Historia komend: konfiguracja i przeszukiwanie

Poziom: podstawy

Dotyczy: bash (domyślnie RHEL/Rocky 9–10), różnice dla innych powłok zaznaczone

Czas: ~18 min

Powłoka pamięta, co wpisałeś — ale zapamiętuje to w dwóch różnych miejscach naraz, w dwóch różnych momentach, i domyślnie robi to w sposób, który w wieloterminalowej pracy potrafi zgubić połowę historii. Ta strona to konfiguracja i przeszukiwanie od podstaw po pułapki.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Fiszka

TYLKO ODCZYT

Wypisanie całej historii bieżącej sesji: `history`

```
history
```

Wypisuje wszystkie zapamiętane polecenia z numerem porządkowym. Numer jest ważny — to po nim odwołujemy się do konkretnej komendy przez `!N`.

Przykład:

```
  495  cd /var/log
  496  tail -f messages
  497  systemctl status nginx
  498  history
```

🔒 TYLKO ODCZYT

Wypisanie ostatnich N wpisów: `history 20`

```
history 20
```

Ten sam wynik co samo `history`, ale obcięty do ostatnich dwudziestu linii. Wygodniejsze niż `history | tail -20`, bo nie uruchamia drugiego procesu.

Przykład:

```
  479  vim /etc/fstab
  480  mount -a
  ...
  498  history 20
```

🔒 TYLKO ODCZYT

Szukanie w historii: `history | grep wzorzec`

```
history | grep systemctl
```

Klasyczne przeszukiwanie, gdy pamiętamy fragment komendy, ale nie pamiętamy, kiedy jej użyliśmy ani z jakimi dokładnie flagami.

Przykład:

```
  432  systemctl restart nginx
  497  systemctl status nginx
```

🔒 TYLKO ODCZYT

Wypisanie samych komend, bez numeracji i bez znaczników czasu

```
history | awk '{$1=""; print substr($0,2)}'
```

Zeruje pierwsze pole (numer wpisu) i wycina je z linii przez `substr`, zostawiając samą treść komendy. Przydaje się, gdy chcemy przekleić fragment historii do skryptu i numery porządkowe tylko przeszkadzają.

Przykład:

```
cd /var/log
tail -f messages
systemctl status nginx
```

Ta sama sztuczka z innym przesunięciem. Jeśli w powłoce ustawiona jest `HISTTIMEFORMAT` (patrz niżej), każda linia `history` ma dodatkowo wbity znacznik czasu i przesunięcie w `substr` trzeba zwiększyć, bo numer porządkowy i data razem zajmują więcej znaków:

```
history | awk '{$1=""; print substr($0,22)}'
```

Wersja bez `HISTTIMEFORMAT` działa z mniejszym przesunięciem, zwykle bliskim `substr($0,19)` — dokładna liczba zależy od szerokości numeru porządkowego (4 czy 5 cyfr) i długości znacznika czasu. Prostszy, mniej wrażliwy na szerokość kolumn wariant:

```
history | cut -c 8-
```

`cut -c 8-` obcina pierwsze siedem znaków (numer i spacje wyrównujące), zostawiając resztę linii — działa dopóki numeracja mieści się w typowej szerokości kolumny; przy bardzo długiej historii i pięciocyfrowych numerach próg trzeba przesunąć.

□□ TYLKO ODCZYT

Czyszczenie bufora w pamięci: `history -c`

```
history -c
```

Kasuje historię **tylko w bieżącej sesji**, w pamięci powłoki. Plik `~/.bash_history` na dysku zostaje nietknięty, dopóki coś go nie nadpisze (patrz sekcja *Częste błędy*).

□□ TYLKO ODCZYT

Zapisanie bufora do pliku: `history -w`

```
history -w
```

Wymusza natychmiastowy zapis całego bufora historii bieżącej sesji do pliku historii, bez czekania na zamknięcie powłoki. Bez argumentu pisze do domyślnego `$HISTFILE` (zwykle `~/.bash_history`); z argumentem — do wskazanego pliku.

🔒 TYLKO ODCZYT

Wczytanie pliku do bufora: `history -r`

```
history -r
```

Odwrotność `-w`: dokłada zawartość pliku historii do bufora bieżącej sesji, nie zastępując tego, co już w niej jest. Razem z `-w` to para poleceń, na której opiera się mechanizm dzielenia historii między terminalami opisany niżej.

🔒 TYLKO ODCZYT

Usunięcie jednego wpisu: `history -d N`

```
history -d 495
```

Usuwa z bufora wpis o konkretnym numerze — typowo tę komendę, w której przez pomyłkę wpisałeś hasło w linii poleceń. Samo `-d` nie dotyka pliku na dysku; żeby usunięcie było trwałe, trzeba potem wywołać `history -w`.

🔒 TYLKO ODCZYT

Powtórzenie ostatniej komendy: `!!`

```
sudo !!
```

> **Pojęcie: rozwinięcie historii (history expansion).** Mechanizm powłoki, który zanim cokolwiek uruchomi, podmienia w wpisanej linii specjalne sekwencje zaczynające się od `!` na fragmenty wcześniej wykonanych komend — dokładnie tak samo, jak ekspansja zmiennych podmienia `$HOME` na ścieżkę, zanim program w ogóle zobaczy argumenty. `!!` rozwija się do całej poprzedniej komendy, więc `sudo !!` to najczęstszy wzorzec: odpaliliśmy coś bez uprawnień, dostaliśmy „Permission denied”, i

> zamiast przepisywać komendę na nowo, dopisujemy `sudo` z przodu.

Przykład:

```
$ apt update
E: Could not open lock file /var/lib/dpkg/lock-frontend - Permission denied
$ sudo !!
sudo apt update
[dalej normalny wynik apt update]
```

Powłoka wypisuje rozwiniętą komendę przed jej wykonaniem — to nie jest cichy skrót, widzimy dokładnie, co się uruchomi.

□□ ZMIENIA SYSTEM

Powtórzenie ostatniego argumentu poprzedniej komendy: `!$`

```
mkdir /opt/aplikacja/dane
cd !$
```

`!$` rozwija się do ostatniego słowa poprzedniej linii poleceń. Tutaj: `cd /opt/aplikacja/dane`, bez przepisywania ścieżki po raz drugi. Działa też w środku dłuższych sekwencji, np. `chmod 750 !$` zaraz po `touch /var/log/app/audit.log`.

□□ **mkdir** = make directory: utwórz katalog · **cd** = change directory: przejdź do katalogu

□□ TYLKO ODCZYT

Powtórzenie komendy o numerze N: `!N`

```
!497
```

Rozwija się dokładnie do komendy z numerem 497 z listy `history`. Trzeba najpierw sprawdzić numer przez samo `history` albo `history | grep`.

□□ TYLKO ODCZYT

Powtórzenie ostatniej komendy zaczynającej się od wzorca: `!wzorzec`

```
!systemctl
```

Szuka wstecz pierwszej (licząc od końca) komendy w historii, która zaczyna się od `systemctl`, i uruchamia ją ponownie. Szybkie, ale ryzykowne — jeśli ostatnią pasującą komendą było `systemctl stop nginx`, a myśleliśmy o `systemctl status`, uruchomimy coś innego niż zamierzaliśmy. Zawsze warto najpierw sprawdzić przez `!wzorzec:p` (wypisuje rozwiniętą komendę bez jej uruchamiania).

TYLKO ODCZYT

Szybka poprawka ostatniej komendy: `^stare^nowe`

```
$ systemclt status nginx
bash: systemclt: command not found
$ ^systemclt^systemctl
systemctl status nginx
```

Podmienia pierwsze wystąpienie `stare` na `nowe` w ostatniej komendzie i od razu ją uruchamia. Odruch po literówce, szybszy niż strzałka w górę i ręczna edycja.

systemctl = system control: sterowanie usługami systemd

TYLKO ODCZYT

Przeszukiwanie wsteczne historii: `Ctrl+R`

Skrót klawiszowy w trybie interaktywnym, nie polecenie do wpisania. Naciskamy `Ctrl+R`, zaczynamy pisać fragment komendy, a powłoka na bieżąco pokazuje najnowsze pasujące dopasowanie z historii.

Przykład:

```
(reverse-i-search)`syst': systemctl status nginx
```

Kolejne `Ctrl+R` przy tym samym fragmencie przesuwa dopasowanie coraz dalej wstecz. `Enter` uruchamia znalezioną komendę, strzałka w prawo/lewo przenosi ją do linii edycji bez uruchamiania.

TYLKO ODCZYT

Anulowanie przeszukiwania: `Ctrl+G`

Wychodzi z trybu `reverse-i-search` bez uruchamiania czegokolwiek i czyści linię poleceń. Przydaje się, gdy `Ctrl+R` znalazło coś zupełnie nie na temat i łatwiej zacząć od zera niż cofać się skrótem.

□□ ZMIENIA SYSTEM

Ustawienie liczby wpisów w pamięci sesji: `HISTSIZE`

```
export HISTSIZE=5000
```

Limit liczby poleceń przechowywanych w buforze historii **bieżącej, żywej powłoki**. Nie ma wpływu na to, ile trafi ostatecznie na dysk — od tego jest `HISTFILESIZE`.

□□ **export** = ustaw zmienną środowiskową widoczną dla uruchamianych programów

□□ ZMIENIA SYSTEM

Ustawienie liczby wpisów w pliku na dysku: `HISTFILESIZE`

```
export HISTFILESIZE=10000
```

Limit liczby linii w pliku `~/.bash_history` po zapisaniu bufora. Zwyczajowo ustawia się większy niż `HISTSIZE`, żeby plik na dysku trzymał dłuższą historię niż to, co mieści się w pamięci pojedynczej sesji.

□□ **export** = ustaw zmienną środowiskową widoczną dla uruchamianych programów

□□ ZMIENIA SYSTEM

Filtrowanie duplikatów w historii: `HISTCONTROL=ignoredups:erasedups`

```
export HISTCONTROL=ignoredups:erasedups
```

`ignoredups` nie zapisuje komendy, jeśli jest identyczna z bezpośrednio poprzedzającą. `erasedups` idzie dalej: przy każdym nowym wpisie usuwa z historii **wszystkie** wcześniejsze wystąpienia tej samej komendy, więc `cd /var/log` wpisane dwadzieścia razy w ciągu dnia zostawia w historii tylko ostatnie wystąpienie, na końcu. Dodatkowa wartość `ignorespace` (często łączona jako `ignoreboth`) pomija komendy zaczynające się od spacji — patrz sekcja *Koniecznie zapamiętaj* niżej.

export = ustaw zmienną środowiskową widoczną dla uruchamianych programów

ZMIENIA SYSTEM

Pomijanie konkretnych wzorców: `HISTIGNORE`

```
export HISTIGNORE='ls:ls -la:cd:cd -:exit:clear'
```

Lista wzorców rozdzielonych dwukropkiem — komendy pasujące do któregokolwiek z nich w ogóle nie trafiają do historii. Sensowne dla poleceń, które wpisujemy dziesiątki razy dziennie i które nigdy nie chcemy odszukiwać przez `Ctrl+R` czy `!wzorzec`.

export = ustaw zmienną środowiskową widoczną dla uruchamianych programów

ZMIENIA SYSTEM

Dodanie znaczników czasu do wpisów: `HISTTIMEFORMAT='%F %T '`

```
export HISTTIMEFORMAT='%F %T '
history | tail -3
```

Bez tej zmiennej `history` w ogóle nie pokazuje, *kiedy* dana komenda została wykonana — zapisuje tylko treść. Format jest zgodny ze `strftime` z `man 3 strftime`.

Przykład:

```
  498  2026-09-18 09:12:03 systemctl status nginx
  499  2026-09-18 09:14:41 journalctl -u nginx -n 50
  500  2026-09-18 09:15:07 history
```

export = ustaw zmienną środowiskową widoczną dla uruchamianych programów

TYLKO ODCZYT

Dopisywanie historii zamiast nadpisywania: `shopt -s histappend`

```
shopt -s histappend
```

Włącza tryb, w którym powłoka przy zamknięciu **dopisuje** swoją historię na koniec pliku, zamiast go nadpisywać. Bez tej opcji ostatnio zamknięta sesja terminala kasuje historię wszystkich sesji, które zamknęły się wcześniej — pełne wyjaśnienie w sekcji *Częste błędy*.

shopt = shell options: włącz albo wyłącz dodatkowe opcje zachowania powłoki bash

ZMIENIA SYSTEM

Natychmiastowy zapis historii po każdej komendzie: `PROMPT_COMMAND='history -a'`

```
export PROMPT_COMMAND='history -a'
```

`history -a` (bez `-c`, bez `-w` — to trzecia, osobna flaga) dopisuje do pliku historii tylko te wpisy z bieżącej sesji, których jeszcze tam nie ma, zamiast zapisywać cały bufor. Ustawione jako `PROMPT_COMMAND`, wykonuje się tuż przed pokazaniem każdego nowego promptu, więc historia trafia na dysk niemal w czasie rzeczywistym, a nie dopiero przy zamknięciu terminala. To jest mechanizm, który realnie daje wspólną, bieżącą historię między kilkoma otwartymi terminalami — `histappend` samo w sobie nie wystarcza, bo bez `history -a` każda sesja i tak zapisuje dopiero na końcu swojego życia.

export = ustaw zmienną środowiskową widoczną dla uruchamianych programów

TYLKO ODCZYT

Sprawdzenie pliku historii i jego uprawnień: `~/.bash_history`

```
ls -l ~/.bash_history
```

Przykład:

```
-rw------- 1 adm_user adm_user 48213 wrz 18 09:15 /home/adm_user/.bash_history
```

Domyślne uprawnienia to `600` — tylko właściciel czyta i pisze. To nieprzypadkowe: w pliku historii regularnie lądują hasła, tokeny i adresy wewnętrznych systemów wklejone w linię poleceń. Jeśli uprawnienia są szersze (`644` po ręcznej edycji edytorem, który je zresetował), każdy użytkownik lokalny odczyta cudzą historię komend.

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Wyłączenie zapisywania historii na czas wpisywania hasła: `set +o history`

```
set +o history
mysql -u root -pHaslo123
set -o history
```

Tymczasowo wyłącza mechanizm historii dla powłoki interaktywnej — nic, co wpiszemy między `set +o history` a `set -o history`, nie trafi do bufora ani do pliku. Jedyny pewny sposób na wpisanie hasła w linii poleceń bez zostawienia go w historii, gdy naprawdę nie da się użyć zmiennej środowiskowej ani pliku z hasłem.

TYLKO ODCZYT

Wszystkie zmienne i flagi

Element	Co robi
`history`	wypisuje historię bieżącej sesji
`history N`	ostatnie N wpisów
`history -c`	czyści bufor w pamięci (nie plik)
`history -w [SciezkaDoPliku]`	zapisuje bufor do pliku historii
`history -r [SciezkaDoPliku]`	wczytuje plik historii do bufora
`history -a`	dopisuje do pliku nowe wpisy z bieżącej sesji
`history -d N`	usuwa wpis numer N z bufora
`!!`	ostatnia komenda
`!N`	komenda o numerze N
`!wzorzec`	ostatnia komenda zaczynająca się od wzorca
`!$`	ostatni argument poprzedniej komendy
`^stare^nowe`	poprawka i ponowne uruchomienie ostatniej komendy
`HISTSIZE`	liczba wpisów w pamięci sesji
`HISTFILESIZE`	liczba linii w pliku na dysku
`HISTCONTROL`	filtrowanie duplikatów / spacji (`ignoredups`, `ignorespace`, `erasedups`)
`HISTIGNORE`	wzorce pomijane przy zapisie
`HISTTIMEFORMAT`	znaczniki czasu przy wpisach
`shopt -s histappend`	dopisywanie zamiast nadpisywania pliku przy zamknięciu

Uważaj

- `history -c` nie usuwa nic z dysku — to częsta pomyłka osób, które chcą „wyczyścić ślady”.
- Domyślnie historia trafia do pliku dopiero **przy zamknięciu powłoki**, nie na bieżąco.
- Hasło wpisane wprost w linii poleceń (`mysql -pHaslo123`) ląduje w historii na trwałe, chyba że aktywnie temu zapobiegniemy.

Z życia wzięte

◫◫ Z ŻYCIA WZIĘTE

Środa, 17:00: Tomek i Ola mają otwarte po jednym terminalu na app01 przez cały dzień. Tomek zamyka swój pierwszy, zapisując godzinę pracy do `~/.bash_history`, a pół godziny później Ola zamyka swój i cała historia Tomka znika bez śladu i bez błędu. Dopiero `shopt -s histappend` w `.bashrc` obu kont sprawia, że kolejne zamknięcia dopisują się do pliku, zamiast go nadpisywać.

Po co to istnieje

Pojęcie: bufor historii i plik historii. To dwa osobne miejsca, które łatwo pomylić. Bufor historii to lista poleceń trzymana **w pamięci** aktualnie działającej powłoki — rośnie z każdą wpisaną komendą i znika, gdy proces powłoki się kończy. Plik historii (domyślnie `~/.bash_history`, ścieżkę trzyma zmienna `$HISTFILE`) to zapis **na dysku**, który przeżywa zamknięcie terminala i z którego nowo otwarta powłoka odczytuje swój startowy bufor. `history -c` czyści tylko to pierwsze; `history -w` przenosi zawartość pierwszego do drugiego. Cała reszta tej strony to w gruncie rzeczy opis tego, kiedy dokładnie i jak te dwa miejsca się ze sobą synchronizują.

Powód, dla którego to w ogóle ma znaczenie: bez historii każda sesja terminala byłaby amnezyjna. Strzałka w górę, `Ctrl+R`, `!!` — wszystko to działa tylko dlatego, że powłoka trzyma gdzieś zapis tego, co już wpisałeś. Dla administratora to nie wygoda, tylko narzędzie pracy: rekonstrukcja tego, co dokładnie zrobiono na serwerze przed awarią, w dużej mierze opiera się na przejrzeniu `~/.bash_history` (i, na porządnie skonfigurowanym systemie, scentralizowanego logowania sesji, co jest osobnym tematem).

Analogia

Wyobraź sobie **dziennik pokładowy pisany ołówkiem, z gumką pod ręką**.

Kapitan zapisuje w nim każdy manewr w trakcie rejsu — ale robi to na brudnopisowej, wyrywanej kartce, która trafia do właściwej, oprawionej księgi dopiero w porcie, przy końcu rejsu. Dopóki statek jest na morzu, brudnopis żyje własnym życiem: można coś z niego wymazać gumką (`history -c`), można dopisać coś z pamięci (`history -r`), a oficjalna księga w porcie nie wie nic o tym, co się dzieje na pokładzie, dopóki ktoś fizycznie nie przeniesie zapisów.

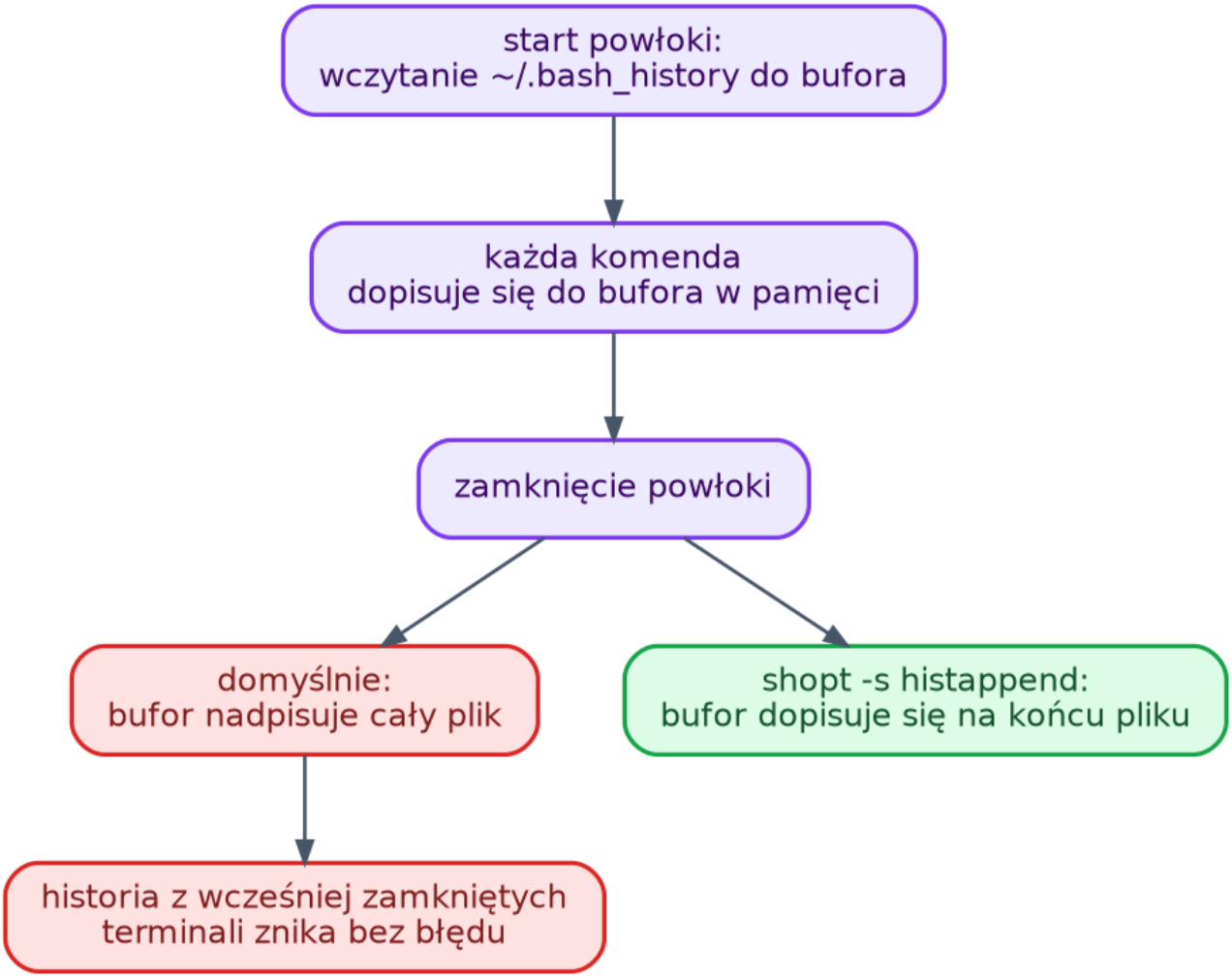
I tu jest haczyk, który zaskakuje najczęściej: jeśli dwa statki wracają do tego samego portu tego samego dnia i obydwa mają swoje brudnopisy, **ten, który dobija do brzegu jako drugi, nadpisuje w księdze to, co zapisał pierwszy** — chyba że port ma zasadę „dopisuj na końcu, nie zaczynaj księgi od nowa". Dokładnie to jest `histappend`: bez niego druga zamknięta sesja terminala kasuje w pliku historię tego, co zdążyła zapisać sesja zamknięta wcześniej.

Jak to działa

Rozwiń: jak to działa pod spodem

Domyślny cykl życia historii w interaktywnej powłoce `bash` wygląda tak:

1. Przy starcie powłoka czyta plik wskazany przez `$HISTFILE` (domyślnie `~/.bash_history`) i ładuje go do bufora w pamięci.
2. Każda wpisana komenda dopisuje się do bufora w pamięci — natychmiast, ale **tylko w pamięci**.
3. Przy normalnym zamknięciu powłoki (`exit`, zamknięcie okna terminala, wylogowanie) bufor z pamięci zapisuje się do pliku na dysku, **nadpisując jego dotychczasową zawartość**, chyba że włączona jest opcja `histappend` (`shopt -s histappend`) — wtedy zapis dopisuje się na końcu pliku zamiast go zastępować.



Cykl życia historii: bufor w pamięci trafia do pliku dopiero przy zamknięciu powłoki, domyślnie nadpisując go, z histappend dopisując się na końcu (kliknięcie otwiera pełny rozmiar)

Krok trzeci to źródło najbardziej mylącego zachowania domyślnej konfiguracji: jeśli masz otwarte dwa terminale, terminal A i terminal B, i zamkniesz najpierw A, a potem B, to przy zamknięciu B jego bufor pamięci **nadpisuje** to, co A zdążyło zapisać — bo B nie wie nic o zmianach, które zaszły w pliku po tym, jak B wystartowało i wczytało go do własnego bufora. Historia komend wpisanych w terminalu A po prostu znika, mimo że A zamknęło się poprawnie.

`shopt -s histappend` rozwiązuje połowę problemu: zamiast nadpisywać, każda zamykana sesja dopisuje swój bufor na końcu pliku. Ale to wciąż dzieje się dopiero **przy zamknięciu** — jeśli terminal padnie (zawieszenie SSH, zamknięcie okna przez system, `kill -9` na powłoce), bufor pamięci ginie bezpowrotnie, bo nigdy nie doszło do zapisu. Drugą połowę problemu rozwiązuje `PROMPT_COMMAND='history -a'`: `history -a` dopisuje do pliku nowe wpisy zaraz po każdej komendzie, więc nawet nagłe zerwanie sesji traci najwyżej ostatnią, jeszcze niewykonaną linię — nie całą historię dnia.

Trzecia rzecz, o której trzeba pamiętać, to że **historia i plik historii to nie to samo co logowanie audytowe**. Historia jest własnością użytkownika: może ją wyczyścić, wyłączyć na chwilę (`set +o history`), edytować plik ręcznie. Do rzetelnego, niemodyfikowalnego zapisu tego, co ktoś zrobił na serwerze produkcyjnym, służą inne mechanizmy — `auditd`, scentralizowane logowanie sesji SSH, `script` z zapisem poza kontrolą użytkownika. `history` to narzędzie wygody dla właściciela powłoki, nie mechanizm bezpieczeństwa.

Krok po kroku

1. Sprawdź bieżącą konfigurację historii.

```
echo "HISTSIZE=$HISTSIZE HISTFILESIZE=$HISTFILESIZE HISTCONTROL=$HISTCONTROL"
shopt histappend
```

```
HISTSIZE=1000 HISTFILESIZE=2000 HISTCONTROL=
histappend      off
```

Domyślna instalacja RHEL/Rocky ma `histappend` wyłączone i `HISTCONTROL` puste — czyli oba problemy opisane wyżej występują od razu po instalacji systemu.

2. Skonfiguruj rozsądne wartości w `~/.bashrc`.

```
cat >> ~/.bashrc <<'EOF'
export HISTSIZE=5000
export HISTFILESIZE=10000
export HISTCONTROL=ignoreboth:erasedups
export HISTTIMEFORMAT='%F %T '
shopt -s histappend
export PROMPT_COMMAND='history -a'${PROMPT_COMMAND:+; $PROMPT_COMMAND}
EOF
```

`ignoreboth` to skrót łączący `ignoredups` (nie zapisuj powtórzenia poprzedniej komendy) i `ignorespace` (nie zapisuj komend zaczynających się od spacji — patrz *Częste błędy*).

3. Wczytaj konfigurację i sprawdź efekt.

```
source ~/.bashrc
echo "HISTCONTROL=$HISTCONTROL"
```

```
HISTCONTROL=ignoreboth:erasedups
```

4. Sprawdź, czy wspólna historia rzeczywiście działa.

Otwórz drugi terminal do tej samej maszyny, wykonaj w nim jakąś komendę, wróć do pierwszego terminala i wpisz:

```
history -n
```

`-n` (mała litera, inna flaga niż opisane wyżej `-c` / `-w` / `-r` / `-a`) dopisuje do bufora bieżącej sesji wpisy z pliku historii, których jeszcze w buforze nie ma — czyli te, które druga sesja właśnie zapisała przez `history -a`. Jeśli konfiguracja z kroku 2 działa, komenda z drugiego terminala pojawi się na liście bez zamykania żadnej z sesji.

Częste błędy

Przekonanie, że `history -c` usuwa historię z dysku.

```
$ history -c
$ exit
```

Po ponownym zalogowaniu cała wcześniejsza historia wraca — bo `history -c` wyczyściło tylko bufor w pamięci, a plik `~/.bash_history` na dysku został nietknięty, i przy `exit` bufor (teraz pusty albo z nowymi komendami z tej sesji) i tak coś do niego zapisuje albo dopisuje. Żeby faktycznie usunąć zapisaną historię: `history -c && history -w`, co czyści bufor i natychmiast nadpisuje nim (czyli pustką) plik na dysku.

Zgubienie historii z jednej sesji przez brak `histappend`.

```
[terminal A, zamknięty jako pierwszy – jego historia zapisana]
[terminal B, zamknięty jako drugi]
$ history | tail -5
```

Terminal A pracował godzinę, wpisał trzydzieści komend, zamknął się poprawnie. Terminal B był otwarty równolegle od dłuższego czasu i przy zamknięciu **nadpisał** cały plik `~/.bash_history` swoim własnym buforem — trzydzieści komend z terminala A po prostu znika, bez żadnego komunikatu błędu, bo z punktu widzenia powłoki nic się nie zepsuło. Naprawa: `shopt -s histappend` w `~/.bashrc`, najlepiej razem z `PROMPT_COMMAND='history -a'`, żeby problem nie wracał przy kolejnym zerwaniu sesji.

Hasło w linii poleceń zostaje w historii na stałe.

```
$ mysql -u root -pTajneHaslo123
$ history | grep mysql
  512  mysql -u root -pTajneHaslo123
```

Komunikatu błędu nie ma — to jest właśnie problem: nic nie ostrzega, że hasło zostało zapisane. Trafia do `~/.bash_history`, skąd przeżyje miesiące, i każdy, kto kiedykolwiek uzyska dostęp do tego pliku (kopia zapasowa, migracja konta, przejęty dostęp), dostaje hasło gratis. Naprawy: zmienna środowiskowa czytana przez program zamiast flagi (`MYSQL_PWD=Tajne mysql -u root`), plik z hasłem podawany flagą (`--defaults-extra-file`), albo `set +o history` przed wpisaniem komendy i `set -o history` zaraz po. Jeśli szkoda już się stała: `history -d 512` usuwa wpis z bufora, ale trzeba też ręcznie wyedytować `~/.bash_history`, bo `-d` nie dotyka pliku na dysku, dopóki nie zawoła się `history -w`.

`HISTCONTROL=ignorespace` bez świadomości, że wymaga spacji na początku.

```
$ export HISTCONTROL=ignorespace
$  mysql -u root -pTajneHaslo123
```

Zadziała tylko wtedy, gdy komenda zaczyna się **dosłownie od spacji** — łatwo o tym zapomnieć w pośpiechu i wpisać komendę normalnie, bez wiodącej spacji, po czym zdziwić się, że jednak trafiła do historii. To nie jest błąd narzędzia, tylko złudne poczucie bezpieczeństwa: `ignorespace` działa, ale wymaga pamiętać o dodatkowym naciśnięciu spacji za każdym razem, co w praktyce zawodzi częściej niż się wydaje. Pewniejszy mechanizm to `set +o history` z kroku wyżej.

Ustawienie `HISTTIMEFORMAT` dopiero po latach pracy bez niego.

```
$ history | tail -3
  980  systemctl restart nginx
  981  journalctl -xe
  982  history
```

Bez `HISTTIMEFORMAT` `history` nigdy nie pokazywała, *kiedy* coś zostało wykonane — ta informacja nie jest w ogóle zapisywana w widocznej formie, dopóki zmienna nie zostanie ustawiona (technicznie znacznik czasu w pliku historii istnieje jako osobna linia zaczynająca się od `#`, ale `history` bez `HISTTIMEFORMAT` go nie pokazuje). Efekt: przy próbie rekonstrukcji „co się działo na serwerze między 14:00 a 14:15 w dniu awarii” administrator odkrywa, że ma treść komend, ale zero informacji, kiedy zostały wykonane, i musi zgadywać z innych logów. To jedna z tych zmiennych, które ustawia się raz — i natychmiast żałuje, że nie wcześniej.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie. Nic nie wymaga uprawnień administratora, jeśli nie napisano inaczej.

Ćwiczenie 1 — histappend na własnej skórze

```
cd /tmp
shopt -u histappend   # tymczasowo wyłącz, żeby zobaczyć domyślne zachowanie
```

Otwórz dwa terminale (dwie osobne sesje `bash`) do tej samej maszyny. W pierwszym wpisz `echo terminal-A-1`, `echo terminal-A-2`, potem zamknij go przez `exit`. W drugim wpisz `echo terminal-B-1`, poczekaj chwilę, potem też zamknij przez `exit`. Otwórz nowy terminal i wykonaj `tail -5 ~/.bash_history`.

Czy widzisz komendy z terminala A? Włącz `shopt -s histappend` (trwale, w `~/.bashrc`) i powtórz eksperyment.

Odpowiedź

Bez `histappend` terminal B, zamknięty jako drugi, nadpisał plik swoim buforem — komend `echo terminal-A-1` i `echo terminal-A-2` w pliku nie ma, jest tylko `echo terminal-B-1`. Po włączeniu `shopt -s histappend` powtórzenie eksperymentu pokazuje w pliku komendy z obu terminali, w kolejności zamykania sesji.

Ćwiczenie 2 — bezpieczne hasło w linii poleceń

```
cd /tmp
set +o history
echo "TO JEST TAJNE: haslo-testowe-123"
set -o history
history | tail -3
```

Sprawdź, czy komenda z `echo "TO JEST TAJNE..."` pojawia się w historii. Potem wykonaj tę samą komendę bez `set +o history` dookoła i porównaj wynik `history | tail -3`.

Odpowiedź

Za pierwszym razem komenda z tajnym tekstem nie pojawia się w `history | tail -3` — widać tylko `set -o history` i to, co wpisano po nim. Za drugim razem (bez wyłączania historii) komenda z hasłem jest widoczna w całości, tak jak każda inna wpisana linia.

Pytania kontrolne

1. Czym różni się `history -c` od `history -w`?

Odpowiedź

`history -c` czyści bufor historii w pamięci bieżącej sesji, nie dotykając pliku na dysku. `history -w` robi coś przeciwnego do czyszczenia: zapisuje bieżący bufor pamięci do pliku, nadpisując jego zawartość. Żeby trwale wyczyścić historię na dysku, trzeba obu naraz: `history -c && history -w`.

2. Masz otwarte trzy terminale przez cały dzień pracy i chcesz, żeby komenda wpisana w jednym z nich była od razu widoczna przez `Ctrl+R` w pozostałych, bez zamykania żadnego z nich. Jakich dwóch ustawień potrzebujesz razem?

Odpowiedź

`shopt -s histappend`, żeby zamykane sesje dopisywały się do pliku zamiast go nadpisywać, oraz `PROMPT_COMMAND='history -a'` (docelowo razem z `history -n` w miejscach, gdzie chcesz odświeżyć bufor bieżącej sesji), żeby zapis do pliku i odczyt z niego działy się na bieżąco, a nie dopiero przy zamknięciu terminala. Samo `histappend` nie wystarczy, bo bez `history -a` zapis wciąż czeka na `exit`.

3. Dlaczego uprawnienia `644` na `~/.bash_history` są problemem bezpieczeństwa, a `600` nie?

Odpowiedź

`644` pozwala każdemu użytkownikowi lokalnemu na odczyt pliku, a historia komend regularnie zawiera hasła i tokeny wpisane wprost w linię poleceń, adresy wewnętrznych systemów i inne dane, które nie powinny być publiczne nawet w obrębie jednej maszyny. `600` ogranicza odczyt i zapis wyłącznie do właściciela pliku, co jest domyślnym i poprawnym ustawieniem tworzonym automatycznie przez powłokę.

4. Ustawiłeś `HISTCONTROL=erasedups`. Wpisujesz `cd /var/log` dwadzieścia razy w ciągu dnia, przeplatane innymi komendami. Ile wystąpień `cd /var/log` zobaczysz w `history` na koniec dnia?

Odpowiedź

Jedno — to ostatnie. `erasedups` przy każdym nowym wpisie identycznym z jakimkolwiek wcześniejszym usuwa z historii wszystkie poprzednie wystąpienia tej komendy, zostawiając tylko najświeższe, na końcu listy. Różni się to od `ignoredups`, które porównuje tylko z komendą bezpośrednio poprzedzającą.

Powiązane: Zmienne środowiskowe i pliki startowe powłoki · Przekierowania i potoki · Uprawnienia do plików

Źródła: `man 1 bash` (sekcja HISTORY) · `help history` · materiały wewnętrzne — sekcja „Historia samych komend (bez daty i numeracji)”

Revision #15
Created 2026-09-18 06:57:17 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:57 UTC by Tomasz Konieczny