libvirt network XMLs. This document will refer to this folder as IMMUTABLE.

To differentiate network templates form domain templates. Network templates must have the following filename format: 'libvirt-net-.xml'

OPTIONS

- `-h`, `--help` :
  Display this help screen.

CONFIG

Config file is in /etc/vmh.conf
Open the default config file for more information.

To create all required folders and set correct file permissions run:
`sudo vmh init`
This can be useful if you want to set up environment files before you have run vmh for the first time.

ACTION

Use the `<action>` variable to manipulate domains in different ways.

- `chain <DESTINATION> <DOMAIN>` : 'shutdown', create new empty disk, appends disk to chain and renames the domain to DESTINATION. DESTINATION is also used to name the disk files. This is helpful to see what images belong to what domain.
- `chain-start <DESTINATION> <DOMAIN>` : Same as 'chain' and 'start'.
- `chain-start-wait <DESTINATION> <DOMAIN>` : Same as 'chain', 'start' and ' wait'.
- `chain-info <DOMAIN>` : Show the disk chain in POOL and IMMUTABLE paths
- `chain-rebase-immutable <DISK_NAME>` : In the IMMUTABLE folder, change a disk's metadata to point to a relative path. Use this if you import disk chains that are not created with vmh.
- `destroy <DOMAIN>` : Hard shutdown the domain.
- `env-deploy <ENVIRONMENT>` : Smart deploy multiple domains in parallel or/and in series. Iterates through the file: `<PATH_IMMUTABLE>/<ENVIRONMENT>.csv` . Will automatically import and start networks from

XMLs, found in `<PATH_IMMUTABLE>/libvirt-net-<NETWORK>.xml` . See 'ENVIRONMENT FILE STRUCTURE'. Based on 'import-chain-start' and 'wait'.

- `env-erase <ENVIRONMENT>` : Works in reverse to 'env-deploy'. Based on the action 'erase'. Requires the ENVIRONMENT file.
- `env-erase-full <ENVIRONMENT>` : Same as 'env-erase' and then 'env-erase-net' combined.
- `env-erase-net <ENVIRONMENT>` : Erase all networks found in an environment CSV. Based on the action 'erase-net'. Requires the ENVIRONMENT file.
- `env-map <ENVIRONMENT>` : Generate an SVG image of the environment network. Files will be stored in the PATH_ENVIRONMENTS folder.
- `env-wait <ENVIRONMENT>` : Based on the action 'wait'. Requires the ENVIRONMENT file. Can be used to quickly test/verify an already deployed environment.
- `erase <DOMAIN>` : 'destroy', remove all disk chains and undefine the domain. Will also remove the host from the /etc/hosts file.
- `erase-net <DOMAIN>` : Erases all virtual networks defined in a domain template XML, not the deployed domain. Use this if there are no more domains that require a certain network. Or use it if you made changes to a networks XML template. Then reimport the network via 'import-...' or 'env-deploy' . Remember domains that where active during deletion of a network, require a full shutdown and start, to reload the new network settings.
- `export-copy <DESTINATION> <DOMAIN>` : 'shutdown', export the domain, XML and an exact disk copy. Will not strip snapshots or old disk data. Largest file. Only recommended if image is already compressed. Most compatible.
- `export-merge <DESTINATION> <DOMAIN>` : 'shutdown', export the domain, XML, compress and sparse the disk copy, can take a long time. Smallest disk image size. Can result in a none working image.
- `export-merge-fast <DESTINATION> <DOMAIN>` : 'shutdown', export the domain, XML and compress the disk copy, Faster than 'export-merge'. Reasonable disk image size. Sometimes more compatible than 'export-merge'.
- `get <CONFIG_TAG>` : Get a variable from the CONFIG file. For example 'vmh get debug'.
- `import-chain <DESTINATION> <DOMAIN>` : Define a domain from a template XML found in the IMMUTABLE folder. Automatically detects and links disk chains to libvirt's POOL folder. Also adds as a new disk to the chain. Requires two files: The disk image with the name 'DOMAIN' and a libvirt compatible XML named 'DOMAIN.xml'. Will automatically import and start networks from XMLs found in `<PATH_IMMUTABLE>/networks/<NETWORK>.xml` .
- `import-chain-start <DESTINATION> <DOMAIN>` : Same as 'import-chain' and ' start'. This is true for all 'import-...' actions.
- `import-chain-start-wait <DESTINATION> <DOMAIN>` : Same as ' import-chain', 'start' and 'wait'.
- `import-clone <DESTINATION> <DOMAIN>` : Define a domain from a template XML found in the IMMUTABLE folder. Automatically detects and copies disk chains to libvirt's POOL folder. Preferably use 'import-link' or 'import-chain', to save disk space.
- `import-link <DESTINATION> <DOMAIN>` : Define a domain from a template XML found in the IMMUTABLE folder. Automatically detects and links disk chains to IMMUTABLE folder. Please don't start the domain after this. Use ' chain' to add a new disk to write to. Else libvirt will try to write to the immutable disk instead, which is not recommended. Preferably use ' import-chain'.
- `init` : See CONFIG section.
- `list` : List all available environments.
- `purge <DOMAIN>` : Remove the disk image with the same name as the domain in the POOL folder. Use after 'erase' to remove a broken domain and linked disk images. Also removes the 'shared' folder and everything in it. Use carefully.
- `purge-environments` : Removes everything in the PATH_ENVIRONMENTS folder.
- `purge-immutable <DOMAIN>` : Remove the disk image with the same name as the domain in the IMMUTABLE folder. Also removes the XML file with the same name. Useful to clean up exported domains before exporting the same domain again.
- `set <CONFIG_TAG> <CONFIG_VARIABLE>` : Set a variable in the CONFIG file. For example 'vmh set debug true'.
- `shutdown <DOMAIN>` : OS shutdown, after 3 minutes do 'destroy'.
- `start <DOMAIN>` : Boot a VM and continue with shell execution.
- `start-wait <DOMAIN>` : Same as 'start' and 'wait'.
- `unlink` : Will unlink all symbolic links in the POOL folder. This will break symlinks. Useful if you want to clean up.
- `wait <DOMAIN>` : Wait for an IP and an open SSH port, will block shell execution. Timeouts are: get MAC 30 sec., get IP 3 min., get SSH 30 sec. On success will automatically add/update the host to the /etc/hosts file.

VERSIONING

The import- and env- actions can also handle versioning of libvirt templates. This is also true for the environment files. A specific version can be specified by using and underscore followed by the version of the libvirt XML. If not specified, the latest version will be used. See the man page of sort-by-version(1).

When using a libvirt template, an underscore followed by the version number, must be specified. For example 'my-template_1.xml' or 'my-template_0.3.xml'.

PLACEHOLDER VARIABLES IN XML

When importing and exporting a domain, the following variables will be replaced in the XML file. This is useful if you want to use the same XML file for multiple domains.
Make sure to use '/' before and after the variable. Else XML verification will fail.

- `/POOL_PATH_PLACEHOLDER/` : Will be replaced with the POOL path.
- `/SHARED_PATH_PLACEHOLDER/` : Will be replaced with the SHARED path /var/lib/libvirt/shared/

ENVIRONMENT FILE STRUCTURE

Each line represents a single domain. Which consists of three strings/settings, separated by a comma.
`<ORDER>,<DOMAIN_NAME>,<DOMAIN_SOURCE>`

- `ORDER` as integer : Is the group number for all domains to deploy in parallel. For example all domains with the order '1', will deploy in parallel and tested for SSH, before the next batch, with the order number '2' is deployed.
- `DOMAIN_NAME` as string : The libvirt name of the domain to deploy to.
- `DOMAIN_SOURCE` as string : The name of libvirt template to deploy from. Without the XML. For example 'my-template_1' or 'my-template_0.3'.

EXAMPLES

Start a single, already imported, domain with the name 'myvm'

```
$ vmh start myvm
```

Import the disk image 'my-disk-source' and start a new domain, as an image chain, with the name 'office'. Also boot/start the domain after that.

```
$ vmh import-chain-start office my-disk-source
```

Connect to a domain in fullscreen mode

```
$ vmhc myvm f
```

Take a screenshot of the domain 'myvm' and save it as 'myvm.ppm'

```
$ vmh-screenshot myvm myvm.ppm
```

Deploy the following environment file 'my-test1.csv' in two parallel turns. In practice this means that the 'gateway' domains will be up before deploying the office domains.

```
1,gateway-a,my-pfsense-image
1,gateway-b,my-pfsense-image
2,ws-office-win,my-windows-image
2,ws-office-lnx,my-debian-image
```

Run:

```
$ vmh env-deploy my-test1
```

Deploy the following environment file 'my-test2.csv' in series.

```
1,ws-office-a,my-debian-image
2,ws-office-b,my-debian-image
3,ws-office-c,my-debian-image
4,ws-office-d,my-debian-image
```

Run:

```
$ vmh env-deploy my-test2
```

LOGGING

Log file is /var/log/vmhandler.log

Enable logging

```
$ vmh-logging-enable
```

Disable logging

```
$ vmh-logging-disable
```

Watch logs

```
$ vmh-watch
```

Clear logs

```
$ vmh-clear-logs
```

COPYRIGHT

See license file

SEE ALSO

virsh, virt-sparsify, waitforit

autokiosk(1) – openbox kiosk starter for libvirt and web

DESCRIPTION

Open a URL or libvirt domain, as full-screen kiosk application. Automatically opens simple-kiosk, chromium or virt-viewer, depending on the URI passed.

USAGE

Syntax:

`autokiosk -h, --help`

`autokiosk [<URL>|<LIBVIRT_DOMAIN>|close-all|hide-all|hide-vms] [close|hide]`

- Will only open one instance of the URL or libvirt domain
- Best used with openbox
- `close` will close a specific processes
- `hide` will minimize a specific processes
- `close-all` will close all simple-kiosk, chromium and virt-viewer processes
- `hide-all` will minimize all simple-kiosk, chromium and virt-viewer processes
- `hide-vms` will minimize all virt-viewer processes

EXAMPLE

Open debian.org in simple-kiosk or Chromium full-screen:

```
$ autokiosk https://www.debian.org/
```

Open a VM named my.domain in virt-viewer fullscreen:

```
$ autokiosk my.domain
```

Close the just opened kiosk windows

```
$ autokiosk www.debian.org close
$ autokiosk my.domain close
```

Quit every simple-kiosk, Chromium and virt-viewer process:

```
$ autokiosk close-all
```

Minimize the virt-viewer process with the title ‘my.domain’:

```
$ autokiosk my.domain hide
```

Minimize all virt-viewer processes:

```
$ autokiosk hide-all
```

COPYRIGHT

See license file

SEE ALSO

openbox(1), simple-kiosk(1), chromium(1), virt-viewer(1)

spice-web-client(1) – HTML5 Spice Web Client

SYNOPSIS

`spice-web-client [OPTIONS] [PATH_WEB_ROOT]`

DESCRIPTION

Complete Spice Web Client written in HTML5 and Javascript.
See README.md for more information.

Domains will be installed to `PATH_WEB_ROOT/<DOMAIN>`

OPTIONS

- `-h`, `--help` : Displays the help screen.
- `i [PATH_WEB_ROOT]` : Install to a custom webroot directory. Default PATH_WEB_ROOT is /var/www/html.
- `u [PATH_WEB_ROOT]` : Uninstall from webroot directory. PATH_WEB_ROOT is optional. Default PATH_WEB_ROOT is /var/www/html.
- `f` : Force. This will force a reinstallation of spice-web-client. Only applicable when using the `i` option. This also restarts all websockify processes.
- `d` : Enable debug mode. This will show extra debug information.

LIBVIRT REQUIREMENTS

Your domain configuration must have the following spice section:

```
<graphics type="spice" port="7201" autoport="no" listen="0.0.0.0">
  <listen type="address" address="0.0.0.0"/>
  <image compression="auto_glz"/>
</graphics>
```

spice-web-client will effectivly ignore listen adresses that are not 0.0.0.0.
Image compression does not need to be auto_glz, but it is recommended because it is much faster.

The recommended port range is 7200-7300.

Setting 'autoport' in the XML will work just fine, but it is not recommended because it will make it harder to configure any firewall and managing websockify processes.

ENABLE WSS

spice-web-client will automatically enable WSS (WebSocket Secure) if the following files exist:

/etc/ssl/certs/spice-web-client.crt
/etc/ssl/private/spice-web-client.key

These should be the same certificates used by the webserver.

WSS domains will be installed to `PATH_WEB_ROOT/wss.<DOMAIN>`. This way both HTTP+WS and HTTPS+WSS can be used.

The WSS port is the same port as the libvirt SPICE server plus 1000.
For example, if the SPICE server is running on port 7203, then the WSS port will be 8203.
For unencrypted WS the port is the same as the SPICE server port.

EXAMPLES

Autoinstall all libvirt domains into seperate folders under /var/www/html

```
$ spice-web-client
```

Install to a custom webroot directory

```
$ spice-web-client i /var/my-web-root
```

Uninstall from default webroot directory and show extra debug information

```
$ spice-web-client ud
```

COPYRIGHT

See license file

favicon.ico from iconfinder.com/icons/3069182/

SEE ALSO

spice-space.org

servermonkeys-devtools(1) – Servermonkeys Development tools

DESCRIPTION

Just a bunch of tools I use to develop Debian and fastpkg packages.

LIST OF TOOLS

This is a collection of programs and scripts. After installing this as APT package you can just run each program directly from your shell. Most of them support a '-h' option for additional help.

- `add-repo_servermonkey` : Add APT repo muspekaren.se/repo-debian to sources.list.d, same as Ansible task: servermonkey.ww.cfg_apt-servermonkey
- `applist2exploitdb <IN> <OUT>` : Generate a list of possible exploits from a list of applications
- `apt-listlocalpkgs` : List all packages from the local apt repo
- `chmod-default <DIRECTORY>` : Recursively change file and folder permissions to default. Directories: 755, Files: 644
- `chmod-secret <DIRECTORY>` : Recursively change file and folder permissions to secret. Directories: 700, Files: 600
- `cinac-bootstrap [ -h | dl ]` : Install CinaC to a new unconfigured Debian installation, supports offline installation.
- `cinac-bootstrap.ini` : Example configuration file for cinac-bootstrap
- `cinac-unittest <OPTION>` : Create a VM, isntall CinaC and run a test blueprint. Useful for developing CinaC blueprints and CinaC itself. Run with '-h, –help' for more information.
- `dir2html <TITLE>` : Create a simple HTML index from current directory
- `guake-dc <PATH>` : Opens a new guake tab in the target path. Use it in DoubleCommander as terminal.
- `monkeyrepo-sftp-uploader` : Upload a fastpkg repo to a sftp-only server, uses the 'pass' tool
- `oldversion2fastpkg <SRC> <DES> <URL_PREFIX>` : Convert oldversion.com .tar archive to a fastpkg repo
- `remove-lines <TARGET_FILE> <FILE_WITH_LIST_OF_LINES>` : Remove lines from a file that are in another file
- `setup-cinac-unattended` : CinaC unattended installer example
- `upgrade-now` : Same as 'apt update and upgrade', also updates fastpkg if installed
- `vmclip` : Copy clipboard to VM without clipboard enabled in VM
- `websitedownloader2fastpkg <DIR_IN> <FILE_OUT> <WWW_ROOT>` : Convert a directory of zip files to a fastpkg packages repo file

LIST OF DOCUMENTS

- `dictionary` : Servermonkeys IT-dictionary for dummies
- `software-philosophy` : Servermonkeys software philosophy
- `windows-isos` : Note on Microsoft Windows ISO files

COPYRIGHT

servermonkeys-templates(1) – Servermonkeys collection of template files

DESCRIPTION

A collection of my personal templates, programming and automation related.

- **ansible** : Template to be used with wildwest and ansible2deb.
- **bootstrap-index** : HTML/CSS template that displays a list.
- **debian** : Genric Debian package make template.
- **ventoy** : ventoy.json unattended installer templates. Includes Debian preseeds and Windows Answer Files. See the README.md inside the ventoy folder for instructions.
- **packages_args.txt** : A list of silent installer arguments used by Windows applications.

USAGE

Copy each template folder to wherever you want to start your project:

```
$ cp -r /usr/share/servermonkeys-templates/templates/<TEMPLATE_FOLDER_NAME> <MY_PROJECT>
```

COPYRIGHT

See license file

isoremixer(1) – Automatic build Cygwin Windows ISO images

SYNOPSIS

`isoremixer <OS>`

DESCRIPTION

Automatically download and build a custom ISO image by combining Cygwin, wine, winetricks and fastpkg. Automatically includes virtualization drivers and deployment scripts for Cygwin and SSHD.

This project is still under development.

How isoremixer works:

1. Download required files from the Internet (via fastpkg). Like Windows ISO's, cygwin.exe installer, drivers and so on.
2. Create a custom Wineprefix.
3. Download Cygwin packages.
4. Install requirements like .NET for Nlite via winetricks.
5. Extract ISO content and drivers to be included in Nlite.
6. Load preset and script files from user or system folder.
7. Start builder program. Like Nlite.

If you encounter errors, log files can be found in each respective wineprefix under ~.local/share/wineprefixes/

OPTIONS

- `-h`, `--help` : Displays the help screen
- `r` : Use Cygwin fastpkg repacks instead of Cygwin mirror. Use this option when you want to use a specific collection of Cygwing packages. Useful when you are in an offline environment. Also, faster than downloading from a mirror.
- `s` : Skip fastpkg downloads. Only use this if you are sure that you have all required files in the fastpkg downloads folder.
- `c` : Only start Cygwin-setup. Only for debugging purposes. Then exit.
- `e` : Erase everything including VM, disks and Wine prefix. Use for cleanup.
- `<OS>` : Select an OS version, Se below for availabe OS:es.

OS ARGUMENT

- `xp` : Windows XP (any version) via Nlite
- `7` : Windows 7 x64 via NTLite and libvirt - UNDER DEVELOPMENT!!!
- `7-x86` : Windows 7 x86 via NTLite and libvirt - UNDER DEVELOPMENT!!!

EXAMPLES

Build a Windows XP ISO image:

```
$ isoremixer xp
```

CUSTOM CONFIG FILES

To use your own preset and/or script files, put your files in these folders.
If not, isoremixer will copy recommended/default preset and script files from: /usr/local/share/isoremixer/presets/

Presets

- `xp` : `"~/.isoremixer/presets/nlite/windows-xp-pro/Last Session.ini"` and `"Last Session_u.ini"`

Scripts

- `xp` : `"~/.isoremixer/presets/setup_cygwin*"` Each script file must start with `"setup_cygwin..."`

Mirrors

For older versions of Cygwin look at:
www.crouchingtigerhiddenfruitbat.org/cygwin/timemachine.html
cygwin.com/setup/
ctm.crouchingtigerhiddenfruitbat.org/pub/cygwin/circa/

COPYRIGHT

SEE ALSO

wine(1), winetricks(1), fastpkg(1), www.cygwin.com

Note on Microsoft Windows ISO files

Microsoft does not provide legacy releases of Windows. In fact they are actively removing links to Windows ISO files.

This is a nightmare for security researchers and system administrators. Because this makes it more or less impossible to create automated download scripts. How can you research a specific version of an operating system, if it is not available?

If you want to use Windows in a deterministic way. You have to get the ISO files elsewhere. As of writing this document you can search archive.org for older releases. But it is probably just a matter of time before the links change or disappear.

The website files.rg-adguard.net provides a collection of hashes. In this way you can at least determine if the files you have are correct.

If you need to build a deterministic environment offline, it is a good idea to host your own internal archive of Windows ISO's.

cinac2deb

Create a .deb package from a CinaC blueprint.

Tested on Debian 11

USAGE

```
cinac2deb -h
```

```
cinac2deb <BLUEPRINT_FOLDER_PATH> <MAINTAINER>
```

Example

```
cinac2deb ./my-blueprint 'Peter Miller <peter@example.com>'
```

Servermonkeys IT-dictionary for dummies

This dictionary is intended to assist people who are not familiar with the complexities of IT. It is specifically designed as a reference for my programs.

Agent: A computer program that is installed on a computer and used to monitor and manage that computer. It is like a spy that reports back to you and does what you tell it to do. Using or not using an agent is a matter of architecture and preference. Adding agents to a computer system always adds complexity.

Ansible: A computer program that is used to configure computers. It uses SSH to connect to the computers but is more versatile than just SSH and more suited for configuration management and automation. See 'SSH'

Ansible collection: Is a collection of Ansible roles and other files that are used to configure a computer system. It is like a LEGO set with multiple LEGO sets inside. See: 'Ansible role', 'Ansible namespace', 'Galaxy-collection'

Ansible role: Is a collection of Ansible playbooks and other files that are used to configure a computer system. It is like a LEGO set. Usually a role is made of multiple Ansible tasks.

Ansible task: Is a computer program that is used to configure a computer system. It is like a LEGO figure that is part of a LEGO set.

Ansible namespace: In Ansible, a namespace is like a container or a label that groups together related things like variables, functions, or modules. Think of it like a folder on your computer. A namespace usually references to an Ansible role or an Ansible collection.

API (application programming interface): Is a way for two or more computer programs to communicate with each other.

APT (Advanced Packaging Tool): A package/software manager for Debian and derived. It is similar to an app-store on a smartphone.

Asset inventory: A list of all the computers and other devices that are connected to a network. It is like an inventory list for the IT department. If used as a SSOT it can also be a shopping and a to-do list. This makes IT-management very easy. See. 'SSOT'

Debian: An operating system that is based on the Linux kernel. Unlike Windows and macOS, Debian is free and open source. Can be used by governments or companies without paying a license fee. It's OpenSource nature and UNIX based architecture makes it easier to develop for. Debian can also be seen as a framework with a CLI API. It comes with over 50000 programs, many of those programs can be combined into automated products. See: 'CLI' and 'API'

Debian PackageManagement system: See 'APT'

Debian preseed: A file that is used to automate the installation of Debian. It is similar to an unattended installation of Windows. See: 'Ventoy'

Debian repo: A collection of software packages that can be installed on a Debian system. It is similar to an app-store on a smartphone with the addition that you can create your own store. See 'APT'

Derived Debian Distribution: Is a Debian distribution that is based on another Debian distribution. For example, Ubuntu is derived from Debian. A derived distribution usually includes preinstalled or custom packages. It is like buying a car, then painting it with a new color or adding a new stereo system.

CLI (Command-line interface): A computer program that is used to interact with a computer system in a text based fashion. Much like a chat program. CLI enables you to automate tasks and is more versatile than a graphical interface but initaly has a harder learning curve.

Cloud platform: Is a computer system that allows users to access software and store data over the internet, instead of their own computer. Usually maintained by a company that charges a monthly subscription fee.

CSV (Comma Separated Values): A file format that is used to store tabular data. Much like Excel, but less complex. It can easily be processed by a human or a computer. Software like Excel or libreOffice can be used to view and edit.

Galaxy-collection: Are Ansible namespaces that are available online at galaxy.ansible.com. They are like a collection of LEGO bricks that can be used to build something. See: 'Ansible namespace'

Golden Image: Is a pre-configured template of a computer system. Mostly it has all the necessary software and configurations already installed. It is used to quickly create multiple identical copies of the same computer system. Think of it like a cookie cutter that can be used to create many cookies that all look and taste the same. See: 'Hypervisor'

Guest computer: A computer in a network that is controlled by another computer. See: 'Host computer'

Host computer: The main computer in a network that controls other computers. See: 'Guest computer'

Hypervisor: Is a software that allows multiple virtual machines (VMs) to run on a single physical computer. VMs are like simulated computers that can run their own operating systems and software. This allows multiple users or applications to share the same physical resources of a computer. The word Hypervisor is a marketing term, a better word would be 'virtualization platform'.

Hyper-converged infrastructure (HCI): Is a marketing term simply meaning that all parts of an IT-infrastructure are virtualized. The word has no real meaning because it is not useful. It is like saying that a car has wheels. This is because all modern IT-infrastructures are virtualized in one way or another.

IaC (Infrastructure as code): Is a way of creating and managing IT-infrastructure using computer programs. It is like sheet music but for computer infrastructure. Instead of playing by ear, a music sheet (code) can be easily changed, shared and will produce the same results every time.

Immutable host/server: Is a computer that once deployed, is never modified, merely replaced with a new updated instance. An immutable host is like a car that you can't change once it's made. This makes sure that you get exactly the same computer system each time you build it.

INI (initialization): A file format that is used to store configuration data. It is similar to JSON and YAML, but less complex. It can easily be processed by a human or a computer.

Kiosk application: Is a computer program designed to run on a self-service kiosk or touch screen device, allowing users to interact with the program and complete tasks without needing assistance from a person. Kiosk applications are usually run on Thin clients.

KVM (Kernel-based Virtual Machine): Is a virtualization technology built into Linux. Lets you turn Linux into a virtualization platform. See: 'QEMU', ' Hypervisor'

libvirt: Is a virtualization platform. It is similar to VirtualBox or VMWare ESXi. Can be used by governments or companies without paying a license fee. See ' Hypervisor', 'KVM' and 'QEMU'

Microservice: Is usually a web application, that is part of a larger network of web applications that work together. This makes the program easier to maintain and scale as it grows larger and more complex.

MIT license: Is a legal agreement that allows people to use and modify software for free. It's like sharing your toys with friends and telling them they can play with them as much as they want and even change them if they want to. However, the MIT license also says that if you use the software, you have to give credit to the person who made it and not hold them responsible if something goes wrong. It's a way for people to share their work with others and let them build upon it in a fair and open way.

Monolithic software architecture: A way of building computer programs where all the different parts of the program are built and run as a single piece of software. This means that if one part of the program needs to be updated or changed, the entire program needs to be updated or changed at once. It can make the program harder to maintain and scale as it grows larger and more complex.

Nested virtualization: When running virtual machines within virtual machines. It is like the movie with Leonardo DiCaprio, 'Inception' but for computer infrastructure.

Provisioning: Is a confusing term that, depends on whom you ask, can mean all kinds of things. Sometimes it means installing software, sometimes it means creating a virtual machine, sometimes it means creating a user account. It is like saying 'I am going to the store' but not specifying what you are going to buy. Sometimes it is used interchangeably with 'deployment' sometimes 'provisioning' is a part of the 'deployment' process. It usually means that you are creating a computer system from scratch and also configuring it. It is better to use the words 'automated configuration' or 'automated installation'. See 'deployment'

Deployment Is a confusing term that depends on whom you ask can mean all kinds of things. See: 'Provisioning'

QEMU (Quick Emulator): Is a computer program that emulates a machine's processor. It can interoperate with KVM. An emulator is like a pretend computer that can act like a different real computer. See 'KVM'

Shell: A computer program that is used to interact with a computer system in a text based fashion. A user interface much like a chat program.

SSH (Secure Shell Protocol): A computer program that is used to securely connect to a remote computer. Used to copy files or control programs on the remote computer. See: 'CLI' and 'Shell'.

SSOT (Single Source Of Truth): Is a way of organizing information so that there is only one place where it is stored. This makes it easier to keep track of information and prevents mistakes from happening.

Unix (*nix): Is an operating system known for being very powerful and reliable. It is used by many big companies, universities and governments around the world. It's also the foundation of many other operating systems like macOS and Linux (including distributions like Debian). Most modern IT-infrastructure is based on the legacy of Unix and the culture behind it.

Ventoy: Is a computer program that is used to easily create a bootable USB drive. A bootable USB drive is like a special key that can start your computer and run different programs, like installing a new operating system or fixing computer problems. With Ventoy, you can add many operating systems to the same USB drive. This makes it really convenient. See: 'Debian preseed'

Virtualization platform: See 'Hypervisor'

Web application: Is a computer program that is stored on a remote computer and accessed by users through a web browser. To use a web application, you often need an Internet connection. Web applications are more complex than command-line interface (CLI) applications. Modern companies often prefer web applications because they can offer services that users can pay for on a subscription basis.

Thin client: Is a computer that relies on a server to do most of its work. Mostly used as user interfaces for Kiosk application or web applications. A thin client is usually small and cheap. Almost any modern computer, Laptop, Phone or Tablet can be turned into a Thin Client. Thin clients are different from web applications because web applications can be run inside a thin client, but thin clients are more powerful because they can also use local resources and programs. When used with kiosk applications, thin clients can prevent user errors such as accidentally closing a browser tab. See 'Kiosk application', ' Web application'