GNU Guix

Gerenciador de pacotes transacional e Distribuição

Wallysson Oliveira, Leonardo Valente, Lucas Bortoleto

O que é o projeto GNU?

GNU é um sistema operacional livre, ou seja, que respeita a liberdade de seus usuários.

O sistema operacional GNU consiste em pacotes GNU (programas especificamente lançados pelo Projeto GNU) bem como software livre lançado por terceiros.

O desenvolvimento do GNU tornou possível o uso de um computador sem um software que ameace sua liberdade.

O gerenciador de pacotes e distribuição (GNU/Linux, Hurd) Guix faz parte do projeto GNU.

O que é um gerenciador de pacotes transacional?

Tal como um Banco de Dados, cada operação de um gerenciador de pacotes transacional é uma transação, ou seja, ou a operação é um sucesso ou nada acontece (o estado atual é mantido).

Guix também mantém cada estado salvo, chamados de gerações, permitindo ao usuário "voltar no tempo" com o comando:

```
guix package -S [PATTERN]
```

Onde PATTERN pode ser um número de geração ou um número prefixado com “+” ou “-”.

Você pode listar as gerações com o comando:

```
guix package -l [PATTERN]
```

Caso nenhum PATTERN seja enviado, Guix irá mostrar todas as gerações.

Exemplo:

```
~ $ guix package -l *| grep "Generation"
Generation 290  May 26 2025 02:30:45
Generation 291  May 30 2025 20:50:30
Generation 292  May 30 2025 21:40:44
Generation 293  May 30 2025 23:53:32
Generation 294  May 31 2025 01:40:23
Generation 295  Jun 12 2025 21:07:42
Generation 296  Jun 12 2025 21:08:14
Generation 297  Jun 12 2025 21:57:03
Generation 298  Jun 12 2025 22:28:54
Generation 299  Jun 12 2025 22:32:00
Generation 300  Jun 12 2025 23:11:54
Generation 301  Jun 12 2025 23:41:55
Generation 302  Jun 13 2025 00:25:33
Generation 303  Jun 17 2025 21:12:11
Generation 304  Jun 22 2025 21:36:09
Generation 305  Jun 22 2025 22:34:51
Generation 306  Jun 22 2025 22:37:33
Generation 307  Jun 22 2025 22:38:26
Generation 308  Jun 22 2025 22:39:24
Generation 309  Jun 22 2025 22:40:50
Generation 310  Jun 22 2025 22:43:24
Generation 311  Jun 22 2025 22:44:07
Generation 312  Jun 22 2025 22:45:09
Generation 313  Jun 22 2025 22:45:55
Generation 314  Jun 22 2025 22:46:43
Generation 315  Jun 22 2025 22:47:27
Generation 316  Jun 22 2025 22:48:10
Generation 317  Jun 22 2025 22:49:15
Generation 318  Jun 22 2025 22:50:13
Generation 319  Jun 22 2025 22:51:04
Generation 320  Jun 22 2025 22:52:23
Generation 321  Jun 22 2025 22:53:46    (current)
```

```
~ $ guix package -S 320
switched from generation 321 to 320
~ $ guix package -S 321
switched from generation 320 to 321
```

Mas isso não ocupa muito armazenamento?

A resposta curta é: Sim!

A resposta longa é: Sim! E por isso Guix vem equipado com um Coletor de Lixo que pode ser executado pelo usuário a qualquer momento comando através do comando:

```
guix gc
```

Guix consegue determinar quais pacotes ainda são utilizados através dos perfis e remover aqueles que não estão mais em uso.

Além disso, o coletor de lixo pode também remover gerações muito antigas, as quais o usuário possivelmente não irá querer voltar, através do comando:

```
guix gc -d [PATTERN]
```

Guix permite mais!

Todas as definições do Guix e de seus pacotes são controladas por versão, e o guix pull permite que você “viaje no tempo” na história do próprio Guix (como um grande repositório Git).

O comando:

```
guix pull -l PATTERN
```

Permite ao usuário listar todas as gerações do Guix e outros canais.

Similar ao guix package, também é possível "voltar no tempo", através do comando:

```
guix pull -S PATTERN
```

Guix ainda permite mais!

É possível criar ambientes de trabalho, com versões específicas de pacotes através do comando:

`guix shell [LISTA_DE_PACOTES]`

Permitindo, assim, ambientes de teste e desenvolvimento. Além disso, é possível conteinerizar esses sistemas através da flag -C ou –container, tornando o sistema completamente isolado.

Mas e o Sistema Operacional?

Além de um gerenciador de pacotes incrível, Guix pode também ser instalado como um sistema operacional completo.

Trazendo em si todas as propriedades do gerenciador de pacotes, mas agora no sistema.

Sendo assim, é possível versionar todas as suas configurações de home, popularmente conhecidas como dotfiles, através de comandos:

```
guix home
```

Por exemplo, o comando:

```
guix home reconfigure [PATH]
```

Permite que uma transação de sua atual configuração da home seja feita, ou seja, caso falhe, nada ocorrerá!

Guix System

Guix também permite controle total do Sistema Operacional como transações de pacotes através de comandos:

`guix system`

Permitindo a reconfiguração total do sistema de forma transacional com:

`guix system reconfigure [PATH]`

Tanto guix home quanto guix system permitem o retorno a gerações antigas caso algo falhe, por exemplo, imagine o seguinte cenário:

Sua configuração do extensível, customizável e livre editor de texto Emacs deixou de funcionar após uma atualização que você realizou na configuração.

Após muito tempo buscando o problema você percebe que já são 2 horas da tarde de uma terça feira e você está atrasado para a aula de Sistemas Operacionais, o que fazer?

Um simples:

```
guix home roll-back
```

Retornaria sua configuração para o estado anterior, onde tudo funcionava corretamente!

Agora imagine um cenário pior:

Após uma atualização de sua configuração de sistema algo deu errado, seu teclado não funciona como esperado e seu mouse está invertido e você está atrasado para a apresentação de seu seminário em Sistemas Operacionais , um simples:

```
guix system roll-back
```

Retornaria todo seu sistema para o estado anterior.

Seu sistema Guix é replicável e de fácil redistribuição!

Guix system também possui o poder de gerar uma imagem atual do seu sistema, isso mesmo, uma cópia exata de seu sistema operacional atual, com as mesmas configurações e pacotes, através do comando:

```
guix system image
```

Por exemplo, caso você tenha um pendrive montado em /dev/sbc, você pode criar uma cópia de seu sistema atual com o comando

```
dd if=$(guix system image [PATH]) of=/dev/sdc
```

Onde PATH é leva para sua configuração atual.

E Fim! Você já pode instalar seu sistema atual em outra máquina, pronto para uso.

Mais contêineres

Além da criação de um .iso pronto para uso, você pode também criar contêineres com configurações e pacotes específicos através do comando:

```
guix system image -t docker [PATH]
```

Por fim, o contêiner pode ser lançado com Docker através dos comandos:

```
image_id="$(docker load < [PATH_TO_IMAGE].tar.gz)"
container_id="$(docker create $image_id)"
docker start $container_id
```

Mas como tudo isso é configurado?

Tanto o gerenciador de pacotes, como o sistema operacional GNU Guix são configurados em GNU Guile. tal como seu init system GNU Shepherd, que é uma implementação de Scheme.

Por exemplo, a configuração do sistema é definida como:

```
(operating-system
  (kernel linux)
  (initrd microcode-initrd))
  ((firmware (list linux-firmware))
  (locale "en_US.utf8")
  (timezone "America/Sao_Paulo")
  (keyboard-layout (keyboard-layout "br" "abnt2"))
  (host-name "Sholum")

  ;; The list of user accounts ('root' is implicit).
  (users (cons* (user-account
                 (name "Sholum")
                 (comment "Wallysson")
                 (group "users")
                 (home-directory "/home/Sholum")
                 (supplementary-groups '("wheel" "netdev" "audio"
                                         "video" "cgroup" "kvm"
                                         "libvirt" "dialout")))
                %base-user-accounts))

  ;; Packages installed system-wide.  Users can also install packages
  ;; under their own account: use 'guix search KEYWORD' to search
  ;; for packages and 'guix install PACKAGE' to install a package.
  (packages (append (list (specification->package "emacs")
                          (specification->package "efibootmgr"))
                    %base-packages))

  ;; Below is the list of system services.  To search for available
  ;; services, run 'guix system search KEYWORD' in a terminal.
  (services
   (append (list
                ;; To configure OpenSSH, pass an 'openssh-configuration'
                ;; record as a second argument to 'service' below.
                (service openssh-service-type)
                (service tor-service-type)
                (service cups-service-type)
                (service iptables-service-type)
                (service rootless-podman-service-type
                         (rootless-podman-configuration
                          (subgids
                           (list (subid-range (name "Sholum"))))
                          (subuids
                           (list (subid-range (name "Sholum"))))))
                (simple-service 'podman-subuid-subgid
                                subids-service-type
                                (subids-extension
                                 (subgids
                                  (list (subid-range (name "Sholum"))))
                                 (subuids
                                  (list (subid-range (name "Sholum"))))))
                (service libvirt-service-type)
                (set-xorg-configuration
                 (xorg-configuration (keyboard-layout keyboard-layout))))

           ;; This is the default list of services we
           ;; are appending to.
```

```
           ;; This is the default list of services we
           ;; are appending to.
           (modify-services %desktop-services
                            (guix-service-type config =>
                                               (guix-configuration
                                                (inherit config)
                                                (substitute-urls
                                                 (append (list "https://substitutes.nonguix.org")
                                                         %default-substitute-urls))
                                                (authorized-keys
                                                 (append (list (local-file "./signing-key.pub"))
                                                         %default-authorized-guix-keys))))))
  (bootloader (bootloader-configuration
               (bootloader grub-efi-removable-bootloader)
               (targets (list "/boot/efi"))
               (keyboard-layout keyboard-layout)))

  ;; The list of file systems that get "mounted".  The unique
  ;; file system identifiers there ("UUIDs") can be obtained
  ;; by running 'blkid' in a terminal.
  (file-systems (cons* (file-system
                         (mount-point "/home")
                         (device (uuid
                                  "72267a24-e934-46d8-8a17-5c89ab622c7f"
                                  'ext4))
                         (type "ext4")
                         (flags '(no-atime)))
                       (file-system
                         (mount-point "/")
                         (device (uuid
                                  "4079f732-9855-4a4a-893e-2642664fb301"
                                  'ext4))
                         (type "ext4")
                         (flags '(no-atime)))
                       (file-system
                         (mount-point "/home/Sholum/sata")
                         (device (uuid
                                  "f8b62d74-b863-48fa-a048-374ed0a67e5a"
                                  'ext4))
                         (type "ext4")
                         (flags '(no-atime)))
                       (file-system
                         (mount-point "/boot/efi")
                         (device (uuid "55CD-33E5"
                                       'fat32))
                         (type "vfat"))
                       %base-file-systems)))
```

Por que Guile?

Guile é uma implementação da linguagem Scheme, que por sua vez é um LISP, também parte do projeto GNU.

Por ser um Scheme é extremamente fácil de ser estendida através de macros e funções que rodam em tempo de compilação, expansão, leitura ou execução.

Facilitando a criação de linguagens de domínio específicos (DSLs), como a própria configuração do Guix mostrada acima.

Além disso, possui um rico ecossistema e uma comunidade fortemente ativa. Dentre projetos que utilizam Guile, merecem destaque:

1. Guix, que possui código Guile em seu core, além de ser a linguagem oficial de configuração;
2. Goblins, projeto que traz uma série de abstrações para lidar com concorrência paralelismo em sistemas distribuídos. Assim o programador pode se concentrar na programação de objetos e não na arquitetura de protocolos;
3. Fibers, projeto que traz um modelo de concorrência similar a implementada na linguagem Go para o Guile.

Outras linguagens dentro do Guile

O compilador do Guile também possui a implementação de outras linguagens, como EmacsLisp e ECMASscript.

Elas são compiladas para a mesma linguagem intermediária, chamada de tree-il, e por fim, executadas pela mesma VM, permitindo assim a comunicação de diferentes linguagens entre si.

A comunidade vem buscando implementar uma versão de Python e Lua, mas toda linguagem é bem aceita!

Aplicações práticas:

Atualmente o uso de Guix vem crescendo muito na Indústria e na Academia, pelo mesmo motivo: reprodutibilidade

Como explicado anteriormente, é muito simples recriar e distribuir o sistema Guix com configurações e pacotes específicos, facilitando a replicabilidade d pesquisas como mostram os papers Reproducible genomics analysis pipelines with GNU Guix e Reproducible and User-Controlled Software Environments in HPC with Guix.

Outra experiência tem sido a minha no meu atual emprego na empresa Buzzlabs, o uso do Guix tem sido estudado para o desenvolvimento de contêineres e a realização de Deploy dos produtos da empresa.

A ideia é a criação de contêineres específicos para produtos específicos, utilizados tanto em desenvolvimento, como na criação de testes, como na distribuição.

Guix possui uma série de ferramentas que podem ser utilizadas para facilitar esse processo, dentre eles, o comando:

```
guix deploy [PATH]
```

Que permite reconfigurar servidores não localmente.

Imagine o cenário em que nós possuímos temos que atualizar uma aplicação e, por conta disso, todos os nossos servidores serão também atualizados.

O guix deploy facilita esse processo como mágica, carregando a mesma configuração em diferentes máquinas através da web.

A lista de máquinas a serem reconfiguradas se encontram no arquivo, escrito em Guile, no PATH, como no exemplo abaixo:

```
(define %system
  (operating-system
   (host-name "gnu-deployed")
   (timezone "Etc/UTC")
   (bootloader (bootloader-configuration
                (bootloader grub-bootloader)
                (targets '("/dev/vda"))
                (terminal-outputs '(console))))
   (file-systems (cons (file-system
                        (mount-point "/")
                        (device "/dev/vda1")
                        (type "ext4"))
                       %base-file-systems))
   (services
    (append (list (service dhcp-client-service-type)
                  (service openssh-service-type
                           (openssh-configuration
                            (permit-root-login #t)
                            (allow-empty-passwords? #t))))
            %base-services))))

(list (machine
       (operating-system %system)
       (environment managed-host-environment-type)
       (configuration (machine-ssh-configuration
                       (host-name "localhost")
                       (system "x86_64-linux")
                       (user "alice")
                       (identity "./id_rsa")
                       (port 2222)))))
```


Guix