Prompt: PS1, kolory, kontekst

Poziom: podstawy **Dotyczy: bash, wszystkie dystrybucje** **Czas: ~16 min**

Prompt to zwykła zmienna tekstowa ze specjalną obróbką — bash podstawia w niej znaczniki na nazwę użytkownika, katalog, godzinę i kolor. Jeden brakujący nawias w tej zmiennej potrafi popsuć edycję linii poleceń na cały dzień, bez żadnego komunikatu błędu.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
ANSI	American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu
BHP	bezpieczeństwo i higiena pracy
PS1	zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy
UID	User ID: numer użytkownika w Linuksie

Fiszka

TYLKO ODCZYT

Sprawdzenie bieżącego promptu: `echo $PS1`

```
echo $PS1
```

`PS1` to zwykła zmienna powłoki — bash odczytuje ją tuż przed pokazaniem znaku zachęty i podstawia w niej specjalne sekwencje (patrz tabela niżej). `echo` wypisze ją w postaci surowej, jeszcze przed podstawieniem.

Przykład:

```
[\u@\h \W]\$
```

PS1 = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

TYLKO ODCZYT

Ustawienie podstawowego promptu: `PS1='\u@\h:\w\$ '`

```
PS1='\u@\h:\w\$ '
```

Zmiana obowiązuje od razu, w bieżącej powłoce, ale tylko do jej zamknięcia. `\u` to nazwa użytkownika, `\h` skrócona nazwa hosta, `\w` pełna ścieżka bieżącego katalogu, `\$` to `$` dla zwykłego użytkownika albo `#` dla roota.

Przykład:

```
adm_user@app01:/var/log$
```

PS1 = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

TYLKO ODCZYT

Sekwencje specjalne — ściąga

Prompt basha rozumie kilkanaście znaczników zaczynających się od `\`, z których każdy podstawia inny fragment bieżącego stanu sesji:

Sekwencja	Podstawia
`\u`	nazwa użytkownika
`\h`	nazwa hosta do pierwszej kropki
`\H`	pełna nazwa hosta (z domeną)
`\w`	pełna ścieżka katalogu roboczego (`$HOME` skrócone do `~`)
`\W`	tylko ostatni człon ścieżki (nazwa bieżącego katalogu)
`\d`	data w formacie „dzień-tygodnia miesiąc dzień"
`\t`	czas 24-godzinny `HH:MM:SS`
`\A`	czas 24-godzinny `HH:MM`, bez sekund

Sekwencja	Podstawia
`\n`	znak nowej linii
`\$`	`$` dla zwykłego użytkownika, `#` dla roota (UID 0)
`\!`	numer bieżącego polecenia w historii sesji
`\#`	numer polecenia liczony od startu tej powłoki

```
PS1='\u@\h [\d \A] \w\n\$ '
```

Przykład:

```
adm_user@app01 [pon lis 17 14:32] /etc/nginx
$
```

UID = User ID: numer użytkownika w Linuksie · **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

TYLKO ODCZYT

Kolorowanie promptu: `\[\e[0;32m\]` i reset `\[\e[0m\]`

```
PS1='\[\e[0;32m\]\u@\h\[\e[0m\]:\w\$ '
```

`\e[0;32m` to sekwencja sterująca ANSI, która każe terminalowi przełączyć kolor tekstu na zielony; `\e[0m` przywraca ustawienia domyślne. Bez tego resetu **cały dalszy tekst w terminalu** — łącznie z wynikiem kolejnych komend — zostałby zielony.

Przykład:

```
adm_user@app01:/var/log$
```

(fragment `adm_user@app01` wyświetla się na zielono, reszta linii w domyślnym kolorze terminala)

ANSI = American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu · **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

TYLKO ODCZYT

Tabela kodów kolorów ANSI

Kod	Kolor tekstu	Kod (jasny)	Kolor tekstu (jasny)
`\e[0;30m`	czarny	`\e[1;30m`	szary
`\e[0;31m`	czerwony	`\e[1;31m`	jasnoczerwony
`\e[0;32m`	zielony	`\e[1;32m`	jasnozielony
`\e[0;33m`	żółty	`\e[1;33m`	jasnożółty
`\e[0;34m`	niebieski	`\e[1;34m`	jasnoniebieski
`\e[0;35m`	fioletowy	`\e[1;35m`	jasnofioletowy
`\e[0;36m`	turkusowy	`\e[1;36m`	jasnoturkusowy
`\e[0;37m`	szary	`\e[1;37m`	biały
`\e[0m`	**reset** — wraca do ustawień domyślnych terminala		

Pierwsza cyfra (`0` albo `1`) to atrybut: `0` normalny, `1` pogrubiony/jasny. Druga liczba (`30`–`37`) to sam kolor. Ten sam schemat działa też jako `\033[...m` — `\033` i `\e` to dwa zapisy tego samego znaku ESC (kod 27), rozumiane identycznie przez basha w `PS1`.

□□ **ANSI** = American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu · **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

□□ **TYLKO ODCZYT**

Oznaczenie sekwencji niedrukowalnych: `\[` i `\]`

```
PS1='\[\e[0;32m\]\u\[\e[0m\]@\h:\w\$ '
```

Wszystko, co ląduje w prompcie, ale **nie zajmuje miejsca na ekranie** — jak sekwencje kolorów — musi być owinięte `\[` na początku i `\]` na końcu. To nie jest kosmetyka: bez tych nawiasów bash źle liczy, ile znaków prompt faktycznie zajmuje na ekranie, i to psuje edycję linii poleceń (pełne wyjaśnienie w *Jak to działa* i pierwszy punkt w *Częste błędy* — to najważniejsza rzecz na tej stronie).

□□ **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

□□ **ZMIENIA SYSTEM**

Ustawienie promptu kontynuacji: `PS2`

```
PS2='> '
```

Pokazywany, gdy wpisana linia jest niekompletna i bash czeka na jej dokończenie — niedomknięty cudzysłów, otwarty `if` bez `fi`, backslash na końcu linii.

Przykład:

```
$ echo "zaczynam linię
> i ją kończę"
zaczynam linię
i ją kończę
```

TYLKO ODCZYT

Ustawienie promptu debugowania: `PS4` (`set -x`)

```
PS4='+ ${BASH_SOURCE}:${LINENO}: '
bash -x skrypt.sh
```

`PS4` poprzedza każdą linię wypisywaną przez `set -x` (albo `bash -x`) — domyślnie to sam znak `+`. Podmiana na coś z nazwą pliku i numerem linii błyskawicznie mówi, **gdzie** w skrypcie jesteś, zamiast zgadywać po samej treści polecenia.

Przykład:

```
+ skrypt.sh:12: cp -v /etc/hosts /tmp/hosts.bak
```

bash = Bourne Again Shell: uruchom powłokę bash albo wskazany w niej skrypt

TYLKO ODCZYT

Wykonanie polecenia przed każdym promptem: `PROMPT_COMMAND`

```
PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD}\007"'
```

Zawartość tej zmiennej bash wykonuje **tuż przed** wyrysowaniem każdego promptu — zanim jeszcze zinterpretuje `PS1`. Powyższy przykład ustawia tytuł okna terminala na `użytkownik@host: katalog`, aktualizowany po każdej komendzie.

PS1 = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

◫◫ ZMIENIA SYSTEM

Gałąź git w prompcie

```
parse_git_branch() {
    git branch 2>/dev/null | sed -n '/^\*/s/^\* //p'
}
PS1='\u@\h:\w\[\e[0;33m\]$(parse_git_branch)\[\e[0m\]\$ '
```

Funkcja wypisuje nazwę bieżącej gałęzi git (albo nic, jeśli katalog nie jest repozytorium). `$(...)` w `PS1` **musi** być ustawione w pojedynczym cudzysłowie — patrz *Częste błędy*, dlaczego podwójny cudzysłów tu zepsuje wszystko.

Przykład:

```
adm_user@app01:/opt/app main$
```

◫◫ **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy · **git** = system kontroli wersji (nazwa to brytyjski slang: nieznośny typ)

◫◫ TYLKO ODCZYT

Ustawienie koloru bez pamiętania kodów ANSI: `tput setaf`

```
GREEN=$(tput setaf 2)
RESET=$(tput sgr0)
PS1="\[${GREEN}\]\u@\h\[${RESET}\]:\w\$ "
```

`tput` odpytuje bazę terminfo o właściwą sekwencję dla danego terminala zamiast zakładać na sztywno kod ANSI — przenośniejsze między różnymi emulatorami terminala. `setaf N` ustawia kolor tekstu (numer 0–7, jak w tabeli kolorów), `sgr0` to odpowiednik resetu `\e[0m`.

◫◫ **ANSI** = American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu · **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

◫◫ ZMIENIA SYSTEM

Ustawienie na stałe w `~/.bashrc`

```
cat >> ~/.bashrc <<'EOF'
PS1='\[\e[0;32m\]\u@\h\[\e[0m\]:\[\e[0;34m\]\w\[\e[0m\]\$ '
EOF
```

Prompt ustawiony bezpośrednio w terminalu ginie wraz z sesją. `~/.bashrc` jest czytany przy każdej nowej interaktywnej powłoce (mechanizm opisuje strona o plikach startowych), więc to standardowe miejsce na własny `PS1`.

PS1 = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Rozróżnienie kolorem promptu roota: `if [ $EUID -eq 0 ]`

```
if [ "$EUID" -eq 0 ]; then
    PS1='\[\e[1;31m\]\u@\h\[\e[0m\]:\w\# '
else
    PS1='\[\e[0;32m\]\u@\h\[\e[0m\]:\w\$ '
fi
```

`$EUID` to efektywny UID bieżącego procesu — `0` oznacza roota. Warunek pozwala trzymać jedną definicję `PS1` w jednym pliku, a mimo to dostać ostrzegawczy czerwony prompt tylko wtedy, gdy naprawdę działamy jako root.

UID = User ID: numer użytkownika w Linuksie · **PS1** = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy

ZMIENIA SYSTEM

Ustawienie promptu dla wszystkich kont: `PS1` w `/etc/profile.d/`

```
sudo tee /etc/profile.d/prompt-produkcja.sh > /dev/null <<'EOF'
if [ "$EUID" -eq 0 ]; then
    PS1='\[\e[1;37;41m\] PRODUKCJA \[\e[0m\] \u@\h:\w\# '
fi
EOF
```

Ustawienie promptu dla **wszystkich** kont na danej maszynie — np. czerwone tło i napis „PRODUKCJA" dla każdego, kto zaloguje się jako root na serwerze produkcyjnym, niezależnie

od tego, czyj prywatny `.bashrc` jest akurat aktywny. Plik musi kończyć się na `.sh`, żeby `/etc/profile` go w ogóle zauważyło (mechanizm opisuje strona o plikach startowych).

Uważaj

- Brak `\[` `\]` wokół sekwencji koloru psuje liczenie szerokości promptu przez bash — patrz *Częste błędy*, punkt pierwszy.
- `PS1` w podwójnym cudzysłowie z `$(polecenie)` w środku zamraża wynik na stałe, zamiast odświeżać go przy każdym prompcie.
- Zmiana `PS1` w terminalu obowiązuje tylko do jego zamknięcia — do trwałego efektu potrzebny jest plik startowy.

PS1 = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy · **tee** = zapisz strumień wejścia do pliku i jednocześnie wypisz go na ekran

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 10:00: Tomek koloruje sobie prompt na app01 kodem `\e[0;32m` bez nawiasów `\[` `\]`, zadowolony z efektu. Po południu wpisuje długą komendę z `find`, cofa się strzałką, żeby poprawić literówkę, a linia na ekranie zaczyna się rozjeżdżać i nadpisywać samą siebie. Dopiero owinięcie sekwencji koloru w `\[` i `\]` przywraca normalną edycję linii poleceń.

Po co to istnieje

Pojęcie: zmienna promptu. `PS1` to zwykła zmienna powłoki, ale z jedną szczególną właściwością: bash interpretuje jej zawartość na nowo przed każdym wyświetleniem znaku zachęty, podstawiając w niej sekwencje zaczynające się od `\` (nazwa użytkownika, katalog, godzina) i wykonując wszelkie `$(polecenie)`, jakie w niej znajdzie — o ile ustawiono ją w pojedynczym cudzysłowie. Nazwa pochodzi od *Prompt String 1*; `PS2`, `PS3` i `PS4` to jej rodzeństwo dla innych sytuacji (kontynuacja linii, menu `select`, debugowanie).

Firmowe notatki z poprzedniej pracy pokazują, jak w praktyce wygląda różnicowanie promptu między środowiskami — dokładnie ten sam mechanizm co w tej Fiszce, tylko dopracowany pod konkretne potrzeby zespołu:

```
# środowiska testowe, zwykły użytkownik
export
PS1="\[\033[00;36m\]\u\[\033[00;32m\]@\[\033[01;32m\]\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[01
;37m\]$\[\033[00m\]: "

# środowiska produkcyjne, zwykły użytkownik — host na czerwono
export
PS1="\[\033[00;36m\]\u\[\033[00;32m\]@\[\033[1;31m\]\h\[\033[00m\]:\[\033[1;31m\]\w\[\033[01;3
7m\]$\[\033[00m\]: "

# root, środowiska testowe — user na czerwono, reszta jak wyżej
export
PS1="\[\033[1;31m\]\u\[\033[1;37m\]@\[\033[01;32m\]\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[01;3
7m\]#\[\033[00m\]: "

# root, środowiska produkcyjne — całość na czerwono
export
PS1="\[\033[1;31m\]\u\[\033[1;37m\]@\[\033[1;31m\]\h\[\033[00m\]:\[\033[1;31m\]\w\[\033[01;37m
\]#\[\033[00m\]: "
```

Ten sam kolor da się też zapisać w formacie 24-bitowym (RGB), zamiast ośmiu podstawowych kolorów ANSI — przydatne, kiedy firma ma własny kolor firmowy i chce go odtworzyć dokładnie, a terminal wspiera tryb *true color*:

```
export PS1="\[\e[38;2;176;132;204m\]\u\[\e[0m\]@\[\e[38;2;176;132;204m\]\h:\w\[\e[0m\] \$ "
```

`\e[38;2;R;G;Bm` to rozszerzenie ANSI (nie każdy stary terminal je obsługuje) — `38` oznacza „ustaw kolor tekstu", `2` mówi „w trybie RGB", a trzy liczby po nim to składowe czerwona, zielona, niebieska w skali 0–255.

Analogia

Wyobraź sobie **kaski ochronne na hali produkcyjnej** — biały dla gościa, żółty dla operatora, czerwony dla kierownika zmiany i ratownika BHP.

Kolor kasku nie zmienia tego, kim dana osoba jest — to metadana doczepiona z zewnątrz, którą każdy odczytuje w ułamek sekundy, bez pytania i bez czytania identyfikatora na piersi. Widzisz czerwony kask kątem oka i wiesz: to ktoś, kogo decyzje trzeba respektować natychmiast, bez zastanawiania się, czy to na pewno on.

Kolorowy prompt działa identycznie. Zielony napis `adm_user@app01` na Twoim ekranie nie zmienia tego, na jakim serwerze jesteś — ale sprawia, że **zanim jeszcze przeczytasz nazwę hosta**, boczne pole widzenia rejestruje kolor i ostrzega: to nie jest ta sama maszyna, na której pracowałeś minutę temu. Dokładnie to samo rozumowanie stoi za czerwonym promptem na produkcji: chodzi o to, żeby pomyłka „myślałem, że jestem na teście" wymagała zignorowania jaskrawego koloru, a nie tylko przeoczenia napisu wtopionego w resztę tekstu.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: sekwencja sterująca ANSI.** Ciąg znaków zaczynający się od bajtu ESC (kod 27, zapisywany w bashu jako `\e` albo `\033`), po którym następują cyfry i litera określające, co ma zrobić terminal — zmienić kolor tekstu, przesunąć kursor, wyczyścić ekran. Terminal **interpretuje** te sekwencje i nigdy nie wyświetla ich jako widocznego tekstu; z punktu widzenia tego, co widzisz na ekranie, `\e[0;32m` zajmuje zero miejsca — ale z punktu widzenia basha, który tylko wysyła bajty, to wciąż pięć znaków w zmiennej `PS1`.

> **Pojęcie: znak niedrukowalny i liczenie szerokości.** Znak (albo sekwencja znaków) niedrukowalny to coś, co terminal przetwarza, ale nie odkłada na ekranie jako widoczny glif — sekwencja koloru ANSI jest tego klasycznym przykładem. Problem w tym, że biblioteka `readline`, która obsługuje edycję linii poleceń (strzałki, `Ctrl+A`, historia, zawijanie długiej komendy do kolejnej linii ekranu), musi wiedzieć, **ile kolumn terminala faktycznie zajmuje prompt**, żeby poprawnie liczyć, gdzie jest kursor. Domyślnie liczy to najprościej, jak można: zlicza wszystkie znaki w `PS1` po podstawieniu sekwencji `\u`, `\h` itd. Jeśli w tej zmiennej są bajty sekwencji ANSI, a nic nie mówi bashowi „te konkretne znaki nie liczą się do szerokości" — biblioteka policzy je tak, jakby faktycznie zajmowały miejsce na ekranie, i pomyli się o dokładnie tyle znaków, ile mają wszystkie użyte sekwencje kolorów razem wzięte.

Właśnie do tego służą `\[` i `\]`: to nie jest kolejna sekwencja koloru, tylko znacznik dla samego basha — „wszystko między tymi nawiasami pomiń przy liczeniu szerokości promptu". Kod koloru dalej trafia do terminala i dalej działa (terminal nie widzi `\[` ani `\]`, one są usuwane przed wysłaniem), ale readline wie, że ma je zignorować przy obliczeniach.

Skutek braku tych nawiasów ujawnia się nie od razu i nie przy krótkich komendach — dlatego ten błąd jest tak podstępny. Dopóki wpisywana linia mieści się w jednym rzędzie terminala, wszystko wygląda normalnie. Problem pojawia się, gdy linia polecenia jest **dłuższa niż**

szerokość terminala i zawija się do kolejnego wiersza, albo gdy używasz strzałek czy `Ctrl+R` (przeszukiwanie historii) do edycji: readline, mając błędne wyobrażenie o tym, gdzie faktycznie kończy się prompt, zaczyna nadpisywać fragment własnej linii zamiast poprawnie przesuwać kursor. Efekt widoczny na ekranie to poszarpany, nadpisujący się tekst, fragmenty starej komendy widoczne obok nowej, kursor migający w zupełnie nie tym miejscu, w którym powinien być — a to wszystko z pojedynczej pary brakujących nawiasów kwadratowych w definicji `PS1`.

Krok po kroku

1. Zobacz surową wartość PS1, zanim zaczniesz zmieniać.

```
echo $PS1
```

2. Zbuduj prompt bez kolorów, tylko z sekwencjami informacyjnymi.

```
PS1='\u@\h:\w\$ '
```

3. Dodaj kolor, pilnując nawiasów `\[` `\]` przy każdej sekwencji ANSI.

```
PS1='\[\e[0;32m\]\u@\h\[\e[0m\]:\[\e[0;34m\]\w\[\e[0m\]\$ '
```

4. Zweryfikuj, że edycja długiej linii działa poprawnie.

```
echo
aaaa
```

```
    PS1='\[\e[0;32m\]\u@\h\[\e[0m\]:\w\$ '
fi
EOF
source ~/.bashrc
```

Częste błędy

Brak `\[` `\]` — edycja długich komend rozjeżdża się na ekranie.

```
PS1="\e[0;32m\u@\h\e[0m:\w\$ "
```

Prompt wygląda normalnie, dopóki nie wpiszesz komendy dłuższej niż szerokość terminala albo nie użyjesz strzałek do edycji — wtedy kursor zaczyna lądować w złych miejscach, a fragmenty starego tekstu zostają na ekranie, nadpisane tylko częściowo. Bash policzył sekwencje `\e[0;32m` i `\e[0m` jako widoczne znaki, więc jego wyobrażenie o tym, ile kolumn zajmuje prompt, jest przesunięte o kilkanaście znaków względem rzeczywistości. Naprawa: owinąć każdą sekwencję ANSI w `\[` i `\]`.

```
PS1="\[\e[0;32m\]\u@\h\[\e[0m\]:\w\$ "
```

Kolor bez resetu — cały terminal zostaje pokolorowany.

```
PS1='\[\e[0;32m\]\u@\h:\w\$ '
```

Prompt jest zielony — i wynik `ls`, `cat`, każdej kolejnej komendy też, bo nic nie kazało terminalowi wrócić do domyślnego koloru. Naprawa: zawsze kończ sekwencją resetu.

```
PS1='\[\e[0;32m\]\u@\h\[\e[0m\]:\w\$ '
```

Podwójny cudzysłów zamraża wynik `$(polecenie)` na stałe.

```
PS1="\u@\h $(git branch --show-current)\$ "
```

Nazwa gałęzi w prompcie nigdy się nie zmienia, nawet po `git checkout inna-galaz` — bo w podwójnym cudzysłowie `$(...)` wykonuje się **raz**, w momencie przypisania do `PS1`, a wynik zostaje wklejony do zmiennej jako zwykły tekst. Naprawa: pojedynczy cudzysłów. Wtedy `$(...)` zostaje częścią wzorca `PS1` i bash wykonuje je na nowo przy każdym wyświetleniu promptu (dzięki domyślnie włączonej opcji `promptvars`).

```
PS1='\u@\h $(git branch --show-current)\$ '
```

Zaskoczenie innym promptem po `sudo -i` albo `su -`.

Administrator starannie koloruje `~/.bashrc` swojego konta, loguje się przez `sudo -i` i widzi zupełnie inny, nieustawiony prompt. Konto roota ma **własny** `/root/.bashrc`, niezależny od pliku zwykłego użytkownika — zmiana w jednym pliku nie dotyka drugiego. Naprawa: albo powielić ustawienia w `/root/.bashrc`, albo — lepiej — ustawić rozróżnienie kolorami przez `/etc/profile.d/`, które obejmuje każde konto na maszynie jednym plikiem (patrz Fiszka).

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie. Zanim zaczniesz edytować `~/.bashrc`, zrób kopię: `cp ~/.bashrc ~/.bashrc.bak`.

Ćwiczenie 1 — zobacz błąd na własne oczy

```
cd /tmp
OLD_PS1="$PS1"
PS1="\e[0;35m[TEST]\e[0m \w\$ "
```

Wpisz teraz dowolną komendę dłuższą niż szerokość Twojego terminala (np. `echo` z kilkuset znakami `x`), a potem pocofaj się strzałką w lewo o kilkanaście znaków i coś skasuj. Opisz, co się dzieje na ekranie. Napraw i porównaj.

Odpowiedź

Bez `\[` `\]` kursor po cofnięciu strzałką ląduje w złym miejscu, a skasowanie znaku potrafi zostawić na ekranie „widmo" poprzedniej wersji linii — bash źle liczy, gdzie kończy się niewidoczna część promptu. Naprawa:

```
PS1="\[\e[0;35m\][TEST]\[\e[0m\] \w\$ "
```

Po tej zmianie te same operacje (długa linia, strzałki, backspace) działają czysto. Przywróć oryginalny prompt: `PS1="$OLD_PS1"`.

Ćwiczenie 2 — gałąź git, która się nie odświeża

```
cd /tmp
git init test-repo
cd test-repo
git commit --allow-empty -m "start" -q
PS1="\w $(git branch --show-current)\$ "
```

Zapisz, jaką gałąź pokazuje prompt. Utwórz nową gałąź (`git checkout -b eksperyment`) i sprawdź, czy prompt się zmienił. Popraw definicję `PS1` tak, żeby się zmieniał.

Odpowiedź

Prompt dalej pokazuje starą gałąź (np. `main`), mimo przełączenia na `eksperyment` — bo `$(...)` w podwójnym cudzysłowie wykonał się raz, przy przypisaniu. Naprawa: pojedynczy cudzysłów.

```
PS1='\w $(git branch --show-current)\$ '
```

Teraz każde odświeżenie promptu (czyli każde naciśnięcie Enter) na nowo pyta gita o bieżącą gałąź.

Pytania kontrolne

1. Czym różni się rola `\[` `\]` od roli `\e[0m`?

Odpowiedź

`\e[0m` to prawdziwa sekwencja ANSI wysyłana do terminala — mówi mu „wyzeruj kolor i atrybuty tekstu". `\[` i `\]` to instrukcja wyłącznie dla samego basha (a ściślej dla readline) — mówią „nie licz tego, co jest w środku, do szerokości promptu na ekranie". Terminal nigdy nie widzi `\[` ani `\]` (bash je usuwa przed wysłaniem), a `\e[0m` nigdy nie trafia do readline jako informacja o szerokości — to dwa zupełnie osobne mechanizmy, które w praktyce zawsze występują razem przy kolorowaniu promptu.

2. Ustawiasz `PS1='\u@\h:\w\$ '` bezpośrednio w terminalu i widzisz efekt natychmiast. Dlaczego po zamknięciu terminala prompt wraca do poprzedniego?

Odpowiedź

Bo zmiana dotyczyła tylko zmiennej `PS1` w pamięci bieżącej powłoki — nic nie zapisało jej na dysku. Żeby prompt przetrwał kolejne logowanie, definicja musi trafić do pliku startowego

czytanego przy starcie powłoki, najczęściej `~/.bashrc` (dla pojedynczego użytkownika) albo pliku w `/etc/profile.d/` (dla wszystkich kont na maszynie).

3. Dlaczego prompt roota na serwerze produkcyjnym warto zrobić inaczej kolorowany niż prompt zwykłego użytkownika na tej samej maszynie?

Odpowiedź

Bo kolor działa jako sygnał zauważalny kątem oka, zanim jeszcze przeczytasz treść promptu — dokładnie tak jak kolor kasku ochronnego. Praca jako root na produkcji ma zupełnie inne konsekwencje pomyłki niż praca jako zwykły użytkownik na środowisku testowym; jaskrawy, konsekwentnie stosowany kolor (np. czerwone tło) zmniejsza ryzyko wykonania niebezpiecznej komendy „nie na tej maszynie", bo wymaga świadomego zignorowania sygnału, a nie tylko przeoczenia napisu.

4. Masz `PS4='+ '` (wartość domyślna) i debugujesz skrypt wywołujący dziesiątki funkcji. Co zmienisz w `PS4`, żeby `bash -x` było czytelniejsze, i dlaczego?

Odpowiedź

Warto dodać do `PS4` nazwę pliku i numer linii, np. `PS4='+ ${BASH_SOURCE}:${LINENO}: '`, żeby każda linia wypisywana przez `set -x` od razu mówiła, w którym pliku i w którym miejscu skryptu aktualnie się wykonuje — bez tego trzeba by zgadywać kontekst po samej treści polecenia, co przy dziesiątkach funkcji i zagnieżdżonych wywołaniach robi się szybko nieczytelne.

Powiązane: Pliki startowe powłoki: co, gdzie i kiedy się wczytuje · Aliasy i funkcje powłoki · Terminal, TTY i powłoka · Zmienne środowiskowe i PATH

Źródła: `man 1 bash` (sekcje PROMPTING, Shell Variables — `PS1`–`PS4`) · `man 5 terminfo` · `man 1 tput` · notatki własne — „Kolorowanie BASH prompt (zmiana PS1)"

Revision #15
Created 2026-09-18 06:57:15 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:55 UTC by Tomasz Konieczny