Diagnostyka konfliktów i dzielenie aktualizacji na części

Poziom: średni **Dotyczy: RHEL/Rocky 9–10** **Czas: ~18 min**

Duża aktualizacja, która wysypuje się w połowie, zostawia system w stanie trudnym do zdiagnozowania — dziesiątki pakietów na raz, jeden z nich winny, a komunikat błędu rzadko wskazuje który. Ta strona pokazuje, jak dzielić aktualizację na kawałki i jak metodycznie namierzyć pakiet, który blokuje całą transakcję.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

ZMIENIA SYSTEM

Wyświetlenie listy tego, co czeka na aktualizację: `dnf check-update`

```
dnf check-update
```

Pierwszy krok przed jakimkolwiek dzieleniem: sprawdzamy, ile w ogóle jest do zaktualizowania i jak duży to zakres, zanim zdecydujemy, czy dzielenie w ogóle ma sens.

Przykład:

```
bash.x86_64                 5.1.8-9.el9              baseos
glibc.x86_64                2.34-100.el9_4.7         baseos
```

```
httpd.x86_64                2.4.57-8.el9_4              appstream
kernel.x86_64               5.14.0-503.14.1.el9_5       baseos
openssl-libs.x86_64         1:3.0.7-27.el9_4            baseos
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Wypisanie samej listy nazw pakietów: `dnf check-update | awk '{print $1}'`

```
dnf check-update | awk '{print $1}'
```

Zdejmujemy kolumny z wersją i repozytorium, zostawiając czyste nazwy pakietów — surowiec do dalszego filtrowania, np. do policzenia, ile pakietów naprawdę czeka na aktualizację, albo do przekazania dalej do innego polecenia.

Przykład:

```
bash.x86_64
glibc.x86_64
httpd.x86_64
kernel.x86_64
openssl-libs.x86_64
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Aktualizacja zakresu alfabetycznego — podział na litery startowe

```
dnf update $(dnf list updates | awk '/^[a-c]/ {print $1}')
```

> **Pojęcie: metoda połowienia przy szukaniu winowajcy.** Zamiast aktualizować wszystko naraz, dzielimy listę pakietów na mniejsze grupy według pierwszej litery nazwy i aktualizujemy grupę po grupie. `awk '/^[a-c]/'` wybiera z wyniku `dnf list updates` tylko linie zaczynające się na `a`, `b` lub `c`, a `$(...)` podstawia wynikowe nazwy pakietów jako argumenty do `dnf update`. Metoda przydatna, gdy duża aktualizacja nie mieści się w oknie serwisowym albo gdy chcemy ograniczyć zasięg ewentualnej awarii do garstki pakietów zamiast całego systemu naraz.

Przykład:

```
Podsumowanie transakcji
================================================================================
Aktualizuj  8 pakietów

Rozmiar pobierania: 34 M
Czy jest to poprawne? [y/N]:
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Podział aktualizacji na pół — klasyczna metoda połowienia

```
dnf update $(dnf list updates | awk '/^[a-j]/ {print $1}')
dnf update $(dnf list updates | awk '/^[k-z]/ {print $1}')
```

Dwa wywołania zamiast jednego wielkiego: pierwsze obejmuje pakiety zaczynające się na litery a–j, drugie na k–z. Jeśli po pierwszej połowie coś się psuje, wiemy, że winowajca jest wśród liter a–j, i możemy połowić dalej wewnątrz tego zakresu (np. a–e i f–j), zamiast przeglądać dziesiątki pakietów na oślep.

Przykład:

```
Podsumowanie transakcji
================================================================================
Aktualizuj  23 pakiety

Rozmiar pobierania: 210 M
Czy jest to poprawne? [y/N]:
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Pominięcie jednego pakietu w transakcji: `dnf update --exclude=nazwa`

```
dnf update --exclude=kernel
```

Wykluczamy wskazany pakiet (i pasujące mu wzorce z gwiazdką, np. `--exclude=kernel*`) z bieżącej aktualizacji, zostawiając resztę nietkniętą. Typowe zastosowanie: chcemy zaktualizować wszystko poza jądrem, bo restart po nowym jądrze wymaga osobnego okna serwisowego.

Przykład:

```
Podsumowanie transakcji
================================================================================
Aktualizuj  46 pakietów (pominięto: kernel, kernel-core)
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Trwałe wykluczenie pakietu: wpis `exclude=` w `dnf.conf`

```
echo 'exclude=kernel*' >> /etc/dnf/dnf.conf
```

Zamiast dopisywać `--exclude` przy każdym wywołaniu, wpisujemy wykluczenie raz, na stałe — pakiet zostaje pominięty przez **każde** przyszłe `dnf update` i `dnf install`, dopóki wpisu nie usuniemy. Dobre dla pakietu, którego wersja jest świadomie zamrożona na dłużej (np. z powodu certyfikacji aplikacji), gorsze dla wykluczenia tymczasowego — łatwo o nim zapomnieć.

ZMIENIA SYSTEM

Pominięcie pakietów, które psują transakcję: `dnf update --skip-broken`

```
dnf update --skip-broken
```

Zamiast przerwać całą transakcję z powodu jednego pakietu z konfliktem zależności, `dnf` pomija problematyczne pakiety i aktualizuje resztę. Dobre do przepchnięcia dużej aktualizacji mimo pojedynczego konfliktu, złe jako nawyk — pomija problem zamiast go zdiagnozować, a pominięty pakiet zostaje na starej, potencjalnie podatnej wersji bez wyraźnego ostrzeżenia poza logiem transakcji.

Przykład:

```
Pakiety pominięte z powodu zależności:
    openssl-libs-1:3.0.7-27.el9_4.x86_64 z instalacji

Podsumowanie transakcji
================================================================================
Aktualizuj  45 pakietów
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Agresywne rozwiązanie konfliktu: `dnf update --best --allowerasing`

```
dnf update --best --allowerasing
```

Ostrzeżenie: `--best` wymusza próbę instalacji najnowszej dostępnej wersji każdego pakietu (zamiast zaakceptować starszą, jeśli najnowsza koliduje), a `--allowerasing` pozwala `dnf` **usunąć** pakiety, które stoją na drodze do rozwiązania konfliktu zależności, byle transakcja przeszła. To bywa jedynym sposobem na przepchnięcie trudnej aktualizacji (klasyczny przypadek: `iptables-ebtables` kontra `iptables-nft` przy migracji między wersjami), ale oznacza, że `dnf` może usunąć pakiet, którego wcale nie planowaliśmy usuwać — zawsze czytamy plan transakcji przed potwierdzeniem, nigdy nie łączymy tego z `-y` w ciemno.

Przykład:

```
Błąd testu transakcji:
  plik /usr/share/man/man8/ebtables-nft.8.gz z instalacji iptables-nft-1.8.10-4.el9_4
  jest w konflikcie z plikiem z pakietu iptables-ebtables-1.8.5-11.el8

Usuwanie:
 iptables-ebtables   x86_64   1.8.5-11.el8   @baseos

Podsumowanie transakcji
================================================================================
Aktualizuj  44 pakiety
Usuń         1 pakiet
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Sprawdzenie pakietów z niespełnionymi zależnościami: `dnf repoquery --unsatisfied`

```
dnf repoquery --unsatisfied
```

Wypisuje pakiety zainstalowane w systemie, których deklarowane zależności nie są obecnie spełnione — zwykle efekt przerwanej w połowie transakcji albo ręcznego usunięcia czegoś, od czego inny pakiet zależał.

Przykład:

```
package: nginx-1:1.24.0-1.el9.x86_64 from @System
  unresolved deps:
    libmodsecurity.so.3()(64bit)
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Pełna weryfikacja wszystkich zainstalowanych pakietów: `rpm -Va`

```
rpm -Va | head -20
```

Rozszerzona wersja `rpm -V` na cały system naraz: porównuje każdy zainstalowany plik z jego oczekiwanym stanem (rozmiar, suma kontrolna, uprawnienia). Wolne (potrafi trwać kilka minut), ale kompletne — dobre po nieudanej aktualizacji, żeby sprawdzić, czy coś zostało uszkodzone w połowie, nie tylko w pakietach objętych ostatnią transakcją.

Przykład:

```
S.5....T.  c /etc/httpd/conf/httpd.conf
missing     /usr/lib64/libmodsecurity.so.3
```

rpm = RPM Package Manager: format i narzędzie paczek RHEL

TYLKO ODCZYT

Sprawdzenie, co zrobiła ostatnia transakcja: `dnf history info last`

```
dnf history info last
```

Zamiast szukać numeru transakcji ręcznie, `last` zawsze wskazuje na najnowszą — szybki sposób sprawdzenia, co dokładnie zrobiła aktualizacja, którą właśnie wykonaliśmy, zanim spróbujemy ją cofnąć albo kontynuować dzielenie.

Przykład:

```
ID transakcji : 41
Zawartość     :
    Aktualizacja bash-5.1.8-9.el9 -> bash-5.1.8-9.el9_5
    Aktualizacja glibc-2.34-100.el9_4.6 -> glibc-2.34-100.el9_4.7
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Powrót jednego pakietu do starszej wersji: `dnf downgrade pakiet`

```
dnf downgrade NazwaPakietu
```

Gdy metoda połowienia już wskazała winowajcę, to jest najszybszy sposób na wycofanie się z problemu bez cofania całej wcześniej udanej części aktualizacji (opis pełny na stronie o `dnf`).

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Zamrożenie pakietu na obecnej wersji: `dnf versionlock add`

```
dnf versionlock add NazwaPakietu
```

Po znalezieniu i wycofaniu winowajcy blokujemy go na sprawdzonej wersji, żeby kolejne, niepodzielone `dnf update` nie podniosło go z powrotem do tej samej, wadliwej wersji, zanim problem zostanie wyjaśniony albo poprawiony przez dostawcę (pełny opis na stronie o `dnf`).

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Sprawdzenie, co jest obecnie zablokowane: `dnf versionlock list`

```
dnf versionlock list
```

Przykład:

```
httpd-0:2.4.57-5.el9.*
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Zdjęcie blokady: `dnf versionlock delete`

```
dnf versionlock delete NazwaPakietu
```

Gdy poprawka od dostawcy jest już dostępna i przetestowana, zdejmujemy blokadę, żeby pakiet znów mógł się aktualizować normalnie.

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Sprawdzenie pakietów z niespójnymi zależnościami: `package-cleanup --problems`

```
package-cleanup --problems
```

Narzędzie z `yum-utils`, starsze niż `dnf repoquery --unsatisfied`, ale wciąż spotykane w środowiskach z przyzwyczajeniami z ery `yum`. Wypisuje pakiety, których zależności są obecnie niespójne — po nieudanej aktualizacji to jedno z pierwszych narzędzi diagnostycznych.

Przykład:

```
Package nginx.x86_64 1:1.24.0-1.el9 has missing requires of libmodsecurity.so.3()(64bit)
```

ZMIENIA SYSTEM

Suchy przebieg całej aktualizacji: `dnf --assumeno update`

```
dnf --assumeno update
```

Pokazujemy pełny plan transakcji; polecenie samo odpowiada „nie" na pytanie o potwierdzenie — bezpieczny sposób, żeby zobaczyć rozmiar i zakres aktualizacji przed

zdecydowaniem, czy w ogóle trzeba ją dzielić na części, bez ryzyka przypadkowego wykonania.

Przykład:

```
Podsumowanie transakcji
================================================================================
Aktualizuj  91 pakietów

Rozmiar pobierania: 480 M
Operacja została anulowana.
```

dnf = Dandified YUM: menedżer pakietów RHEL/Rocky

TYLKO ODCZYT

Wszystkie flagi i komendy

Komenda	Co robi
`dnf check-update`	lista dostępnych aktualizacji, nic nie zmienia
`dnf update 'a*' 'b*' 'c*'`	aktualizacja tylko pakietów pasujących do wzorca nazwy
`dnf update --exclude=nazwa`	pomija jeden pakiet w tej transakcji
`exclude=` w `dnf.conf`	trwałe wykluczenie pakietu
`dnf update --skip-broken`	pomija pakiety powodujące konflikt
`dnf update --best --allowerasing`	wymusza rozwiązanie konfliktu, może usuwać pakiety
`dnf repoquery --unsatisfied`	pakiety z niespełnionymi zależnościami
`rpm -Va`	pełna weryfikacja integralności całego systemu
`dnf history info last`	szczegóły ostatniej transakcji
`dnf downgrade pakiet`	powrót jednego pakietu do starszej wersji
`dnf versionlock add/list/delete`	blokada wersji pakietu
`package-cleanup --problems`	pakiety z niespójnymi zależnościami (starsze narzędzie)
`dnf --assumeno update`	suchy przebieg, plan bez wykonania

Uważaj

- `--allowerasing` może usunąć pakiet, którego nie planowałeś ruszać — zawsze czytaj plan transakcji, nigdy nie łącz z `-y` bez wcześniejszego suchego przebiegu.
- `--skip-broken` maskuje problem zamiast go rozwiązać — pominięty pakiet zostaje na starej wersji cicho, bez alarmu poza logiem.
- Podział alfabetyczny nie gwarantuje, że każda część zmieści się w oknie serwisowym — duży pakiet (np. `kernel` z zależnościami) może sam w sobie zająć większość czasu jednej „połowy".

Z życia wzięte

Z ŻYCIA WZIĘTE

Sobota, okno serwisowe. Duża aktualizacja na k3s01 wysypuje się w połowie, a nikt nie wie, który z dziewięćdziesięciu pakietów jest winny. Tomek dzieli listę aktualizacji na pół, potem znowu na pół, aż po czterech próbach zostaje z jednym konkretnym pakietem, którego wcześniej nikt by nie podejrzewał.

Po co to istnieje

Aktualizacja stu kilkudziesięciu pakietów naraz ma trzy realne problemy w środowisku produkcyjnym. Po pierwsze — **czas i okno serwisowe**: pobranie kilkuset megabajtów i instalacja stu pakietów może nie zmieścić się w przydzielonym oknie, szczególnie przy wolnym łączu do repozytorium. Po drugie — **zasięg awarii**: jeśli coś pójdzie nie tak w połowie wielkiej transakcji (brak miejsca na dysku, przerwane połączenie, konflikt zależności), system zostaje w stanie pośrednim, częściowo zaktualizowanym, co jest dużo trudniejsze do opanowania niż stan sprzed aktualizacji albo stan w pełni po niej. Po trzecie — **diagnoza winowajcy**: gdy usługa przestaje działać po aktualizacji stu pakietów naraz, wskazanie, który z nich jest odpowiedzialny, wymaga przejrzenia całej listy albo cofnięcia wszystkiego i testowania pakiet po pakiecie od zera.

Dzielenie aktualizacji na mniejsze partie — alfabetycznie, po literach nazwy pakietu, albo metodą połowienia — adresuje wszystkie trzy naraz: każda partia mieści się łatwiej w oknie, awaria jednej partii nie dotyka pakietów z pozostałych partii (które już zostały bezpiecznie zaktualizowane i przetestowane), a jeśli któraś partia faktycznie powoduje problem, wiadomo dokładnie, w jakim zakresie nazw szukać winowajcy — i tam właśnie połowienie pokazuje swoją siłę.

Analogia

To jest dokładnie **metoda używana przy szukaniu przepalonej lampki w łańcuchu świątecznych lampek starego typu**, gdzie jedna przepalona żarówka gasi cały ciąg.

Zamiast wymieniać wszystkie pięćdziesiąt żarówek po kolei i sprawdzać po każdej, czy łańcuch zaświecił (to jest odpowiednik aktualizowania pakietu po pakiecie — działa, ale trwa wieczność), dzielisz łańcuch na dwie połowy i podłączasz zasilanie do każdej osobno. Połowa, która się zaświeca, jest sprawna — możesz o niej zapomnieć. Połowa, która nie działa, zawiera winowajcę, więc dzielisz *ją* na kolejne dwie połówki i powtarzasz. Przy pięćdziesięciu żarówkach znajdziesz winną w sześciu, siedmiu próbach zamiast pięćdziesięciu.

Ta sama logika stoi za `dnf update 'a-j'` i `dnf update 'k-z'`: jeśli pierwsza połowa przechodzi bez problemu, winowajca (jeśli jest) siedzi w drugiej połowie — i tam wracasz z kolejnym podziałem, zamiast przeglądać sto pakietów na oślep.

Jak to działa

Podział alfabetyczny opiera się na tym, że `dnf list updates` (albo `dnf check-update`) wypisuje pakiety jeden pod drugim, z nazwą pakietu w pierwszej kolumnie. `awk '/^[a-j]/ {print $1}'` czyta ten wynik linia po linii, sprawdza wyrażeniem regularnym `^[a-j]`, czy linia **zaczyna się** od litery z zakresu a–j (wielkość liter ma znaczenie — pakiety pisane wielką literą, rzadkie, ale możliwe, wypadną poza ten filtr), i jeśli tak, wypisuje pierwszą kolumnę tej linii. Podstawienie `$(...)` w powłoce Bash zamienia cały ten wynik na listę argumentów przekazanych do `dnf update`, dokładnie tak samo, jakbyś wpisał nazwy pakietów ręcznie, jeden po drugim.

Ograniczenie tej metody: dzieli **liczbę pakietów**, a nie ich rozmiar ani czas instalacji. Litera `k` może objąć jeden ogromny pakiet (`kernel` ze wszystkimi modułami) porównywalny wagą do dwudziestu małych pakietów z litery `a`. Jeśli celem jest zmieszczenie się w oknie czasowym, warto sprawdzić `dnf check-update` i rozmiar pobierania (`dnf --assumeno update` pokaże sumę) przed podziałem, a nie zakładać, że połowa liter to połowa czasu.

`--exclude` i wpis `exclude=` w `dnf.conf` działają na poziomie rozwiązywania zależności: wykluczony pakiet jest traktowany tak, jakby nie miał dostępnej nowszej wersji — `dnf` planuje transakcję tak, jakby repozytorium nie oferowało dla niego nic nowego. To różni się od `--skip-broken`, które **próbuje** zaktualizować wszystko, a dopiero gdy natrafi na konflikt, wycofuje z planu tylko te pakiety, które ten konflikt powodują.

Krok po kroku

1. Zobacz pełny zakres przed podziałem.

```
dnf --assumeno update
```

```
Podsumowanie transakcji
================================================================================
Aktualizuj  91 pakietów
Rozmiar pobierania: 480 M
```

2. Podziel na dwie połowy alfabetyczne.

```
dnf update $(dnf list updates | awk '/^[a-j]/ {print $1}')
```

```
Podsumowanie transakcji
================================================================================
Aktualizuj  38 pakietów
```

3. Po powodzeniu pierwszej połowy, sprawdź działanie kluczowych usług, zanim pójdziesz dalej.

```
systemctl --failed
```

4. Jeśli wszystko w porządku, wykonaj drugą połowę.

```
dnf update $(dnf list updates | awk '/^[k-z]/ {print $1}')
```

5. Jeśli któraś połowa się wysypie — podziel ją dalej.

```
dnf update $(dnf list updates | awk '/^[a-e]/ {print $1}')
dnf update $(dnf list updates | awk '/^[f-j]/ {print $1}')
```

Powtarzasz podział wewnątrz problematycznego zakresu, aż liczba pakietów w partii, która faktycznie wywala transakcję, zejdzie do kilku sztuk — na tym poziomie zwykle łatwo już wskazać winowajcę wprost z komunikatu błędu albo przez `dnf history info last`.

6. Po namierzeniu winowajcy, zamroź go i kontynuuj resztę.

```
dnf versionlock add nazwa-winowajcy
dnf update --exclude=nazwa-winowajcy
```

Częste błędy

`--allowerasing` uruchomione z `-y` bez wcześniejszego przeglądu planu.

```
dnf update -y --best --allowerasing
```

```
Usuwanie:
 firewalld      noarch   1.3.4-1.el9   @baseos
```

Transakcja przechodzi bez pytania, ale przy okazji usuwa `firewalld`, bo `dnf` uznał to za najprostszy sposób rozwiązania konfliktu zależności — administrator dowiaduje się o tym dopiero, gdy zauważa, że zapora przestała działać. Naprawa: najpierw `dnf --assumeno update --best --allowerasing`, przeczytać sekcję „Usuwanie" w planie, dopiero potem `-y`.

Podział alfabetyczny bez cudzysłowów wokół wzorca w interaktywnej powłoce.

```
dnf update a* b* c*
```

```
-bash: brak dopasowania: a*
```

W tej wersji polecenia gwiazdka jest przekazywana bezpośrednio jako argument `dnf update`, więc to nie ten sam wzorzec co w `awk`. Powłoka próbuje najpierw rozwinąć `a*` jako nazwę pliku w bieżącym katalogu — jeśli żaden plik na `a` tam nie istnieje, Bash (z domyślnym `nullglob` wyłączonym) zgłasza błąd zamiast przekazać dosłowne `a*` do `dnf`. Naprawa: cudzysłów wokół każdego wzorca (`dnf update 'a*' 'b*' 'c*'`), żeby to `dnf`, a nie powłoka, interpretował gwiazdkę jako wzorzec nazwy pakietu.

`--skip-broken` używane rutynowo, bez sprawdzenia, co dokładnie zostało pominięte.

```
Pakiety pominięte z powodu zależności:
    openssl-libs-1:3.0.7-27.el9_4.x86_64 z instalacji
```

Transakcja kończy się sukcesem, więc administrator przechodzi dalej — a `openssl-libs` zostaje na starej wersji z niezałataną luką bezpieczeństwa, bez żadnego wyraźnego alertu poza jedną linią w logu, którą łatwo przewinąć. Naprawa: po każdym `--skip-broken` uruchom `dnf check-update`, żeby zobaczyć, co nadal czeka, i zdiagnozuj pominięty konflikt osobno, zamiast zostawiać go na zawsze.

Podział na pół zastosowany, gdy problem wcale nie leży w konkretnym pakiecie.

```
dnf update $(dnf list updates | awk '/^[a-j]/ {print $1}')
Błąd: Brak miejsca na dysku w filesystemie /var
```

Metoda połowienia zakłada, że winowajcą jest jeden konkretny pakiet z konfliktem zależności — jeśli błąd to w rzeczywistości brak miejsca na dysku (`/var`, gdzie `dnf` cache'uje pobrane pakiety) albo problem sieciowy z repozytorium, żadna liczba podziałów alfabetycznych tego nie naprawi, bo błąd nie zależy od tego, które konkretnie pakiety są w partii. Naprawa: przeczytaj treść błędu przed sięgnięciem po metodę połowienia — `dnf clean packages` i sprawdzenie `df -h /var` to szybszy pierwszy krok niż godzina dzielenia listy na pół.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Żadne ćwiczenie nie instaluje realnych pakietów — wszystkie kroki z `dnf update` są zastąpione `--assumeno` albo `repoquery`, żeby zobaczyć plan bez wykonania.

Ćwiczenie 1 — podział listy aktualizacji na litery bez ich instalowania

```
cd /tmp
dnf check-update > lista.txt 2>/dev/null || true
awk '/^[a-j]/ {print $1}' lista.txt > polowa1.txt
awk '/^[k-z]/ {print $1}' lista.txt > polowa2.txt
wc -l polowa1.txt polowa2.txt
```

Jeśli lista jest pusta (system w pełni zaktualizowany), z czego to wynika i jak inaczej mógłbyś to ćwiczenie zasymulować?

Odpowiedź

Pusta lista oznacza, że `dnf check-update` nie znalazł żadnych pakietów z dostępną nowszą wersją — system jest aktualny albo repozytoria nie są skonfigurowane/dostępne. Symulacja bez prawdziwych aktualizacji: wygeneruj sztuczną listę poleceniem `rpm -qa | sort | head -20 > lista.txt` i przećwicz te same komendy `awk` na niej — mechanika podziału jest identyczna, tylko dane wejściowe są inne.

Ćwiczenie 2 — suchy przebieg z wykluczeniem

```
cd /tmp
dnf --assumeno update --exclude=kernel* 2>&1 | tee wynik.txt
grep -i "pominięto\|kernel" wynik.txt
```

Czym różniłby się ten wynik od zwykłego `dnf --assumeno update` bez `--exclude`?

Odpowiedź

Plan transakcji z `--exclude=kernel*` nie zawierałby żadnego pakietu jądra (`kernel`, `kernel-core`, `kernel-modules`) niezależnie od tego, czy jest dla nich dostępna aktualizacja — `dnf` planuje tak, jakby te pakiety nie miały nowszej wersji do zaoferowania. Bez `--exclude` plan objąłby też jądro, jeśli aktualizacja jest dostępna, co zwykle oznacza też konieczność restartu po instalacji.

Pytania kontrolne

1. Masz 80 pakietów do aktualizacji i podzieliłeś je na pół alfabetycznie. Pierwsza połowa (a–j) przechodzi bez problemu, druga (k–z) zgłasza konflikt zależności. Jaki jest kolejny krok metody połowienia?

Odpowiedź

Dzielisz zakres k–z na kolejne dwie połowy, np. k–r i s–z, i aktualizujesz je osobno. Powtarzasz podział wewnątrz zakresu, który akurat zawiera problem, aż liczba pakietów w problematycznej partii zejdzie do rozmiaru, w którym da się wskazać winowajcę wprost z komunikatu błędu.

2. Czym różni się `--skip-broken` od `--allowerasing` w podejściu do konfliktu zależności?

Odpowiedź

`--skip-broken` po prostu pomija z transakcji te pakiety, które powodują konflikt, zostawiając je na starej wersji — reszta aktualizacji przechodzi, problematyczny pakiet zostaje nietknięty. `--allowerasing` aktywnie próbuje rozwiązać konflikt, w razie potrzeby usuwając pakiet, który stoi na przeszkodzie — może to być pakiet inny niż ten, który się aktualizuje, co bywa zaskakujące, jeśli nie przeczyta się planu transakcji przed potwierdzeniem.

3. Po co w ogóle dzielić dużą aktualizację na części, skoro `dnf history undo` i tak pozwala cofnąć całą transakcję naraz?

Odpowiedź

Trzy niezależne powody: po pierwsze okno serwisowe — duża transakcja może nie zmieścić się w przydzielonym czasie, niezależnie od tego, czy da się ją później cofnąć. Po drugie ograniczenie zasięgu awarii — jeśli druga połowa się wysypie, pierwsza już zaktualizowana i sprawdzona połowa nie musi być cofana razem z nią. Po trzecie diagnoza — cofnięcie całej transakcji przez `history undo` nie mówi, który konkretnie pakiet spowodował problem, tylko usuwa efekt całej operacji naraz; dzielenie na części od razu zawęża podejrzanych.

4. `dnf update 'a*'` zwraca błąd o braku miejsca na dysku w `/var`, a nie o konflikcie zależności. Czy dalszy podział alfabetyczny pomoże znaleźć przyczynę?

Odpowiedź

Nie. Metoda połowienia ma sens tylko wtedy, gdy problem zależy od tego, które konkretnie pakiety znalazły się w danej partii — czyli przy konfliktach zależności czy plikach kolidujących między pakietami. Brak miejsca na dysku jest niezależny od podziału: nawet najmniejsza partia pakietów nie pobierze się, jeśli `/var` (gdzie `dnf` trzyma cache pobranych pakietów) jest pełne. W takim wypadku właściwym krokiem jest `dnf clean packages` i sprawdzenie `df -h /var`, nie dalsze dzielenie listy.

Powiązane: dnf: instalacja, aktualizacja, historia, wycofywanie zmian · Repozytoria: konfiguracja, EPEL, priorytety · Kernele: lista, usuwanie starych, brak miejsca na /boot

Źródła: `man 8 dnf` (sekcje FILTER OPTIONS, PACKAGE MATCHING) · `man 5 dnf.conf` · `man 1 awk` · materiały wewnętrzne — procedura podziału dużych aktualizacji na produkcji

---

Revision #15
Created 2026-09-18 06:56:57 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:34 UTC by Tomasz Konieczny