Kernele: lista, usuwanie starych, brak miejsca na /boot

◫◫ Poziom: średni

◫◫ Dotyczy: RHEL/Rocky 8–10 (dnf), różnice RHEL/CentOS 5–7 (package-cleanup) zaznaczone

▯ Czas: ~20 min

`dnf` domyślnie trzyma kilka ostatnich jąder na wypadek, gdyby najnowsze nie chciało wystartować. Ale każde jądro to kilkaset megabajtów na `/boot`, a `/boot` jest małą, osobną partycją — po kilku latach aktualizacji bez sprzątania pełne `/boot` potrafi zablokować kolejną aktualizację. Ta strona pokazuje, jak sprzątać bezpiecznie i jak rozpoznać, że już jest za późno na bezpieczne sprzątanie.

◫◫ Skróty użyte na tej stronie (9)

Skrót	Co znaczy
BIOS	Basic Input/Output System: oprogramowanie płyty głównej, które startuje komputer
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
GRUB	Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić
LVM	Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny
MBR	Master Boot Record: stary sposób dzielenia dysku na partycje, do 2 TB
RAID	Redundant Array of Independent Disks: sposób łączenia kilku fizycznych dysków w jeden, widziany przez system jako pojedynczy dysk, zwykle z dodatkową odpornością na awarię jednego z nich
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
RPM	RPM Package Manager: format paczek i narzędzie instalujące oprogramowanie w RHEL/Rocky, dnf korzysta z niego pod spodem
UEFI	Unified Extensible Firmware Interface: nowoczesne oprogramowanie firmowe płyty głównej albo maszyny wirtualnej, następca starszego BIOS-u, obsługuje między innymi Secure Boot

Fiszka

□□ TYLKO ODCZYT

Sprawdzenie, który kernel jest uruchomiony teraz: `uname -r`

```
uname -r
```

Pierwsza komenda, zawsze, zanim cokolwiek usuniemy. Pokazujemy wersję jądra, na którym system aktualnie działa — to jest ten jeden numer, którego nie wolno pomylić z żadnym innym przy sprzątaniu starych kerneli.

Przykład:

```
5.14.0-503.14.1.el9_5.x86_64
```

□□ TYLKO ODCZYT

Wyświetlenie listy zainstalowanych pakietów jądra: `rpm -q kernel`

```
rpm -q kernel
```

Wszystkie wersje jądra zainstalowane w systemie naraz — RHEL/Rocky celowo trzyma kilka jednocześnie, żeby dało się wystartować ze starszej, gdyby najnowsza się nie podniosła.

Przykład:

```
kernel-5.14.0-427.31.1.el9_4.x86_64
kernel-5.14.0-503.11.1.el9_5.x86_64
kernel-5.14.0-503.14.1.el9_5.x86_64
```

□□ **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **rpm** = RPM Package Manager: format i narzędzie paczek RHEL

□□ TYLKO ODCZYT

Zaznaczenie aktywnego jądra na liście: `rpm -q kernel | sed`

```
rpm -q kernel | sed "/$(uname -r)/ s/$/ AKTYWNY/"
```

> **Pojęcie: jądro systemu i obrazy rozruchowe.** Jądro (kernel) to rdzeń systemu operacyjnego — kod zarządzający procesorem, pamięcią, urządzeniami i wszystkim, na czym działa reszta systemu. Każda wersja jądra jest osobnym pakietem RPM, a każdy zainstalowany pakiet jądra zostawia w `/boot` swój obraz rozruchowy i wpis w menu GRUB, dzięki czemu przy starcie systemu można wybrać, którą wersję uruchomić. Ta komenda podstawia bieżącą wersję jądra (`uname -r`) do wyrażenia `sed` i dopisuje słowo `AKTYWNY` przy linii, która pasuje — pozwala jednym spojrzeniem zweryfikować, że to, co uruchomione, jest tym samym, co widnieje jako najnowsze na liście `rpm -q kernel`.

Przykład:

```
kernel-5.14.0-427.31.1.el9_4.x86_64
kernel-5.14.0-503.11.1.el9_5.x86_64
kernel-5.14.0-503.14.1.el9_5.x86_64 AKTYWNY
```

Jeśli `AKTYWNY` wyląduje nie przy najnowszej pozycji na liście, to znak, że system nie został jeszcze zrestartowany po ostatniej aktualizacji jądra — sprzątanie starych wersji poczeka do restartu.

RPM = RPM Package Manager: format paczek i narzędzie instalujące oprogramowanie w RHEL/Rocky, dnf korzysta z niego pod spodem · **GRUB** = Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić ·**rpm** = RPM Package Manager: format i narzędzie paczek RHEL

TYLKO ODCZYT

Sprawdzenie zainstalowanych jąder z perspektywy `dnf`: `dnf list installed kernel`

```
dnf list installed kernel
```

Odpowiednik `rpm -q kernel`, ale w formacie `dnf`, z widoczną kolumną repozytorium źródłowego.

Przykład:

```
Zainstalowane pakiety
kernel.x86_64      5.14.0-427.31.1.el9_4      @baseos
kernel.x86_64      5.14.0-503.11.1.el9_5      @updates
kernel.x86_64      5.14.0-503.14.1.el9_5      @updates
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Sprawdzenie, czy jest jeszcze miejsce: `df -h /boot`

```
df -h /boot
```

> **Pojęcie: installonly_limit.** `/boot` to zwykle mała, osobna partycja (500 MB – 1 GB), bo tradycyjnie musiała mieścić się w ograniczeniach starych bootloaderów. Każde jądro zajmuje tam kilkadziesiąt–kilkaset megabajtów (obraz jądra, `initramfs`, mapa symboli). `installonly_limit` w `/etc/dnf/dnf.conf` określa, ile wersji jądra `dnf` ma trzymać jednocześnie, zanim zacznie automatycznie usuwać najstarsze przy kolejnej aktualizacji — domyślnie 3.

Przykład:

```
Systemplik           Rozm. Uż. Dost. Uż% zamont. na
/dev/sda1              960M  892M   19M  98% /boot
```

98% zapełnienia przy zaledwie 19 MB wolnego miejsca to sygnał, że kolejna aktualizacja jądra (a każda potrzebuje kilkuset megabajtów wolnego miejsca na nowy obraz i `initramfs`) może się nie udać z błędem braku miejsca na dysku — dokładnie tak to wygląda, zanim ktoś zgłosi, że `dnf update` przestał działać.

GB = Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej · **df** = disk free: ile miejsca wolnego na dyskach

TYLKO ODCZYT

Komenda usunięcia nieużywanych kerneli w nowszych systemach operacyjnych EL 8+: `dnf remove --oldinstallonly`

```
dnf remove --oldinstallonly --setopt installonly_limit=2 kernel
```

Komenda dla nowszych systemów (RHEL/Rocky 8 i wyżej). Usuwamy nią wszystkie zainstalowane jądra ponad limit podany w `installonly_limit`, zostawiając wskazaną liczbę najnowszych — tutaj dwie, czyli bieżącą plus jedną zapasową. **Nigdy nie usuwamy jądra, na którym system aktualnie działa**, nawet jeśli formalnie wypadałoby ono poza limit — `dnf` sam wyklucza aktywne jądro z kandydatów do usunięcia.

Przykład:

```
Zależności rozwiązane.
================================================================================
 Pakiet         Architektura   Wersja                    Repozytorium
================================================================================
Usuwanie:
 kernel         x86_64         5.14.0-427.31.1.el9_4     @baseos
 kernel-core    x86_64         5.14.0-427.31.1.el9_4     @baseos

Podsumowanie transakcji
================================================================================
Usuń  2 pakiety
```

□□ **EL** = Enterprise Linux: rodzina RHEL i jej klonów (Rocky, Alma, Oracle Linux) · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky

□□ TYLKO ODCZYT

Komenda usunięcia nieużywanych kerneli w starszych systemach operacyjnych EL (wersje 5-7): `package-cleanup --oldkernels`

```
package-cleanup --oldkernels --count=1
```

Odpowiednik dla starszych systemów (RHEL/CentOS 5–7), zanim `dnf` dostał wbudowaną flagę `--oldinstallonly`. Narzędzie z pakietu `yum-utils`. `--count=1` oznacza „zostaw tylko jedno jądro" — czyli bieżące; `--count=2` zostawia bieżące plus jedno zapasowe, analogicznie do `installonly_limit=2` wyżej.

Przykład:

```
Nie usuwamy najnowszego jądra, ponieważ jest aktywne.
Usuwanie: kernel-3.10.0-1160.el7.x86_64
```

□□ **EL** = Enterprise Linux: rodzina RHEL i jej klonów (Rocky, Alma, Oracle Linux) · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

□□ TYLKO ODCZYT

Usunięcie jednej konkretnej wersji: `dnf remove kernel-<wersja>`

```
dnf remove kernel-5.14.0-427.31.1.el9_4
```

Zamiast polegać na limicie, wskazujemy dokładnie tę wersję, którą chcemy usunąć. Przydatne, gdy chcemy zachować kontrolę nad tym, co dokładnie znika, zamiast ufać automatycznemu wyliczeniu limitu — szczególnie gdy na liście są też jądra z poprawkami bezpieczeństwa, które wolimy zatrzymać mimo że są starsze niż bieżące.

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Ręczne usuwanie z `/boot`, gdy `dnf`/`package-cleanup` nie wystarczają

```
cd /boot
find . -name '*5.14.0-427.31.1.el9_4*'
```

Ostateczność, gdy baza RPM jest niespójna albo pliki w `/boot` zostały osierocone przez nieudaną wcześniejszą aktualizację. Najpierw wyszukujemy wszystkie pliki związane z konkretną wersją jądra po numerze w nazwie.

Przykład:

```
./vmlinuz-5.14.0-427.31.1.el9_4.x86_64
./initramfs-5.14.0-427.31.1.el9_4.x86_64.img
./config-5.14.0-427.31.1.el9_4.x86_64
./System.map-5.14.0-427.31.1.el9_4.x86_64
```

```
rm ./vmlinuz-5.14.0-427.31.1.el9_4.x86_64 \
   ./initramfs-5.14.0-427.31.1.el9_4.x86_64.img \
   ./config-5.14.0-427.31.1.el9_4.x86_64 \
   ./System.map-5.14.0-427.31.1.el9_4.x86_64
```

Po ręcznym usunięciu plików trzeba przebudować konfigurację GRUB-a — usunięcie plików z `/boot` nie aktualizuje samo z siebie menu rozruchowego, więc bez tego kroku GRUB nadal będzie próbował zaproponować (choć nie uruchomić — bootloader poradzi sobie z brakującym wpisem) wpis do jądra, którego już nie ma na dysku.

RPM = RPM Package Manager: format paczek i narzędzie instalujące oprogramowanie w RHEL/Rocky, dnf korzysta z niego pod spodem · **GRUB** = Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić · **cd** = change directory: przejdź do katalogu · **find** = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru · **rm** = remove: usuń (bez kosza, na zawsze)

TYLKO ODCZYT

Wyświetlenie listy wpisów w GRUB-ie: `grubby --info=ALL`

```
grubby --info=ALL
```

Pokazujemy wszystkie wpisy rozruchowe znane bootloaderowi, niezależnie od tego, czy odpowiadające im pliki jądra nadal istnieją na dysku. Dobre pierwsze sprawdzenie po ręcznym usuwaniu plików — czy GRUB nadal „pamięta" usunięte jądro.

Przykład:

```
index=0
kernel="/boot/vmlinuz-5.14.0-503.14.1.el9_5.x86_64"
args="ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M"
root="/dev/mapper/vgOS-lvOS"
initrd="/boot/initramfs-5.14.0-503.14.1.el9_5.x86_64.img"
title="Rocky Linux (5.14.0-503.14.1.el9_5.x86_64) 9.5 (Blue Onyx)"
```

GRUB = Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić

TYLKO ODCZYT

Sprawdzenie, który wpis wystartuje domyślnie: `grubby --default-kernel`

```
grubby --default-kernel
```

Pokazujemy ścieżkę do obrazu jądra, który GRUB uruchomi bez ingerencji użytkownika. Sprawdzamy to po sprzątaniu, żeby upewnić się, że domyślny wybór nadal wskazuje na istniejący plik.

Przykład:

```
/boot/vmlinuz-5.14.0-503.14.1.el9_5.x86_64
```

GRUB = Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić

ZMIENIA SYSTEM

Zmiana domyślnego jądra: `grubby --set-default`

```
grubby --set-default /boot/vmlinuz-5.14.0-503.14.1.el9_5.x86_64
```

Ustawiamy wskazany wpis jako domyślny przy starcie. Rzadko potrzebne przy zwykłym sprzątaniu (usunięcie starego jądra samo w sobie nie zmienia domyślnego wyboru, o ile nie usuwaliśmy akurat domyślnego), ale przydatne po ręcznej ingerencji w GRUB.

GRUB = Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić

TYLKO ODCZYT

Przebudowa obrazu `initramfs`: `dracut -f`

```
dracut -f /boot/initramfs-$(uname -r).img $(uname -r)
```

> **Pojęcie: initramfs.** Tymczasowy system plików ładowany do pamięci razem z jądrem przy starcie, zawierający sterowniki i narzędzia potrzebne, zanim jądro zdąży zamontować właściwy dysk systemowy (np. sterowniki RAID, LVM, szyfrowania). Jeśli `initramfs` dla bieżącego jądra jest uszkodzony albo brakuje go po nieudanej operacji na `/boot`, `dracut -f` odbudowuje go od nowa dla wskazanej wersji jądra. To komenda ratunkowa, nie rutynowa — normalna instalacja pakietu `kernel` generuje `initramfs` sama.

Przykład:

```
dracut: Executing: /usr/bin/dracut -f /boot/initramfs-5.14.0-503.14.1.el9_5.x86_64.img
5.14.0-503.14.1.el9_5.x86_64
```

RAID = Redundant Array of Independent Disks: sposób łączenia kilku fizycznych dysków w jeden, widziany przez system jako pojedynczy dysk, zwykle z dodatkową odpornością na awarię jednego z nich · **LVM** = Logical Volume Manager: warstwa w Linuksie składająca dyski w elastyczne wolumeny

TYLKO ODCZYT

Przebudowa pliku konfiguracyjnego GRUB-a: `grub2-mkconfig -o`

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

Skanujemy ponownie zainstalowane jądra i regenerujemy menu rozruchowe GRUB-a od zera na podstawie tego, co faktycznie istnieje w `/boot` — usuwamy wpisy do jąder, których pliki zniknęły, i dodajemy wpisy do nowych. To jest krok, który wykonujemy **zawsze po ręcznym usuwaniu plików jądra z** `/boot`, żeby menu rozruchowe zgadzało się ze stanem dysku. Dla systemów z UEFI (zamiast klasycznego BIOS/MBR) ścieżka wyjściowa jest inna:

```
grub2-mkconfig -o /boot/efi/EFI/rocky/grub.cfg
```

Wykonujemy **tylko jedną** z tych dwóch komend, zależnie od tego, czy maszyna startuje w trybie BIOS (MBR) czy UEFI — nie obie naraz.

Przykład:

```
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.14.0-503.14.1.el9_5.x86_64
Found initrd image: /boot/initramfs-5.14.0-503.14.1.el9_5.x86_64.img
done
```

UEFI = Unified Extensible Firmware Interface: nowoczesne oprogramowanie firmowe płyty głównej albo maszyny wirtualnej, następca starszego BIOS-u, obsługuje między innymi Secure Boot · **BIOS** = Basic Input/Output System: oprogramowanie płyty głównej, które startuje komputer · **MBR** = Master Boot Record: stary sposób dzielenia dysku na partycje, do 2 TB

TYLKO ODCZYT

`installonly_limit` w `/etc/dnf/dnf.conf`

```
grep installonly_limit /etc/dnf/dnf.conf
```

Trwałe ustawienie liczby jąder trzymanych automatycznie przy każdej przyszłej aktualizacji — zmieniamy je raz, zamiast za każdym razem dopisywać `--setopt` do `dnf remove`.

Przykład:

```
installonly_limit=3
```

grep = global regular expression print: szukaj wzorca w tekście

TYLKO ODCZYT

Wszystkie flagi i komendy

Komenda	Co robi
`uname -r`	wersja jądra uruchomionego teraz — sprawdzasz zawsze pierwsza
`rpm -q kernel`	lista zainstalowanych wersji jądra
`dnf list installed kernel`	to samo z perspektywy `dnf`, z repozytorium źródłowym
`df -h /boot`	zapełnienie partycji `/boot`
`dnf remove --oldinstallonly --setopt installonly_limit=N kernel`	usuwa stare jądra, RHEL/Rocky 8+
`package-cleanup --oldkernels --count=N`	to samo, starsze RHEL/CentOS 5–7
`dnf remove kernel-<wersja>`	usuwa jedną, wskazaną wersję
`cd /boot && find . -name '*wersja*'`	ręczne wyszukanie plików jądra
`grubby --info=ALL`	lista wpisów rozruchowych GRUB-a
`grubby --default-kernel` / `--set-default`	odczyt / zmiana domyślnego jądra
`dracut -f`	przebudowa `initramfs`
`grub2-mkconfig -o ścieżka`	przebudowa konfiguracji GRUB-a po zmianach w `/boot`
`installonly_limit` (`/etc/dnf/dnf.conf`)	trwały limit trzymanych jąder

Uważaj

- **Nigdy nie usuwaj jądra zwróconego przez `uname -r`** — jeśli je usuniesz, a system się zrestartuje zanim zainstalujesz cokolwiek nowego, może nie mieć czym wystartować.
- Ręczne `rm` w `/boot` bez późniejszego `grub2-mkconfig` zostawia GRUB z martwymi wpisami — nieszkodliwe od razu, ale mylące przy następnej awarii.
- Pełne `/boot` blokuje **każdą** kolejną aktualizację jądra, nie tylko tę jedną, którą akurat próbujesz zainstalować — dopóki nie zwolnisz miejsca, `dnf update` będzie się wywalać za każdym razem w tym samym miejscu.

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **GRUB** = Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić

Z życia wzięte

Z ŻYCIA WZIĘTE

Wtorek po Patch Tuesday. Aktualizacja jądra na ci01 przerywa się w połowie komunikatem o braku miejsca na /boot. Tomek sprawdza uname -r, zostawia dwie najnowsze wersje jądra zamiast pięciu zalegających, i dopiero wtedy aktualizacja przechodzi bez błędu.

Po co to istnieje

Kiedy instalujesz nową wersję jądra, RHEL/Rocky **nie podmienia** starej wersji — instaluje nową obok, zostawiając starą jako pozycję rezerwową w menu GRUB-a. To celowe: jeśli nowe jądro nie wystartuje (niekompatybilny sterownik, uszkodzony `initramfs`, regresja w samym jądrze), administrator wybiera z menu rozruchowego poprzednią, sprawdzoną wersję i system wraca do życia bez żadnej dodatkowej pracy. Ta strategia ma jednak koszt: każda zatrzymana wersja jądra to komplet plików w `/boot` — obraz jądra (`vmlinuz`), `initramfs`, mapa symboli, plik konfiguracji kompilacji — łącznie zwykle 100–300 MB na wersję. Partycja `/boot` jest tradycyjnie mała, więc po kilkudziesięciu aktualizacjach bez sprzątania po prostu się zapełnia.

`installonly_limit` w `dnf.conf` to wbudowany mechanizm, który ma nie dopuścić do tego scenariusza automatycznie — domyślnie trzyma trzy najnowsze jądra i przy każdej kolejnej instalacji jądra usuwa najstarsze ponad ten limit. Problem pojawia się, gdy limit został kiedyś podniesiony ręcznie, gdy system był migrowany bez zachowania tego ustawienia, albo gdy ktoś ręcznie instalował jądra bez przechodzenia przez zwykły `dnf update` — wtedy liczba zalegających wersji rośnie bez kontroli, aż `/boot` się zapełni.

Analogia

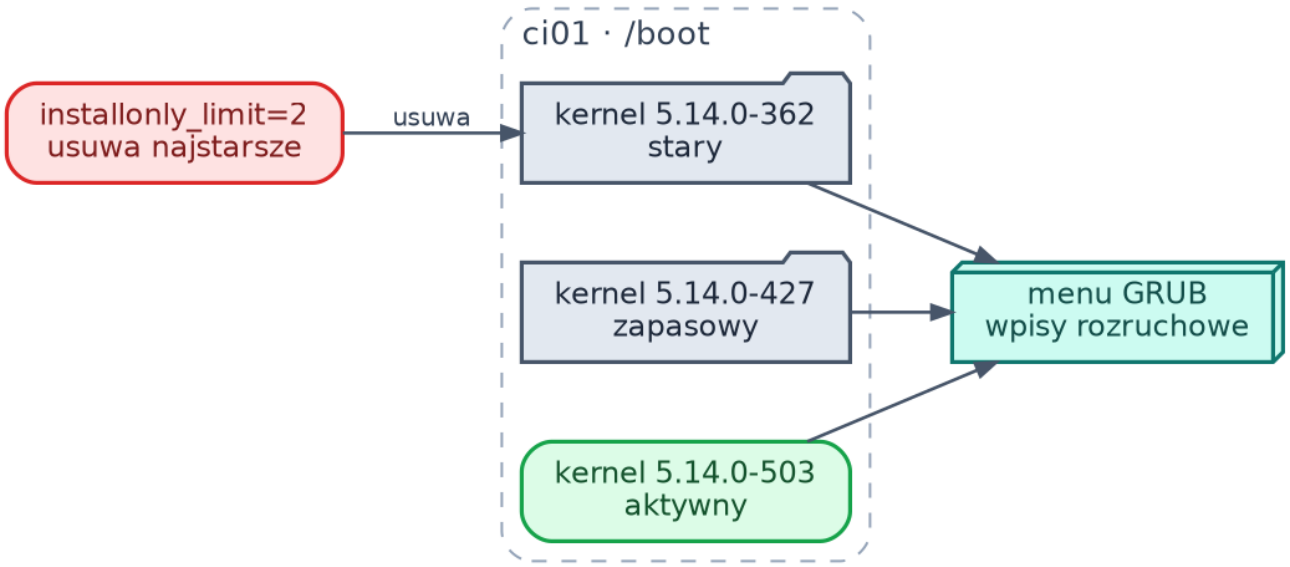
`/boot` to **mała szafka na zapasowe koło w bagażniku samochodu**.

Trzymasz tam zapasowe koło (poprzednie jądro) na wypadek, gdyby to zamontowane właśnie teraz (bieżące jądro) się rozerwało w trasie. Rozsądnie jest mieć jedno zapasowe koło — może nawet dwa, gdybyś jeździł w trudnym terenie. Ale szafka bagażnika ma skończony rozmiar. Jeśli po każdej wymianie opony (każdej aktualizacji jądra) wrzucasz starą oponę do tej samej szafki i nigdy jej nie opróżniasz, w końcu nie zmieści się tam nawet nowe zapasowe koło — a to jest dokładnie moment, w którym kolejna „wymiana opony" (`dnf update`) odmawia wykonania z powodu braku miejsca.

Kluczowa zasada bezpieczeństwa tej analogii: nigdy nie wyrzucasz koła, które akurat jest zamontowane na aucie i którym właśnie jedziesz — usuwasz wyłącznie te leżące w szafce jako zapas. `uname -r` to sprawdzenie, które koło jest teraz zamontowane, zanim zaczniesz cokolwiek wyrzucać.

Jak to działa

Każdy pakiet `kernel` jest w RPM oznaczony jako pakiet typu `installonly` — kategoria pakietów, których `dnf` nigdy nie aktualizuje przez podmianę w miejscu, tylko instaluje jako zupełnie nową, równoległą wersję. Inne przykłady tej kategorii to `kernel-core`, `kernel-modules` i sterowniki jądra dostarczane osobno. Standardowe pakiety (np. `httpd`) przy aktualizacji podmieniają starą wersję na nową; pakiety `installonly` zamiast tego się kumulują, dopóki coś (limit albo ręczna interwencja) ich nie posprząta.



Kolejne wersje jądra w /boot trafiają do menu GRUB, a installonly_limit usuwa najstarsze z nich. (kliknięcie otwiera pełny rozmiar)

`installonly_limit` w `/etc/dnf/dnf.conf` określa, ile wersji takiego pakietu `dnf` ma tolerować naraz. Gdy limit zostaje przekroczony przy instalacji kolejnej wersji, `dnf` automatycznie kwalifikuje najstarsze wersje ponad limit do usunięcia — ale **zawsze z wyłączeniem wersji aktualnie uruchomionej**, niezależnie od tego, jak stara by ona formalnie była względem limitu. To zabezpieczenie działa na poziomie samego `dnf`, nie tylko jako dobra praktyka administratora.

Menu rozruchowe GRUB-a (`/boot/grub2/grub.cfg` na BIOS, `/boot/efi/EFI/<dystrybucja>/grub.cfg` na UEFI) jest plikiem **generowanym**, nie edytowanym ręcznie na co dzień. Standardowa instalacja/usunięcie pakietu `kernel` przez `dnf` samo aktualizuje ten plik za pomocą skryptów poinstalacyjnych pakietu. Ręczne usunięcie plików jądra z `/boot` przez `rm` omija te skrypty całkowicie — GRUB nie dowiaduje się o zmianie, dopóki ktoś ręcznie nie uruchomi `grub2-mkconfig`, żeby przeskanować `/boot` od nowa i przepisać menu na podstawie tego, co tam faktycznie zostało.

Krok po kroku

1. Ustal, które jądro jest aktywne — zawsze pierwszy krok.

```
uname -r
```

```
5.14.0-503.14.1.el9_5.x86_64
```

2. Zobacz pełną listę i potwierdź, że aktywne jest najnowsze.

```
rpm -q kernel | sed "/$(uname -r)/ s/$/ AKTYWNY/"
```

```
kernel-5.14.0-427.31.1.el9_4.x86_64
kernel-5.14.0-503.11.1.el9_5.x86_64
kernel-5.14.0-503.14.1.el9_5.x86_64 AKTYWNY
```

Jeśli `AKTYWNY` nie jest przy ostatniej pozycji, system czeka na restart po aktualizacji — nie sprzątaj jeszcze.

3. Sprawdź zapełnienie `/boot` przed sprzątaniem, żeby mieć punkt odniesienia.

```
df -h /boot
```

4. Usuń stare jądra automatycznym mechanizmem, zostawiając dwie wersje.

```
dnf remove --oldinstallonly --setopt installonly_limit=2 kernel
```

5. Zweryfikuj, że kernele rzeczywiście zniknęły i że aktywne zostało.

```
rpm -q kernel
uname -r
```

```
kernel-5.14.0-503.11.1.el9_5.x86_64
kernel-5.14.0-503.14.1.el9_5.x86_64
5.14.0-503.14.1.el9_5.x86_64
```

6. Sprawdź, ile miejsca odzyskałeś.

```
df -h /boot
```

Częste błędy

`/boot` zapełnione do 100% blokuje kolejną aktualizację jądra.

```
Błąd:
 Uruchomienie testu transakcji
 Błąd testu transakcji:
  Brak miejsca na dysku w filesystemie /boot: potrzeba 187 MB, dostępne 4 MB
```

`dnf` sprawdza dostępne miejsce przed instalacją i odmawia kontynuowania, żeby nie zostawić systemu z niekompletnym obrazem jądra. Naprawa: `dnf remove --oldinstallonly --setopt installonly_limit=1 kernel`, żeby chwilowo zejść do samego minimum (tylko aktywne jądro), zwolnić miejsce i dopiero wtedy przeprowadzić aktualizację — po niej z powrotem podnieść `installonly_limit` do rozsądnej wartości (2–3).

Ręczne `rm` w `/boot` bez przebudowy GRUB-a.

```
cd /boot && rm vmlinuz-5.14.0-427.31.1.el9_4.x86_64 initramfs-5.14.0-427.31.1.el9_4.x86_64.img
grubby --info=ALL | grep 427.31.1
```

```
kernel="/boot/vmlinuz-5.14.0-427.31.1.el9_4.x86_64"
```

GRUB dalej „widzi" wpis do jądra, którego plików fizycznie już nie ma na dysku — bo `rm` nie uruchamia żadnych skryptów aktualizujących konfigurację bootloadera. Nowoczesny GRUB2 sam pominie martwy wpis przy starcie zamiast się zawiesić, ale menu rozruchowe pokazuje mylącą opcję, a diagnoza kolejnej awarii jest trudniejsza. Naprawa: `grub2-mkconfig -o /boot/grub2/grub.cfg` (BIOS) albo `grub2-mkconfig -o /boot/efi/EFI/rocky/grub.cfg` (UEFI) od razu po ręcznym usuwaniu plików.

Próba usunięcia aktywnego jądra ręcznym `dnf remove kernel-<wersja>`.

```
uname -r
dnf remove kernel-5.14.0-503.14.1.el9_5
```

```
5.14.0-503.14.1.el9_5.x86_64
Błąd:
 Nie można usunąć kernel-5.14.0-503.14.1.el9_5.x86_64: aktualnie działający kernel
```

W przeciwieństwie do `--oldinstallonly` (który sam wyklucza aktywne jądro z automatycznego wyboru), ręczne wskazanie konkretnej wersji do `dnf remove` nadal jest chronione — `dnf` odmawia usunięcia jądra, na którym system aktualnie działa. To jest zabezpieczenie wbudowane w `dnf`, nie coś, na czym trzeba polegać wyłącznie własną uwagą — ale nie zwalnia to z nawyku sprawdzania

`uname -r` jako pierwszego kroku, bo nie każde narzędzie (patrz niżej) ma tę samą ochronę.

Ręczne `rm` po nazwie pliku bez wcześniejszego sprawdzenia `uname -r`.

```
cd /boot
rm vmlinuz-5.14.0-503.14.1.el9_5.x86_64
```

W odróżnieniu od `dnf remove`, zwykłe `rm` **nie ma żadnej ochrony** przed usunięciem pliku aktywnego jądra — system działa dalej normalnie (jądro jest już załadowane do pamięci), ale przy najbliższym restarcie okaże się, że plik, z którego GRUB miał wystartować, zniknął. Naprawa, jeśli to się już stało: nie restartuj maszyny, zainstaluj ponownie pakiet tej samej wersji jądra przez `dnf reinstall kernel-core-5.14.0-503.14.1.el9_5` (jeśli pakiet jest jeszcze w cache lub repozytorium) albo zainstaluj nowszą wersję jądra i dopiero wtedy bezpiecznie zrestartuj. Zapobieganie: zawsze `uname -r` przed jakimkolwiek `rm` w `/boot`, nigdy na odwrót.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Żadne ćwiczenie nie usuwa ani nie modyfikuje prawdziwego jądra ani plików w `/boot` — to jest strefa, w której eksperymentowanie na produkcji jest zbyt ryzykowne, więc poniżej tylko odczyt i symulacja.

Ćwiczenie 1 — bezpieczne rozpoznanie stanu bez żadnej zmiany

```
cd /tmp
uname -r > aktywny.txt
rpm -q kernel > wszystkie.txt
sed "/$(cat aktywny.txt)/ s/$/ AKTYWNY/" wszystkie.txt
df -h /boot
```

Ile masz zainstalowanych wersji jądra i ile procent `/boot` jest zajęte? Czy jesteś w strefie bezpiecznej (poniżej ~80%) czy w strefie ostrzegawczej?

Odpowiedź

Liczba wersji i procent zapełnienia zależą od konkretnej maszyny. Zasada niezależna od liczb: jeśli `/boot` przekracza 80–85% zapełnienia, warto zaplanować sprzątanie starych jąder *zanim*

trzeba będzie robić to w pośpiechu w momencie, gdy kolejna aktualizacja odmówi działania z braku miejsca.

Ćwiczenie 2 — symulacja wyszukiwania plików konkretnej wersji

```
cd /tmp
mkdir -p boot-symulacja
cd boot-symulacja
touch vmlinuz-5.14.0-427.31.1.el9_4.x86_64
touch initramfs-5.14.0-427.31.1.el9_4.x86_64.img
touch vmlinuz-5.14.0-503.14.1.el9_5.x86_64
find . -name '*5.14.0-427.31.1.el9_4*'
```

Napisz, jak wyglądałaby pełna procedura ręcznego usunięcia tych dwóch plików i jaki musiałby być kolejny krok po `rm`, gdyby to był prawdziwy `/boot`.

Odpowiedź

```
find . -name '*5.14.0-427.31.1.el9_4*' -delete
```

(albo `rm` na wypisanych plikach po ręcznej weryfikacji, że lista nie zawiera aktywnego jądra). Kolejny, obowiązkowy krok na prawdziwym `/boot`: `grub2-mkconfig -o /boot/grub2/grub.cfg` (albo ścieżka UEFI), żeby GRUB przestał odwoływać się do usuniętych plików.

Pytania kontrolne

1. Dlaczego `uname -r` jest zawsze pierwszą komendą w tej procedurze, a nie na przykład `rpm -q kernel`?

Odpowiedź

Bo `rpm -q kernel` pokazuje tylko listę zainstalowanych wersji, bez wskazania, która z nich jest aktualnie uruchomiona. `uname -r` daje dokładnie ten jeden numer, którego nie wolno usunąć — bez niego lista z `rpm -q` jest ślepa: nie wiadomo, czy najnowsza pozycja na liście to naprawdę ta, na której system stoi (mógł być zainstalowany nowy kernel bez restartu).

2. Czym różni się `dnf remove --oldinstallonly` od ręcznego `rm` plików w `/boot` pod względem bezpieczeństwa?

Odpowiedź

`dnf remove --oldinstallonly` działa przez menedżer pakietów: aktualizuje bazę RPM, usuwa wpis z GRUB-a i nigdy nie usunie aktywnego jądra — te zabezpieczenia są wbudowane. Ręczne `rm` usuwa tylko pliki z dysku, nic nie wie o bazie RPM ani o GRUB-ie, nie ma żadnej ochrony przed usunięciem aktywnego jądra, i wymaga ręcznego `grub2-mkconfig` po fakcie, żeby bootloader zgadzał się ze stanem `/boot`.

3. Jak rozpoznasz, że pełne `/boot` jest przyczyną nieudanej aktualizacji, a nie coś innego?

Odpowiedź

Komunikat błędu `dnf` w takim przypadku wprost mówi o braku miejsca na konkretnym systemie plików (`/boot`), zwykle na etapie testu transakcji, jeszcze przed pobraniem pakietów. Potwierdzenie: `df -h /boot` pokaże zapełnienie bliskie 100% z ilością wolnego miejsca mniejszą niż potrzebny rozmiar nowego jądra (zwykle 150–250 MB).

4. Po ręcznym usunięciu plików starego jądra z `/boot` zapomniałeś uruchomić `grub2-mkconfig`. Co dokładnie się popsuje?

Odpowiedź

Nic natychmiast — system działa dalej na aktywnym jądrze, które nie zostało ruszone. Problem jest odłożony: menu rozruchowe GRUB-a nadal zawiera wpis do usuniętej wersji jądra, który wskazuje na nieistniejące pliki. Nowoczesny GRUB2 zwykle pominie taki martwy wpis przy wyborze domyślnego, ale w razie awarii aktywnego jądra i próby ręcznego wyboru starszej wersji z menu, wybór tego konkretnego wpisu zakończy się błędem braku pliku zamiast uruchomieniem zapasowego jądra.

Powiązane: dnf: instalacja, aktualizacja, historia, wycofywanie zmian · Repozytoria: konfiguracja, EPEL, priorytety · Diagnostyka konfliktów i dzielenie aktualizacji na części

Źródła: `man 8 dnf` (sekcja o pakietach `installonly`) · `man 8 grubby` · `man 8 grub2-mkconfig` · `man 8 dracut` · materiały wewnętrzne — procedura sprzątania kerneli i standardowej aktualizacji Linux

Revision #15
Created 2026-09-18 06:56:55 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:32 UTC by Tomasz Konieczny