Szkielet skryptu produkcyjnego

Typ: referencja **Dotyczy: bash 4+/5+, wszystkie dystrybucje** **Czas: ~10 min**

Jeden kompletny szkielet skryptu produkcyjnego, gotowy do skopiowania i wypełnienia właściwą logiką w funkcji `main`. Zawiera wszystko z poprzednich stron tego rozdziału w jednym miejscu: tryb ścisły, `trap`, funkcje pomocnicze, parsowanie opcji i tryb `--dry-run`.

Skróty użyte na tej stronie (1)

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

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Poniedziałek, 8:00. Tomek zaczyna nowy skrypt porządkujący stare archiwa na bkp01, kopiując ten szkielet zamiast pisać od zera. `require_cmd rsync` na starcie `main` zatrzymuje go zaraz po uruchomieniu, bo `rsync` akurat nie jest zainstalowany na tym serwerze, zamiast wywalić się dopiero w połowie kopiowania z niejasnym `command not found`. Dziesięć minut na `dnf install rsync` i skrypt rusza dalej, tym razem do końca.

Szkielet do skopiowania

```
#!/bin/bash
#
# NazwaSkryptu.sh — jedno zdanie o tym, co ten skrypt robi.
#
# Autor:  Zespół Administracji Systemów
```

```
# Data:   2026-09-25
# Użycie: NazwaSkryptu.sh [-h] [-v] [-n] NAZWA_USLUGI
#
set -euo pipefail
IFS=$'\n\t'

readonly SKRYPT_NAZWA="$(basename "$0")"
readonly SKRYPT_KATALOG="$(cd "$(dirname "$0")" && pwd)"
readonly PLIK_LOG="/var/log/${SKRYPT_NAZWA%.sh}.log"

verbose=0
dry_run=0

usage() {
    cat <<EOF
Użycie: ${SKRYPT_NAZWA} [-h] [-v] [-n] NAZWA_USLUGI

  -h    wyświetla tę pomoc i kończy działanie
  -v    włącza szczegółowe komunikaty (verbose)
  -n    tryb dry-run – pokazuje, co zostałoby zrobione, bez wprowadzania zmian
EOF
}

log() {
    local znacznik_czasu
    znacznik_czasu="$(date '+%Y-%m-%d %H:%M:%S')"
    printf '[%s] %s\n' "$znacznik_czasu" "$1"
}

die() {
    echo "BŁĄD: $1" >&2
    exit "${2:-1}"
}

cleanup() {
    local kod_wyjscia=$?
    [[ -n "${katalog_tymczasowy:-}" ]] && rm -rf -- "$katalog_tymczasowy"
    exit "$kod_wyjscia"
}
```

```
require_root() {
    [[ $EUID -eq 0 ]] || die "ten skrypt wymaga uprawnień roota (uruchom przez sudo)" 1
}

require_cmd() {
    command -v "$1" &>/dev/null || die "wymagane narzędzie '$1' nie jest zainstalowane" 127
}

wykonaj() {
    if (( dry_run )); then
        echo "[dry-run] $*"
    else
        "$@"
    fi
}

main() {
    trap cleanup EXIT

    while getopts ":hvn" opcja; do
        case "$opcja" in
            h) usage; exit 0 ;;
            v) verbose=1 ;;
            n) dry_run=1 ;;
            \?) die "nieznana opcja: -${OPTARG}" 2 ;;
        esac
    done
    shift $((OPTIND - 1))

    [[ $# -eq 1 ]] || { usage >&2; die "wymagany dokładnie jeden argument: NAZWA_USLUGI" 2; }
    local nazwa_uslugi="$1"

    require_root
    require_cmd systemctl

    katalog_tymczasowy="$(mktemp -d)"

    (( verbose )) && log "katalog tymczasowy: ${katalog_tymczasowy}"
```

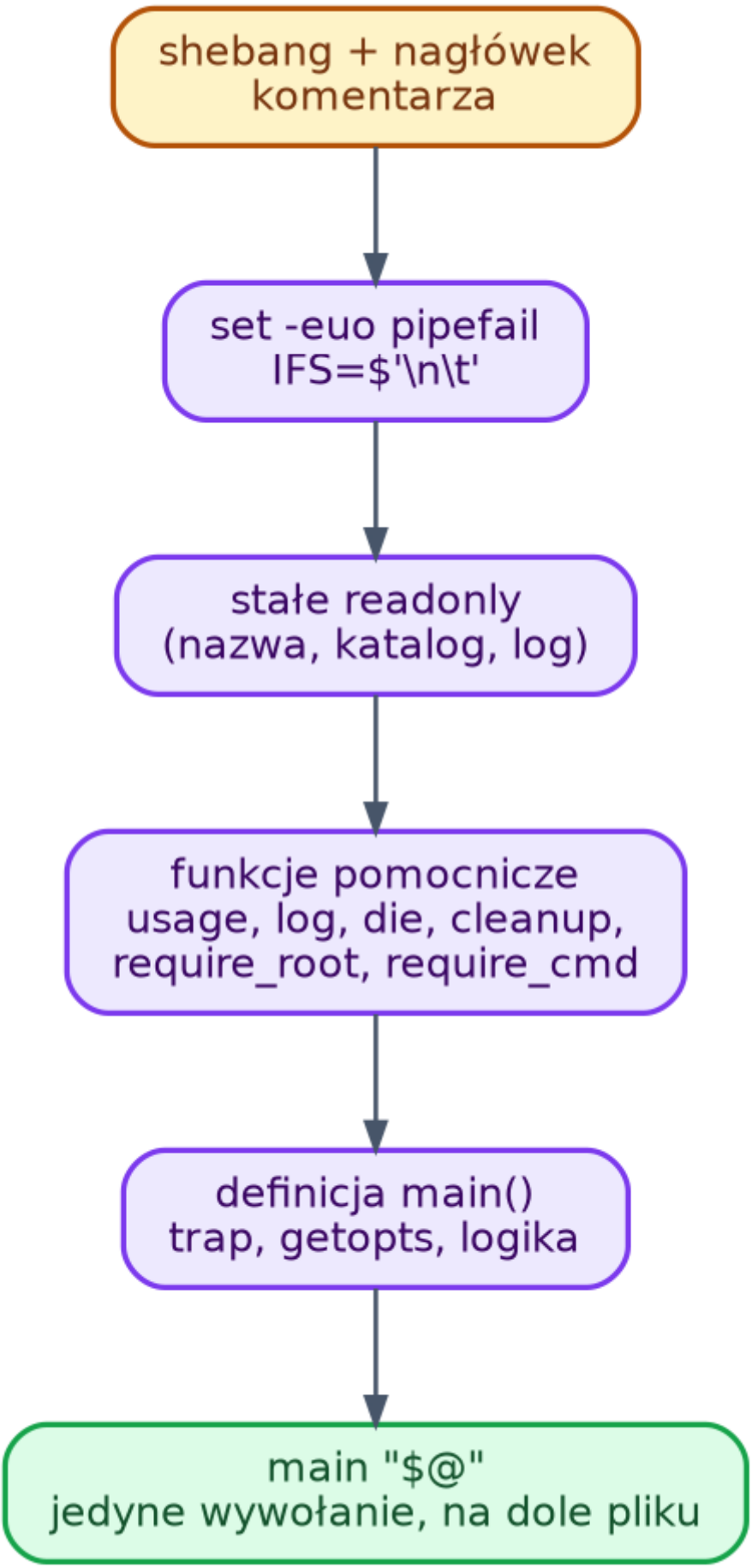
```
    log "sprawdzam usługę: ${nazwa_uslugi}"

    if systemctl is-active --quiet "$nazwa_uslugi"; then
        log "usługa ${nazwa_uslugi} już działa"
    else
        wykonaj systemctl enable --now "$nazwa_uslugi"
        log "usługa ${nazwa_uslugi} uruchomiona"
    fi

    systemctl is-active --quiet "$nazwa_uslugi" || die "usługa ${nazwa_uslugi} nie
wystartowała" 1
    log "stan docelowy osiągnięty"
}

main "$@"
```

Omówienie poszczególnych sekcji


shebang + nagłówek komentarza
set -euo pipefail IFS=$'\n\t'
stałe readonly (nazwa, katalog, log)
funkcje pomocnicze usage, log, die, cleanup, require_root, require_cmd
definicja main() trap, getopts, logika
main "$@" jedyne wywołanie, na dole pliku

Sześć warstw szkieletu od góry do dołu pliku: nagłówek, tryb ścisły, stałe, funkcje pomocnicze, definicja main i jej jedyne wywołanie na końcu. (kliknięcie otwiera pełny rozmiar)

Shebang i nagłówek komentarza

```
#!/bin/bash
#
# NazwaSkryptu.sh — jedno zdanie o tym, co ten skrypt robi.
#
# Autor:  Zespół Administracji Systemów
# Data:   2026-09-25
# Użycie: NazwaSkryptu.sh [-h] [-v] [-n] NAZWA_USLUGI
#
```

`#!/bin/bash` wybieramy zamiast `#!/usr/bin/env bash` na maszynach, o których wiemy, że mają bash pod tą stałą ścieżką — czyli praktycznie każdy serwer RHEL/Rocky/Debian/Ubuntu/SLES, zgodnie z rozstrzygnięciem na stronie Pierwszy skrypt: shebang, uprawnienia, uruchomienie. Nagłówek komentarza nie jest wykonywany przez powłokę, ale jest pierwszą rzeczą, jaką czyta człowiek otwierający plik — cel, autor, data ostatniej istotnej zmiany i sygnatura wywołania oszczędzają konieczność czytania całego kodu, żeby zrozumieć, do czego skrypt służy.

Tryb ścisły i separator słów

```
set -euo pipefail
IFS=$'\n\t'
```

Zestaw opisany w całości na stronie Tryb ścisły: set -euo pipefail — zatrzymanie przy błędzie, przy nieustawionej zmiennej, przy niepowodzeniu w środku potoku, oraz bezpieczniejszy separator podziału na słowa. Wklejamy to zawsze na samej górze, zaraz po nagłówku, zanim padnie jakakolwiek inna komenda.

Stałe tylko do odczytu

```
readonly SKRYPT_NAZWA="$(basename "$0")"
readonly SKRYPT_KATALOG="$(cd "$(dirname "$0")" && pwd)"
readonly PLIK_LOG="/var/log/${SKRYPT_NAZWA%.sh}.log"
```

`readonly` gwarantuje, że te trzy wartości nie zostaną przypadkiem nadpisane gdzieś w dalszej części skryptu — próba przypisania do nich kończy się natychmiastowym błędem powłoki, a nie

cichym, trudnym do wyśledzenia pominięciem. `SKRYPT_KATALOG` liczony przez `cd ... && pwd` zwraca zawsze pełną, kanoniczną ścieżkę katalogu skryptu, niezależnie od tego, czy uruchomiono go przez ścieżkę względną czy bezwzględną — przydatne, gdy skrypt ma wczytywać bibliotekę funkcji leżącą obok niego, jak w przykładzie na stronie Funkcje.

Funkcja usage

```
usage() {
    cat <<EOF
Użycie: ${SKRYPT_NAZWA} [-h] [-v] [-n] NAZWA_USLUGI
  ...
EOF
}
```

Jedno miejsce opisujące sposób wywołania skryptu, wywoływane zarówno przy fladze `-h`, jak i przy błędnym wywołaniu bez wymaganych argumentów — dzięki temu komunikat pomocy nigdy nie rozjeżdża się między dwoma kopiami tego samego tekstu.

Funkcja log

```
log() {
    local znacznik_czasu
    znacznik_czasu="$(date '+%Y-%m-%d %H:%M:%S')"
    printf '[%s] %s\n' "$znacznik_czasu" "$1"
}
```

Znacznik czasu przy każdej linii, zapisany zgodnie z zasadą ze strony Obsługa błędów, trap i sprzątanie: deklaracja `local` i przypisanie z podstawienia komendy rozdzielone na dwie linie, żeby kod wyjścia `date` nie został przypadkiem zamaskowany kodem wyjścia `local` (`SC2155` w ShellCheck).

Funkcja die

```
die() {
    echo "BŁĄD: $1" >&2
    exit "${2:-1}"
}
```

Jedno miejsce kończące skrypt z komunikatem na stderr i kontrolowanym kodem wyjścia — wywoływane przez `|| die "komunikat" kod` wszędzie tam, gdzie warunek wstępny nie jest spełniony.

Funkcja cleanup i trap

```
cleanup() {
    local kod_wyjscia=$?
    [[ -n "${katalog_tymczasowy:-}" ]] && rm -rf -- "$katalog_tymczasowy"
    exit "$kod_wyjscia"
}
```

Zapisujemy `$?` na samym początku funkcji, zanim jakakolwiek kolejna komenda go nadpisze — to jest kod, który spowodował wejście do pułapki, i chcemy go zachować jako ostateczny kod wyjścia skryptu. Sprawdzenie `[[ -n "${katalog_tymczasowy:-}" ]]` chroni przed błędem `unbound variable` pod `set -u`, gdyby `cleanup` uruchomiło się zanim `katalog_tymczasowy` w ogóle powstał — na przykład przy błędnym wywołaniu, które kończy się na etapie parsowania opcji. `trap cleanup EXIT` rejestrujemy na samym początku `main`, zanim cokolwiek innego się wydarzy.

Funkcje require_root i require_cmd

```
require_root() {
    [[ $EUID -eq 0 ]] || die "ten skrypt wymaga uprawnień roota (uruchom przez sudo)" 1
}

require_cmd() {
    command -v "$1" &>/dev/null || die "wymagane narzędzie '$1' nie jest zainstalowane" 127
}
```

Dwie najczęstsze walidacje wstępne w skryptach administracyjnych, wydzielone raz do funkcji zamiast powtarzane w każdym skrypcie od nowa. `require_cmd systemctl` na początku `main` sprawia, że skrypt kończy się czytelnym komunikatem, zanim dotrze do linii, która i tak zawiodłaby z mniej zrozumiałym błędem basha „command not found" gdzieś w środku logiki.

Parsowanie opcji: -h, -v, -n

```
while getopts ":hvn" opcja; do
    case "$opcja" in
        h) usage; exit 0 ;;
```

```
        v) verbose=1 ;;
        n) dry_run=1 ;;
        \?) die "nieznana opcja: -${OPTARG}" 2 ;;
    esac
done
shift $((OPTIND - 1))
```

Trzy standardowe flagi, które powtarzają się w niemal każdym skrypcie administracyjnym: `-h` do pomocy, `-v` do trybu gadatliwego, `-n` do trybu na sucho. Mechanizm `getopts` i `OPTIND` opisuje szczegółowo strona Zmienne, parametry i argumenty.

Funkcja main i wywołanie na końcu

```
main() {
    ...
}

main "$@"
```

Cała właściwa logika skryptu mieszka w `main`, a jedyne wywołanie na poziomie samego pliku to `main "$@"` w ostatniej linii — konwencja opisana na stronie Funkcje. Dzięki temu skrypt można wczytać przez `source` w teście `bats` i wywołać pojedyncze funkcje pomocnicze bez uruchamiania całego `main`, a struktura pliku czyta się jak spis treści: najpierw definicje, na końcu jedno wywołanie.

Skrócona wersja dla krótkich skryptów jednorazowych

Pełny szkielet ma sens dla skryptu, który trafi do repozytorium i będzie uruchamiany wielokrotnie przez różne osoby. Dla jednorazowego skryptu pomocniczego — na przykład szybkiego sprzątania podczas incydentu — poniższy szkielet zachowuje najważniejsze zabezpieczenia bez narzutu pełnej struktury:

```
#!/bin/bash
# Cel: krótki opis w jednym zdaniu.
set -euo pipefail
```

```
die() {
    echo "BŁĄD: $1" >&2
    exit 1
}

[[ $# -eq 1 ]] || die "użycie: $0 ARGUMENT"

katalog_tymczasowy=$(mktemp -d)
trap 'rm -rf -- "$katalog_tymczasowy"' EXIT

echo "pracuję z argumentem: $1, katalog roboczy: $katalog_tymczasowy"
```

Zachowujemy tryb ścisły, `die` do czytelnych błędów i `trap` do sprzątania — trzy elementy, których pominięcie najczęściej boli, nawet w skrypcie napisanym „na chwilę". Rezygnujemy z `getopts`, z osobnych funkcji `usage`/`log`/`require_root` i z podziału na `main` — dla dwudziestu linijek kodu ta struktura byłaby czystym narzutem, a nie ułatwieniem.

> **Uważaj**
>
> - Skrócona wersja nadal zaczyna się od `set -euo pipefail` i `trap ... EXIT` — to jest minimum, poniżej którego nie schodzimy nawet w skrypcie jednorazowym, bo koszt jego dodania jest znikomy, a koszt jego braku bywa katastrofalny, jak pokazuje przykład na stronie Tryb ścisły: set -euo pipefail.
> - Skrypt „jednorazowy" bardzo często przeżywa dłużej niż planowano — jeśli w ciągu tygodnia uruchamiamy go po raz trzeci, to sygnał, żeby przenieść go do pełnego szkieletu.

Powiązane: Pierwszy skrypt: shebang, uprawnienia, uruchomienie · Tryb ścisły: set -euo pipefail · Obsługa błędów, trap i sprzątanie · Funkcje · Zmienne, parametry i argumenty · ShellCheck i testowanie skryptów · Idempotentność: skrypt, który wolno uruchomić dwa razy

Źródła: `man 1 bash` · `help getopts` · `help trap` · dokumentacja ShellCheck (wiki, sekcja *Style guide*)

Revision #11
Created 2026-09-25 00:40:56 UTC by Tomasz Konieczny
Updated 2026-10-03 19:10:27 UTC by Tomasz Konieczny