Void Linux

O sistema operacional Void

O **Void Linux** é um sistema operacional, descrito como um sistema operacional de uso geral, baseado no *kernel* monolítico[1] do Linux. É uma distribuição[2] de Linux independente, desenvolvida completamente por voluntários, sem se basear nem derivar de outra distribuição existente. [1]

A distribuição[2] Void possui diversas características e funcionalidades que a distinguem das demais distribuições Linux, como o uso do gerenciador de pacotes **XBPS** (*X Binary Package System*), que foi projetado e implementado do zero e conta com seus próprios repositórios, apoio e suporte para as implementações da biblioteca padrão do C[3] **musl** (ou **musl libc**) e **glibc** (ou **GNU libc**), o uso do **runit**[4] no papel de sistema de inicialização e supervisor de serviços, um sistema de atualizações contínuas com foco em estabilidade ("**stable rolling-release**"") e compilação de pacotes através do construtor de pacotes XBPS, o **xbps-src**.[5] [1, 3]

Em Abril de 2025, o Void Linux mantém um repositório oficial no GitHub com milhares de pacotes, incluindo pacotes *"nonfree"* e *multilib*, com 2.695 mil estrelas e 1.343 colaboradores. O repositório do XBPS por sua vez conta com 875 estrelas e 59 colaboradores. A comunidade do Void Linux é ativa e está presente em vários lugares da internet, como o subreddit oficial r/voidlinux, que conta com 16.156 membros inscritos e o canal de IRC `#voidlinux` na rede Libera.Chat. [1, 4, 5]

História e histórico de desenvolvimento

Void Linux foi criado em 2008 por Juan Romero Pardines[6], um ex-desenvolvedor do NetBSD, e surgiu inicialmente como um ambiente de testes para o gerenciador de pacotes XBPS, desenvolvido do zero para a distribuição. [6] A distribuição oferece um sistema enxuto e modular, privilegiando ferramentas próprias (como o XBPS) e dando aos usuários maior controle sobre as configurações. Como distro de lançamento contínuo, o Void recebe atualizações constantes sem necessidade de reinstalação periódica, garantindo que os binários acompanhem sempre as versões mais recentes das aplicações. [3, 6]

Ao longo dos anos, o desenvolvimento do Void Linux passou por marcos importantes. Em 2014, por

---

[1] Tipo de *kernel* (núcleo) onde todos os serviços principais do sistema rodam no mesmo espaço de memória (espaço de *kernel*).

[2] Muitas vezes chamadas de 'distros', as distribuições Linux são sistemas operacionais que inclui o *kernel* Linux para fornecer suas funcionalidades principais de núcleo.

[3] Frequentemente abreviado como libc, aqui se referem as implementações da biblioteca padrão da linguagem de programação C, seguindo a especificação do padrão ISO C, sobre a API de chamadas de sistema do Linux.

[4] O runit é um sistema de inicialização e gerenciamento de serviços para sistemas Unix-like, que inicia, supervisiona e finaliza processos. [2]

[5] O xbps-src também foi escrito do zero, e é capaz de ser usado para compilar software em ambientes isolados sem exigir root, com suporte a *cross-compilation* e múltiplas bibliotecas C.

[6] Conhecido como 'xtraeme'.

exemplo, a equipe optou por abandonar o **systemd**[7] em favor do **runit** como sistema de inicialização principal [3], reforçando o objetivo de manter o sistema leve e fácil de auditar. Naquela mesma época, o Void tornou-se a primeira distribuição Linux a usar o **LibreSSL** como biblioteca de criptografia padrão, substituindo o **OpenSSL**. [7] No entanto, em 2021 a equipe anunciou oficialmente o retorno ao OpenSSL (efetivado em 5 de março de 2021) devido à complexidade de manter patches necessários para o LibreSSL. [8] Esses ajustes refletiram a filosofia prática do projeto: adotar tecnologias enxutas, mas reverter mudanças que compliquem o suporte a softwares amplamente usados. A cada lançamento, o Void também incorpora melhorias de compatibilidade (como suporte completo a arquiteturas **ARM**[8] e **ARM64**[9]) e mantém builds nativos através do sistema **xbps-src**, inspirado nas coleções de ports do BSD.

Atualmente, o Void Linux segue como um projeto de comunidade com poucos desenvolvedores voluntários. O projeto sobrevive graças ao esforço conjunto de líderes de infraestrutura, manutenção de pacotes e documentação, todos atuando em tempo livre e decidindo coletivamente os rumos do desenvolvimento. [3, 6] Em 2018, o projeto enfrentou uma crise organizacional quando o mantenedor original desapareceu, forçando a equipe a **recriar a organização no GitHub** e realocar domínios para manter o controle do repositório. [9, 10] Esse episódio reforçou a estrutura colaborativa: decisões importantes agora são tomadas por consenso entre os principais contribuidores. Apesar dos desafios organizacionais, o Void Linux tem recebido elogios por sua agilidade e design minimalista - por exemplo, chegou a aparecer entre as distribuições **mais bem cotadas no DistroWatch**, mantendo bom nível de estabilidade para uso diário. [7]

Gerenciador de serviços e sistema de inicialização runit

Diferentemente da maioria das distribuições Linux modernas, o Void Linux utiliza o runit como sistema de inicialização (*init*) e para a supervisão de serviços. [3] O runit é uma suíte de inicialização Unix minimalista, criada por **Gerrit Pape**, que organiza o boot em três estágios: inicialização única do sistema, execução contínua de serviços e desligamento. [2] Na prática, ao entrar no *Stage 2* o runit executa o programa `runsvdir`, que escaneia diretórios de serviços e os inicia simultaneamente, resultando em um boot rápido e determinístico. Esse design segmentado contribui para a confiabilidade e agilidade do Void: o código executado como processo 1 (`PID 1`) é **muito pequeno**, o que facilita auditoria e reduz pontos únicos de falha. [2] Os *"runlevels"* são gerenciados pelas ferramentas internas do runit (`runsvdir` e `runsvchdir`), e o sistema resolve automaticamente dependências simples entre serviços durante a inicialização. [2]

Cada serviço no Void sob runit **é representado por um diretório** dedicado dentro de `/etc/sv/` (ou `/var/service/`). Esse diretório deve conter obrigatoriamente um script executável chamado `run`, que inicia o serviço em primeiro plano. [3] Opcionalmente, podem existir outros arquivos nesse diretório: um `check` para testar se o serviço está ativo, um `finish` para procedimentos de parada, um arquivo `conf` com variáveis de ambiente e um subdiretório `log` para logs dedicados. [3] Ao habilitar um serviço (por exemplo, criando um link simbólico em `/var/service/`), o runit automaticamente cria uma pasta `supervise` na primeira execução, começando a monitorar o processo. Com esse esquema, cada reinicialização de serviço ocorre em um ambiente consistente e isolado, e o runit mantém um *pipeline* de logs ativo enquanto o serviço estiver em execução. [3] Caso seja necessário desligar ou reinicializar o sistema, o runit interrompe todos os serviços supervisionados e executa o *Stage 3* (`/etc/runit/3`), que finaliza tarefas do sistema e realiza o *halt* ou *reboot*. [2]

[7] Systemd é um sistema de init usado em várias distribuições Linux, responsável por inicializar o sistema, gerenciar serviços em segundo plano e controlar o estado geral do sistema, com foco em paralelismo e dependências entre processos.

[8] ARM é uma arquitetura de processador de 32 bits, comum em dispositivos móveis e embarcados.

[9] ARM64 (ou AArch64) é a versão de 64 bits da arquitetura ARM, usada em dispositivos mais modernos, oferecendo melhor desempenho e suporte a mais memória.

Comparação entre glibc e musl

As bibliotecas C são componentes centrais de qualquer sistema Linux. A **GNU C Library** é a implementação mais difundida, usada por padrão na maioria das distribuições populares (**Debian**, **Ubuntu**, **Fedora** etc.). [11] Criada em 1988 sob a licença LGPL©, a glibc fornece extensões específicas do GNU e optimizações que melhoram o desempenho em tempo de execução. [11] Em contrapartida, o musl é uma alternativa lançada em 2011 com **licença MIT** (mais permissiva), concebida para ser simples, eficiente em recursos e estritamente compatível com padrões POSIX. [3, 11] Em vez de incorporar várias extensões de plataforma, o musl mantém uma base de código enxuta e estruturada para oferecer correção e segurança, tornando o sistema geralmente mais fácil de auditar e resulta em binários menores.

Na prática, escolher entre glibc e musl traz diferenças em **compatibilidade e performance**. A glibc possui um conjunto rico de funcionalidades. Por exemplo, suporta *lazy binding* de bibliotecas (carregamento sob demanda), robustos recursos de *threading* e extensões de plataforma (como otimizações para arquiteturas específicas). [11] Com isso, muitos softwares comerciais e drivers proprietários (como o **CUDA da NVIDIA**) só oferecem suporte oficial em sistemas glibc. [3, 11] Em contrapartida, compilação com musl tende a produzir executáveis menores e conclui builds mais rapidamente, embora em alguns cenários de uso intensivo de recursos o desempenho em tempo de execução possa ser ligeiramente inferior. [11] Além disso, devido ao tamanho do glibc, ele consome mais memória e tem tempos de compilação mais longos do que o musl, mas se beneficia de ferramentas adicionais (*sanitizers*, etc.) não presentes no musl. [11]

O Void Linux destaca-se por oferecer **suporte oficial a ambas as bibliotecas C**: é possível instalar o sistema-base utilizando glibc ou musl, conforme a preferência do usuário. [3] Na prática, todos os pacotes compatíveis estão disponíveis nas versões glibc e musl, permitindo alternar quando necessário. [3] Uma instalação baseada em musl tende a resultar em um sistema final de *footprint* menor (devido aos binários compactos) e inicialização potencialmente mais rápida, mantendo funcionalidades equivalentes às de uma instalação glibc. [3, 11] No entanto, vale lembrar que o **musl segue estritamente os padrões POSIX e não inclui extensões próprias do GNU**, por isso, certos softwares podem exigir ajustes para compilar ou funcionar corretamente em musl. [3] Adicionalmente, alguns programas proprietários não suportam musl (como os drivers oficiais da NVIDIA) [3], o que pode influenciar a escolha da libc em sistemas que dependem desse tipo de software.

O sistema de pacotes XBPS

O **XBPS** (*X Binary Package System*) é o **gerenciador de pacotes nativo do Void Linux**, criado inteiramente do zero pela equipe do Void. [1] De licença BSD simplificada (2 cláusulas), o XBPS foi desenvolvido internamente e não é um fork de outro sistema existente. [1] Ele é **extremamente rápido** e permite instalar, atualizar e remover software de forma ágil. Os pacotes binários do Void são pré-compilados e assinados; alternativamente, o usuário pode optar por compilar um pacote a partir do código-fonte, usando a coleção de pacotes-fonte XBPS. Segundo o site oficial, "o sistema de pacotes do Void permite instalar, atualizar e remover software rapidamente; o software é fornecido em pacotes binários ou pode ser construído diretamente de fontes". Durante a instalação ou remoção de pacotes, o XBPS realiza **checagens de integridade**, identificando bibliotecas compartilhadas incompatíveis e verificando dependências para evitar quebras acidentais no sistema.

Os usuários interagem com o XBPS por meio de várias ferramentas de linha de comando. Entre os principais comandos estão:

- `xbps-query`: pesquisa e exibe informações sobre pacotes instalados localmente ou disponíveis nos repositórios configurados.

- `xbps-install`: instala ou atualiza pacotes binários e sincroniza os índices dos repositórios.
- `xbps-remove`: remove pacotes instalados do sistema (incluindo pacotes órfãos e arquivos em cache).
- `xbps-reconfigure`: reexecuta scripts de configuração de pacotes já instalados, útil para reconfigurar software após alterações em arquivos de configuração.
- `xbps-alternatives`: gerencia o sistema de alternativas (semelhante ao update-alternatives do Debian), permitindo que múltiplos pacotes forneçam implementações diferentes de um mesmo recurso comum.
- `xbps-pkgdb`: verifica e corrige problemas no banco de dados de pacotes local.
- `xbps-rindex`: cria ou atualiza repositórios locais de pacotes binários a partir de diretórios já populados, útil para criar espelhos offline.

Além desses comandos, o usuário pode personalizar repositórios oficiais editando arquivos `.conf` em `/etc/xbps.d/` ou `/usr/share/xbps.d/`. O XBPS lida nativamente com múltiplas arquiteturas (**x86_64**, **ARM**, etc.) e suporta tanto o glibc quanto o musl como bibliotecas C. O código-fonte do XBPS está disponível no repositório oficial do GitHub, onde pode-se acompanhar seu desenvolvimento e contribuições da comunidade.

Rolling release?

O Void Linux adota o modelo *rolling release*[10] (ou lançamento contínuo). Isso significa que não existem versões pontuais pré-definidas do sistema[11] em vez disso, o usuário mantém o Void sempre atualizado com as últimas versões dos pacotes disponibilizados nos repositórios. Segundo a documentação oficial, o Void é "uma distribuição independente, de lançamento contínuo (*rolling release*), desenvolvida do zero com foco na estabilidade em vez do bleeding edge". Em outras palavras, apesar de receber atualizações frequentes, o Void prioriza a confiabilidade: as mudanças são testadas e distribuídas de forma a minimizar impactos, evitando pacotes de código-fonte instáveis.

No site oficial do Void, essa filosofia aparece sob o lema "Stable *rolling release*". Ou seja, o Void se concentra em ser estável em vez de sempre ter o software mais recente. A recomendação é "instale uma vez, atualize rotineiramente e com segurança" . Graças ao seu sistema de build contínuo, sempre que um desenvolvedor comita alterações no repositório "void-packages", novas versões binárias dos pacotes são imediatamente construídas e enviadas para os espelhos oficiais. Isso permite que o usuário receba atualizações atuais e seguras quase em tempo real, sem precisar reinstalar o sistema, mantendo-o moderno sem comprometer a estabilidade.

Em resumo, o modelo rolling release do Void garante um fluxo contínuo de atualizações de segurança e de recursos, diferentemente de distribuições baseadas em lançamentos pontuais. Para o usuário final, isso significa manter o sistema atualizado simplesmente executando xbps-install -Su periodicamente, sem necessidade de "migração de versão" no estilo de grandes lançamentos a cada ano.

xbps-src

O xbps-src é o sistema de compilação (build system) de pacotes-fonte do Void Linux. Integrado ao repositório void-packages (https://github.com/void-linux/void-packages), ele permite construir pacotes diretamente das fontes usando templates específicos. Inspirado em sistemas de ports de BSD, o xbps-src foi escrito do zero e é uma das principais forças do Void. De acordo com o Handbook, "você pode usar o xbps-src no repositório void-packages para construir pacotes (incluindo os restritos) a partir de templates". Em outras palavras, o Void oferece um único repositório de

[10] Termo em inglês que indica que o sistema recebe atualizações constantes sem lançamentos de versão numérica periódica.

[11] Exemplo, Ubuntu 22.04 LTS (MELHORAR FOOTNOTE)

templates (srcpkgs/) contendo os metadados necessários (origem, patches, dependências) para compilação de cada pacote; o xbps-src orquestra o processo de compilação com base nesses templates.

Todo o processo de build do xbps-src é feito em contêineres isolados: ele utiliza chroots baseados nos namespaces do Linux para montar um ambiente limpo para cada compilação, garantindo que nenhum binário do sistema host seja usado inadvertidamente. Por exemplo, ele cria um rootfs temporário baseado no glibc (ou no musl, conforme desejado) e executa a compilação lá. Dessa forma, não é necessário privilégio de root para compilar pacotes; o usuário comum consegue gerar o pacote binário completo em segurança. Além disso, o xbps-src suporta compilação cruzada: é possível compilar pacotes para arquiteturas diferentes da atual. Ao final do processo, o resultado é um pacote .xbps que pode ser instalado normalmente pelo xbps-install ou disponibilizado em um repositório local.

Para utilizar o xbps-src, normalmente seguem-se estes passos básicos:

1. Instalar ferramentas de desenvolvimento: por exemplo, executar sudo xbps-install git base-devel xtools para obter compilers, headers e utilitários necessários.
2. Clonar o repositório oficial de receitas: git clone https://github.com/void-linux/void-packages.git (ou navegar até ele, caso já esteja).
3. Navegar até a pasta void-packages e preparar o ambiente de compilação: executar ./xbps-src binary-bootstrap (essa etapa inicializa os diretórios de rootfs para glibc e musl e baixa dependências básicas).
4. Compilar o pacote desejado: ./xbps-src -N (-N para build nativo). O xbps-src baixa o código-fonte, aplica patches e gera o pacote binário. Opcionalmente pode-se usar ./xbps-src clean para remover compilados anteriores antes da build.

Gerenciamento de memória, processos e dispositivos

TO-DO.

Bibliografia

1 “Enter the Void”, https://voidlinux.org/, acesso em fevereiro de 2025

2 “runit - a UNIX init scheme with service supervision”, https://smarden.org/runit/, acesso em março de 2025

3 “Void Linux Handbook”, https://docs.voidlinux.org/, acesso em fevereiro de 2025

4 “The Void source packages collection”, https://github.com/void-linux/void-packages, acesso em abril de 2025

5 “The X Binary Package System (XBPS)”, https://github.com/void-linux/xbps, acesso em abril de 2025

6 “Distro Walk - Void Linux >> Linux Magazine”, https://www.linux-magazine.com/Issues/2021/249/Void-Linux, acesso em abril de 2025

7 “Distrowatch - Void”, https://distrowatch.com/table.php?distribution=void, acesso em fevereiro de 2025

8 “OpenSSL in Void Linux”, https://voidlinux.org/news/2021/02/OpenSSL.html, acesso em abril de 2025

9 “GitHub Organisation is moving”, https://voidlinux.org/news/2018/06/GitHub-Organisation-is-moving.html, acesso em abril de 2025

10 “Serious issues with Void Linux”, https://voidlinux.org/news/2018/05/serious-issues.html, acesso em abril de 2025

11 “glibc vs. musl - Chainguard Academy”, https://edu.chainguard.dev/chainguard/chainguard-images/about/images-compiled-programs/glibc-vs-musl/, acesso em abril de 2025