terminal — pod-skora-systemu — 80×24

```
M.ZdrowY@book : ~ $ cat ~/ksiazka.txt
```

POD SKÓRĄ SYSTEMU

przewodnik po terminalu dla Windows i macOS

```
~/ksiazka$
ls -la
total 144
drwxr-xr-x
drwxr-xr-x
-rw-r--r--  1 441 chapters/
-rw-r--r--  1 2K cmd-powershell.md
-rw-r--r--  1 4.2K siec.md
-rw-r--r--  1 2.1K skrypty.md
-rw-r--r--  1 8.1K winget.md
-rw-r--r--  1 3.3K ssh.md
-rw-r--r--  1 6.1K git.md
```

Pod skórą systemu

Pod skórą systemu

Przewodnik po terminalu dla Windows i macOS

M.ZdrowY

mzdrowy@gmail.com

2026

Pod skórą systemu

Przewodnik po terminalu dla Windows i macOS

Autor: M.ZdrowY

mzdrowy@gmail.com

Spis treści

Prolog — Skąd się wziął ten czarny ekran

W 1969 roku Ken Thompson siedział w Bell Labs i chciał zagrać w grę. Grę kosmiczną — Space Travel. Problem był prozaiczny: komputer, na którym działała, kosztował około 75 dolarów za sesję. Thompson znalazł w kącie stary PDP-7. Nikt go nie używał. Nikt za niego nie płacił. Przeniósł grę. Przy okazji napisał system operacyjny.

Ten system operacyjny nazywał się Unix.

Nie ma tu morału o tym, że wielkie rzeczy rodzą się z małych. Chodzi o coś innego: Unix od pierwszego dnia był narzędziem dla kogoś, kto chce coś zrobić — nie dla kogoś, kto chce, żeby coś samo się zrobiło. Wszystko było tekstem. Wszystkim sterowałeś przez klawiaturę. Nie dlatego, że tak było ładniej — dlatego, że tak było szybciej i dlatego, że w 1969 roku nie było innego wyjścia.

Minęło ponad pół wieku. Masz myszkę, ekran dotykowy, głosowego asystenta, który rozumie połowę tego, co mówisz. I nadal jest czarny prostokąt z migającym kursorem.

Dlaczego?

Bo część rzeczy nie ma przycisku. Bo jak musisz zrobić coś sto razy — robisz skrypt i komputer robi to za ciebie. Bo jak coś nie działa i nie wiesz, co — terminal ci powie, a okienko z komunikatem błędu powie „coś poszło nie tak" i zaoferuje przycisk OK.

To nie jest narzędzie dla hakerów z filmów. To narzędzie dla ludzi, którym skończyła się cierpliwość.

Windows i ten czarny ekran

Microsoft przez długi czas miał inne zdanie na temat terminala. Strategia była prosta: schować wszystko pod przyciski, żeby każdy mógł obsłużyć drukarkę. CMD — wiersz poleceń, który znasz

z filmów, gdzie ktoś szybko coś wpisuje — wywodzi się z MS-DOS z 1981 roku, choć sam CMD.EXE to nowa implementacja z Windows NT (1993). Microsoft mógł go wyrzucić. Nie wyrzucił — miliony skryptów w firmach na całym świecie przestałyby działać z dnia na dzień.

PowerShell pojawił się w 2006 roku jako odpowiedź na pytanie, którego Microsoft długo unikał: co jeśli terminal miałby działać sensownie? Spóźniona odpowiedź — Linux zdobywał dominację na serwerach od lat, administratorzy narzekali od lat — ale lepsza późno niż wcale.

macOS i ten sam czarny ekran

Mac ma inne korzenie. W 2001 roku Apple wypuściło Mac OS X zbudowany na fundamencie, który Steve Jobs przywiózł z firmy NeXT — a NeXT był oparty na BSD, czyli krewnym Unixa. Pod błyszczącą powierzchnią z Dockiem i animowanym Geniem siedzi system, który pamięta Bell Labs.

Dlatego kiedy otworzysz Terminal na Macu i wpiszesz `ls` — zobaczysz listę plików. Tak samo jak na Linuksie. Tak samo jak na serwerze w centrum danych po drugiej stronie świata. Pół wieku tej samej komendy.

O czym jest ta książka

O tym, żebyś umiał posługiwać się terminalem na Windows i na macOS. Nie żebyś rozumiał, jak działa jądro systemu — żebyś umiał zrobić to, co chcesz zrobić, wiedział, czemu to działa, i nie bał się czarnego ekranu.

Każdy temat ma dwa poziomy. Pierwszy — krótka odpowiedź, wystarczy, żeby działać. Drugi — jeśli chcesz wiedzieć dlaczego. Możesz czytać tylko pierwszy poziom i będzie dobrze. Możesz czytać oba. Możesz wrócić do drugiego za tydzień.

Zaczynamy od Windows. Jeśli masz Maca — przeskocz do części drugiej. Wszystko jest tam.

CMD i PowerShell — dwie powłoki, jedna decyzja

Windows ma dwie **powłoki** — dwa programy, które przyjmują twoje polecenia i uruchamiają inne programy. Pierwsza wywodzi się z **MS-DOS** z 1981 roku. Druga zadebiutowała w 2006. Obie robią to samo — każda na swój sposób. Pytanie brzmi: której użyć?

Czym się różni CMD od PowerShell

CMD to potomek **MS-DOS**, systemu operacyjnego z ekranem tekstowym i żadną myszą. Gdy w 1981 roku IBM wypuścił pierwszy PC, nie miał Windows ani ikon — miał czarny ekran, migający kursor i interpreter **COMMAND.COM**. Sam CMD.EXE to nowszy program: zadebiutował w 1993 roku razem z Windows NT 3.1. Składniowo jest zgodny z COMMAND.COM, więc stare skrypty z 1981 działają bez zmian. Czyta twój tekst, wykonuje program, wypluwa tekst. Tyle.

PowerShell to zupełnie nowa powłoka od Microsoftu. Nie produkuje tekstu — produkuje obiekty. Jeśli poprosisz o listę plików, dostajesz nie string do parsowania, ale gotowe obiekty z właściwościami: nazwa, rozmiar, data, uprawnienia. Możesz je od razu filtrować, sortować, zapisywać.

> Analogia? CMD jest jak kalkulator. Wpisz liczby, dostaniesz wynik. PowerShell jest jak Excel. Dostajesz arkusz z danymi, możesz go ciąć, sortować, sumować — bez pisania formuł do parsowania.

> Windows nie zaczynał jako niezależny system. Pierwsze wersje były tylko graficzną nakładką na MS-DOS — tekstową powłokę z 1981 roku.
> To oznaczało, że Windows od początku musiał żyć z przeszłością, której nie dało się po prostu usunąć. Dlatego do dziś w systemie istnieją CMD i PowerShell obok siebie. Dlatego stare skrypty w firmach wciąż działają — od 1993 roku do dziś.
> W świecie Windows obowiązuje niepisana zasada: jeśli coś działa w firmach, nie wolno tego zepsuć.

Kiedy użyć którego

CMD jest do starych skryptów. Jeśli plik **.bat** z 1995 roku wciąż działa — nie ruszaj go. CMD przydaje się też w sytuacjach, gdzie PowerShell nie jest dostępny: niektóre instalatory Windows, narzędzia recovery, WinRE. I czasem zwykłe `dir` i `cd` jest szybsze niż kombinowanie z obiektami.

PowerShell do wszystkiego nowego. Instalacja roli Windows, zarządzanie usługami, automatyzacja, API, pliki JSON, rejestr — PowerShell ma do tego natywne polecenia. Od aktualizacji Windows 10 Creators Update (v1703, kwiecień 2017) PowerShell zastąpił CMD w menu Win+X. W Windows 11 nowa aplikacja Terminal uruchamia PowerShell w domyślnej zakładce. CMD dostajesz na dokładkę — otworzysz go z listy rozwijanej.

Jak uruchomić

W Windows 11 można uruchomić powłokę na kilka sposobów.

Najszybszy: naciśnij **Win + R**, wpisz `cmd` albo `powershell`, wciśnij Enter.

Drugi: otwórz Start, zacznij pisać "cmd" albo "powershell", kliknij.

Trzeci: kliknij prawym przyciskiem przycisk Start → **Terminal**. To nowa aplikacja, która łączy cmd, powershell, a nawet wiersz Linuksa w zakładkach. Domyślnie uruchamia PowerShell.

Admin vs zwykły. Wiele poleceń wymaga praw administratora. W systemach Windows 10/11 uruchom program jako administrator: kliknij prawym przyciskiem → "Uruchom jako administrator". Zobaczysz komunikat UAC — zaakceptuj. Bez tego część komend zwróci "Access denied".

Podstawowe komendy — obok siebie

Tabela najczęściej używanych komend:

Operacja	CMD	PowerShell
Wyświetl pliki	dir	ls (alias) lub Get-ChildItem
Zmień katalog	cd folder	cd folder
Wyczyść ekran	cls	cls lub clear
Wypisz tekst	echo "tekst"	echo "tekst" lub Write-Output
Szukaj pomocy	help komenda	Get-Help komenda
Kopiuj plik	copy a b	Copy-Item a b
Usuń plik	del plik	Remove-Item plik
Sprawdź procesy	tasklist	Get-Process

Widać wzór. W CMD wszystko jest krótkie, stare, na skróty. W PowerShell komendy są długie, czytelne, w formacie Czasownik-Rzeczownik (`Get-ChildItem`, `Remove-Item`, `Copy-Item`). Jeśli znasz angielski, domyślasz się co robią.

Wiele skrótów z CMD działa też w PowerShell. `cd`, `cls`, `dir` — wszystkie przepisane jako aliasy, żeby start był płynniejszy.

Różnica filozofii — tekst vs obiekty

To najważniejsza różnica między CMD a PowerShell. W CMD wszystko jest tekstem. Wynik polecenia to ciąg znaków. Chcesz znaleźć pliki .txt? Odpalisz dir i przepuścisz przez find:

```
dir | find ".txt"
```

Działa. Jeśli chcesz przeszukać cały dysk C w poszukiwaniu starych plików tymczasowych:

```
dir C:\*.tmp /s | find ".tmp"
```

A co z plikami większymi niż 1 MB posortowanymi według rozmiaru? W CMD robi się to masakrycznie. Trzeba parsować kolumny, obcinać spacje, sortować po odpowiednim polu. Każda wersja językowa Windows zwraca inny format daty.

PowerShell podaje ci listę obiektów. Każdy plik ma właściwości: `Name`, `Length`, `LastWriteTime`, `Extension`. Możesz je filtrować po wartości:

```
Get-ChildItem | Where-Object { $_.Extension -eq ".txt" }
```

To samo, ale nie parshujesz stringa. Sprawdzasz właściwość `Extension`. Kod działa tak samo na polskim, angielskim i japońskim Windows.

A posortowane pliki > 1 MB? W PowerShell wystarczy dodać warunek i sortowanie. Krok po kroku:

```
Get-ChildItem
# wypluwa liste plikow jak dir
```

Krok 1:

Get-ChildItem pobiera liste plików

```
Get-ChildItem | Where-Object { $_.Length -gt 1MB }
# filtruje: zostawia tylko pliki > 1 MB
```

Krok 2:

Where-Object odcina małe pliki

```
Get-ChildItem | Where-Object { $_.Length -gt 1MB } | Sort-Object Length
# sortuje od najmniejszego
```

Krok 3:

Sort-Object porzadkuje wyniki

To jest obiektowy pipeline. Każda komenda produkuje obiekty, kolejna je filtruje, następna sortuje. Nie ma parsowania.

Linux mówi: wszystko jest plikiem i tekstem.

PowerShell mówi: wszystko jest obiektem i strukturą danych.

To nie różnica kosmetyczna. To dwa sposoby myślenia. Linux — składasz rzeczy jak klocki LEGO. PowerShell — operujesz na gotowych modelach danych. Efekt jest podobny, ale filozofia — zupełnie inna.

Skąd w ogóle PowerShell

Pod koniec lat 90. Microsoft miał poważny problem. Linux miał basha. Windows miał CMD — który składniowo wywodził się w prostej linii z MS-DOS/COMMAND.COM z 1981 roku. Administratorzy Linuksa mogli skryptować praktycznie wszystko. Administratorzy Windows klepali kreatory i guziki.

Microsoft nie zamierzał usuwać CMD — do dziś pozostaje on integralną częścią systemu. Próbował za to załatać jego ograniczenia dodatkami: Windows Script Host, VBScript, WMI. To były plastry na dziurawą instalację. Zmieniała się widoczność CMD w interfejsie użytkownika (menu Win+X, menu kontekstowe), ale sam program nigdy nie był zagrożony usunięciem.

W 2006 roku Jeffrey Snover zaproponował coś radykalnego: powłokę, która nie operuje na tekście, ale na obiektach .NET. Pomysł był tak nowy, że początkowo odrzucono go jako zbyt odległy od tradycji Unixa. Snover nie odpuścił. Napisał dokument "Monad Manifesto", w którym wyłożył filozofię obiektowego shella. Microsoft dał mu zielone światło.

Tak powstał PowerShell — początkowo jako "Monad", potem "Windows PowerShell", w końcu "PowerShell" (wersja open source od 2016 roku).

Przez lata Microsoft wierzył, że przyszłość to GUI: kliknięcia, kreatory, okienka. Terminal traktowano jako techniczne zaplecze — coś dla administratorów, nie dla zwykłych użytkowników.

Problem pojawił się, gdy firmy zaczęły zarządzać tysiącami komputerów naraz. Klikanie przestało mieć sens. I wtedy padło pytanie, którego Microsoft długo unikał: jak zarządzać systemem bez myszy?

PowerShell był odpowiedzią — nie jako ewolucja CMD, ale jako całkowite przepisanie od zera. To nie była modernizacja. To była korekta błędnego założenia sprzed dekad.

Aliasy — dlaczego ls nie jest prawdziwe

W PowerShell wpisz `ls`. Działa. Wpisz `Get-ChildItem` — też działa. To samo.

`ls` to alias. Microsoft dodał go, żeby ludzie przychodzący z Unixa nie czuli się zagubieni. Podobnie `cat` (alias `Get-Content`), `mv` (`Move-Item`), `rm` (`Remove-Item`).

Sprawdź to sam. Otwórz PowerShell i wpisz:

```
Get-Alias ls
```

Zobaczysz: `ls -> Get-ChildItem`. Każdy alias ma swój prawdziwy odpowiednik. Kiedy nauczysz się formy `Czasownik-Rzeczownik`, odblokowujesz całą moc PowerShell — bo `Get-` to tylko jeden z czasowników. Są też `Set-`, `New-`, `Remove-`, `Invoke-`, `Test-`, `ConvertTo-` — setki.

CMD nadal żyje

CMD nie zniknie. Windows 11 ma nowy Terminal, który ma dedykowaną zakładkę dla CMD. Powód: miliony skryptów `.bat` i `.cmd` w firmach na całym świecie. Banki, szpitale, urzędy — tam wciąż leci legacy code z lat 90.

Nowy kod powstaje w PowerShell. Stary kod zostaje w spokoju. CMD jest od szybkich sprawdzeń — adres IP, kopiowanie pliku, odpalenie skryptu, który działa od 20 lat. Do wszystkiego innego — PowerShell.

Jak znaleźć odpowiedź samemu

Nie trzeba znać wszystkich komend. Trzeba wiedzieć, jak je znaleźć, gdy są potrzebne. Oto trzy metody, które zostają na lata.

Get-Help — wbudowana dokumentacja PowerShell

W PowerShell wpisz:

```
Get-Help Get-Process
```

Zobaczysz opis komendy, składnię, parametry. To odpowiednik manuala (man) z Linuksa — tylko że PowerShell generuje go na podstawie kodu, więc zawsze jest aktualny.

Chcesz zobaczyć przykłady? Dodaj flagę -Examples:

```
Get-Help Get-Process -Examples
```

PowerShell wypisze konkretne scenariusze: jak wyświetlić procesy, jak posortować po pamięci, jak zapisać do pliku. To najszybszy sposób nauki — kopiujesz przykład i zmieniasz parametry.

Szukaj po słowie kluczowym:

```
Get-Help *process*
```

Znajdziesz wszystkie komendy z "process" w nazwie: `Get-Process`, `Stop-Process`, `Debug-Process` i inne.

help w CMD

CMD ma własny system pomocy. Wpisz:

```
help
```

Zobaczysz listę komend systemowych. Aby poznać konkretną:

```
help dir
```

Możesz też użyć flagi **/?** — działa przy każdej komendzie:

```
dir /?
```

To szybsze niż help dir i działa nawet gdy help nie jest dostępny (np. w WinRE).

Get-Command — znajdź właściwą komendę

Nie pamiętasz dokładnej nazwy? Get-Command szuka po czasowniku lub rzeczowniku:

```
Get-Command -Verb Get
# wszystkie komendy zaczynające się od Get-
```

```
Get-Command -Noun Process
# wszystkie komendy kończące się na Process
```

Połącz oba:

```
Get-Command -Verb Get -Noun Process
# dokładnie: Get-Process
```

Analogia: Get-Help to instrukcja obsługi konkretnego urządzenia. Get-Command to spis treści całego sklepu. Używasz Get-Command gdy wiesz mniej więcej czego szukasz, ale nie znasz dokładnej nazwy.

Sprawdź to sam

1. Sprawdź wersję PowerShell. Otwórz PowerShell (Win + R → `powershell`) i wpisz `$PSVersionTable.PSVersion`. Zobaczysz numer wersji, np. 7.4.0.
2. Wylistuj pliki z podkatalogami. W PowerShell wpisz `Get-ChildItem -Recurse`. Zobaczysz wszystkie pliki i foldery w bieżącym katalogu i poniżej.
3. Filtruj wyniki. W PowerShell wpisz `Get-ChildItem | Where-Object { $_.Length -gt 1MB }`. To pokaże tylko pliki większe niż 1 MB. Zmień `-gt` na `-lt` i zobaczysz odwrotność.
4. Zobacz różnicę: w CMD wpisz `dir`, a potem w PowerShell wpisz `Get-ChildItem | Select-Object Name, Length, LastWriteTime | ConvertTo-Html`. Oba wyświetlają pliki, ale drugi produkuje stronę HTML. W CMD nie zrobisz tego bez zewnętrznego narzędzia.

Gdzie jesteś i jak się poruszać

Terminal zawsze działa w jakimś katalogu. Powłoka śledzi bieżący katalog roboczy — domyślnie to katalog domowy. Każde polecenie wykonuje się w tym katalogu. Dlatego trzeba wiedzieć, gdzie się jest i jak przejść gdzie indziej. Tej wiedzy wystarczy na 90% pracy w terminalu.

pwd — pokaż, gdzie jesteś

Wpisz `pwd` (ang. *print working directory*) i naciśnij Enter. Terminal wypisze ścieżkę do bieżącego katalogu.

Windows (PowerShell):

```
PS C:\Users\Jan> pwd
C:\Users\Jan
```

Windows (CMD):

```
C:\Users\Jan> cd
C:\Users\Jan
```

macOS:

```
$ pwd
/Users/jan
```

W CMD zamiast `pwd` wpisujesz `cd` bez argumentu — działa tak samo.

Litera `C:` w Windows oznacza pierwszy dysk twardy. Na macOS i Linuxie nie ma liter — wszystko zaczyna się od `/` (korzeń).

cd — zmień katalog

`cd` (ang. *change directory*) przenosi do innego katalogu. Działa we wszystkich powłokach tak samo.

Windows:

```
cd Dokumenty
```

macOS:

```
cd Documents
```

W obu przypadkach przejście prowadzi do katalogu Documents/Dokumenty. Różnica w nazwie to lokalizacja — polski Windows tłumaczy nazwy folderów.

Wykonaj, a potem wpisz `pwd` — zobaczysz, że jesteś w `C:\Users\Jan\Dokumenty` (Windows) lub `/Users/jan/Dokumenty` (macOS).

Specjalne lokalizacje:

Polecenie	Gdzie trafiasz
cd ..	Katalog nadrzędny (wyżej)
cd \ (CMD) / cd / (PS)	Korzeń dysku
cd ~	Katalog domowy (PS, macOS)
cd %USERPROFILE%	Katalog domowy (CMD)
cd C:\ (CMD/PS)	Korzeń dysku C:

Analogia: ścieżka absolutna to adres zamieszkania: „ul. Marszałkowska 1, 00-001 Warszawa". Ścieżka względna to „drzwi obok". Jedno jest jednoznaczne, drugie zależy od tego, gdzie stoisz.

ls / dir — co jest w bieżącym katalogu

Windows CMD:

```
dir
```

Windows PowerShell:

```
ls
```

Ewentualnie `Get-ChildItem` — pełna nazwa.

macOS:

```
ls
```

Opcje:

- `ls -la` (macOS, Linux) — szczegółowa lista z ukrytymi plikami
- `dir /w` (CMD) — szeroka lista
- `ls -Force` (PowerShell) — pokaż ukryte pliki

Mapa katalogów — Windows

Katalog	Zawartość	Uwaga
C:\Windows	Pliki systemowe Windows	Nie ruszaj ręcznie
C:\Windows\System32	Narzędzia systemowe (cmd.exe, ping, ipconfig)	Tutaj mieszka twój terminal
C:\Program Files	Zainstalowane programy 64-bit	Nie kasuj ręcznie
C:\Program Files (x86)	Programy 32-bit	Rzadziej używany
C:\Users	Profile użytkowników	Katalogi domowe
C:\Users\twojanazwa\Documents	Moje dokumenty	Tu trzymasz pliki
C:\Users\twojanazwa\Desktop	Pulpit	To co widzisz na ekranie
C:\Users\twojanazwa\Downloads	Pobrane pliki	Tu ląduje wszystko z internetu
C:\Users\twojanazwa\AppData	Dane aplikacji	Ukryty — konfiguracja programów

Mapa katalogów — macOS

Katalog	Zawartość	Uwaga
/Applications	Zainstalowane programy	Przeciągnij aby zainstalować

Katalog	Zawartość	Uwaga
/Users	Profile użytkowników	Odpowiednik C:\Users
/Users/twojanazwa/Documents	Dokumenty	Odpowiednik Moje dokumenty
/Users/twojanazwa/Desktop	Pulpit	To co widzisz na ekranie
/Users/twojanazwa/Downloads	Pobrane	Tu lądują pliki z Safari
/System	Pliki systemowe macOS	Chronione przez SIP
/Library	Biblioteki systemowe	Rzadko zaglądasz
/tmp	Pliki tymczasowe	Czyści się przy restarcie
/Volumes	Zamontowane dyski	Pendrive pojawi się tutaj

tree — zobacz całe drzewo

W Windows wpisz `tree`. Terminal wyświetli całe drzewo katalogów od bieżącego miejsca. Przydaje się do zrozumienia struktury.

Na macOS nie ma `tree`. Możesz użyć:

```
find . -type d
```

To wyświetli wszystkie katalogi poniżej bieżącego.

Skąd litery dysków w Windows?

W MS-DOS (1981) stacje dyskietek dostawały litery A: i B:. Pierwszy dysk twardy dostawał C:. Dziś nikt nie używa dyskietek, ale system wciąż rezerwuje A: i B:, a dysk twardy zaczyna się od C:. To relikt, którego Microsoft nie może usunąć bez łamania starych programów.

Na Macu i Linuxie nie ma liter — wszystko jest w jednym drzewie zaczynającym się od `/`. Dysk twardy jest zamontowany w `/`. Dodatkowe dyski pojawiają się w `/Volumes/` (macOS) lub `/mnt/` (Linux). Nie ma czytelniejszego rozwiązania.

Tylda (~) i zmienne środowiskowe

~ (tylda) to skrót na katalog domowy. Pochodzi z Unixa z lat 70. W PowerShell i na macOS działa bez problemu:

```
PS> cd ~
$ cd ~
```

W CMD nie ma tyldy. Zamiast tego używasz zmiennej środowiskowej:

```
C:\> cd %USERPROFILE%
```

Zmienne środowiskowe to nazwy w `%...%` (CMD) lub `$env:...` (PowerShell), które przechowują ścieżki i ustawienia systemu. Najprzydatniejsze:

Zmienna	CMD	PowerShell	Znaczenie
USERPROFILE	%USERPROFILE%	$env:USERPROFILE	Katalog domowy
TEMP	%TEMP%	$env:TEMP	Pliki tymczasowe
PATH	%PATH%	$env:PATH	Ścieżki do programów
COMPUTERNAME	%COMPUTERNAME%	$env:COMPUTERNAME	Nazwa komputera

Na macOS nie ma zmiennych środowiskowych dla katalogu domowego? Są: `$HOME` to to samo co ~. Wpisz `echo $HOME` w terminalu na Macu — zobaczysz `/Users/twojanazwa`.

Znajdź zamiast tree

`tree` to wygoda. Jeśli go nie masz (macOS), użyj `find`:

```
find . -type d
```

To wyświetli strukturę katalogów. Więcej o `find` w dalszych rozdziałach.

Dokańczanie tabulatorem

Wystarczy zacząć pisać ścieżkę i nacisnąć **Tab**. Terminal sam dokończy.

Przykład: wpisz `cd Desk` i naciśnij **Tab**. Jeśli masz folder `Desktop` (lub `Pulpit` na polskim Windows), terminal uzupełni resztę.

```
PS> cd Desk[TAB]
PS> cd Desktop\
```

Działa we wszystkich powłokach: CMD, PowerShell, bash, zsh. Jeśli jest kilka pasujących opcji, naciśnij **Tab** dwa razy — zobaczysz listę.

Historia poleceń

Naciśnij ↑ (strzałka w górę). Terminal wkleja poprzednią komendę. Naciśnij jeszcze raz — jeszcze wcześniejszą. ↓ cofa do nowszych.

W PowerShell możesz też podejrzeć całą historię:

```
Get-History
```

Zobaczysz listę z numerami. Uruchom ponownie konkretną komendę:

```
Invoke-History 3
```

W CMD odpowiednikiem jest `doskey /history`. Na Macu `history`.

```
$ history
 1  pwd
 2  cd ~
 3  ls -la
```

Ctrl+C i Ctrl+Z — zatrzymaj, co zacząłeś

Czasem polecenie działa za długo. Albo w ogóle nie chciałeś go uruchamiać (hello, `ping` na Macu, który pinguje w nieskończoność). Zatrzymujesz je przez **Ctrl + C**.

```
$ ping google.com
64 bytes from 142.250.184.78: icmp_seq=0 ttl=117 time=10.3 ms
64 bytes from 142.250.184.78: icmp_seq=1 ttl=117 time=9.8 ms
^C
$
```

Ctrl + C wysyła sygnał SIGINT (interrupt) do procesu. Większość programów kończy się grzecznie.

Ctrl + Z zawiesza proces, nie kończy go. Proces czeka w tle. Możesz go wznowić przez `fg` (foreground, na pierwszy plan) lub `bg` (background, niech działa w tle). Przydaje się, gdy chcesz na chwilę wyjść z edytora `nano` lub `vim` bez zamykania go.

```
$ nano notatka.txt
^Z
[1]+  Zawieszony  nano notatka.txt
$ fg
# wracasz do nano
```

Ctrl + D kończy strumień wejściowy. W pustym terminalu zamyka sesję. W środku `cat` bez argumentu kończy wprowadzanie tekstu.

Sprawdź to sam

5. Sprawdź gdzie jesteś. Otwórz terminal (PowerShell na Windows, Terminal na Macu) i wpisz `pwd`. Zobaczysz ścieżkę — np. `C:\Users\twojanazwa` lub `/Users/twojanazwa`.
6. Przejdź do katalogu domowego. Wpisz `cd ~` (lub w CMD `cd %USERPROFILE%`). Potem `pwd` — potwierdź że jesteś w domu.
7. Wylistuj zawartość. W PowerShell wpisz `ls`, w CMD `dir`, na Macu `ls -la`. Zobaczysz co masz w bieżącym katalogu.
8. Wróć do poprzedniego katalogu. Przejdź gdzieś najpierw: `cd Documents` (lub `cd Dokumenty` na polskim Windows). Potem wpisz `cd -` — wracasz tam, gdzie byłeś przed chwilą. Zobacz efekt przez `pwd`.

Jak nie zepsuć systemu

W 2020 roku administrator siedział przed terminalem w sobotni wieczór. Chciał wyczyścić stary katalog na serwerze. Wpisał sudo rm -rf /var/www/old — ale autouzupełnienie dodało spację przed ostatnim członem. Komenda poszła na cały /var/www. System obsługiwał trzydzieści stron internetowych.

Trwało to trzy sekundy.

Ta historia powtarza się w różnych wariantach — na Linuksie, macOS i Windows. Różne polecenia, ten sam mechanizm: jedna nieprzeczytana komenda, jeden enter, zero możliwości cofnięcia.

Terminal nie ostrzega. Nie pyta „jesteś pewien?". Wykonuje.

Pięć typowych pułapek. Każda wygląda niewinnie.

POZIOM 1

sudo — klucz do wszystkiego (i dlaczego nie nosi się go w kieszeni)

sudo to skrót od **superuser do** — wykonaj jako superużytkownik. Na Linuksie i macOS otwiera drzwi do absolutnie wszystkiego w systemie.

Wyobraź sobie budynek z setką zamkniętych pomieszczeń. Portiernia ma klucz główny do każdego z nich. Nikt rozsądny nie nosi go cały dzień przy sobie — bierze z szuflady, otwiera konkretne drzwi, oddaje.

sudo działa tak samo. Bez niego możesz oglądać pliki, uruchamiać programy, przeglądać logi. Z sudo — możesz wszystko. Usunąć system, zmienić hasła, wyłączyć firewall.

Na Windows odpowiednikiem jest „Uruchom jako administrator". Ten sam mechanizm, inna nazwa.

```
sudo rm -rf /
```

Samo sudo nie jest niebezpieczne. Niebezpieczne jest to, co po nim wpiszesz. Przed enterem przeczytaj całe polecenie jeszcze raz. Zwłaszcza fragment po sudo.

rm -rf — armagedon w 8 znakach

rm -rf to najkrótsza droga do zniszczenia systemu.

rm — remove, usuwa plik. **-r** — recursive, wchodzi do podfolderów. **-f** — force, nie pyta o potwierdzenie.

Niebezpieczne warianty:

- rm -rf / — usuwa cały system od korzenia
- rm -rf ~ — usuwa katalog domowy (wszystkie pliki użytkownika)
- rm -rf . — usuwa bieżący katalog
- rm -rf * — usuwa wszystko w bieżącym katalogu

Na Windows bezpieczniejszy odpowiednik:

```
Remove-Item -Path "C:\temp" -Recurse -Confirm:$true
```

Flaga -Confirm:$true wymusza potwierdzenie przed każdą operacją. Na macOS zamiast rm można użyć **trash** — przenosi pliki do kosza zamiast usuwać na stałe.

Zanim naciśniesz enter:

- Sprawdź, w którym jesteś katalogu — **pwd** (print working directory) pokazuje ścieżkę
- Jeśli komenda obsługuje **--dry-run**, uruchom ją w trybie symulacji — pokaże co zrobi bez faktycznego działania

Skrypty z internetu — zaufanie ma cenę

Na forum ktoś pisze: „Mam rozwiązanie twojego problemu, wklej to":

```
curl -sSL https://przyklad.pl/install.sh | bash
```

To polecenie robi dwie rzeczy: pobiera skrypt z internetu i od razu go wykonuje. Jeśli dodałeś sudo na początku — działa jako root. Nie widzisz, co skrypt robi. Może instalować program. Może też wysłać wszystkie hasła na serwer.

Bezpieczna wersja: pobierz, przeczytaj, wykonaj:

```
curl -sSL https://przyklad.pl/install.sh > skrypt.sh
less skrypt.sh
bash skrypt.sh
```

Analogia: ktoś na ulicy daje tabletkę i mówi „łyknij, pomaga". Nie łykasz bez sprawdzenia składu. Tutaj działa to samo.

PowerShell jako Administrator — broń, nie zabawka

Na Windows wiele poradników zaczyna się od: „Uruchom PowerShell jako administrator". Problem: jako administrator masz dostęp do rejestru, usług, plików systemowych. Wystarczy jedna pomyłka:

```
Remove-Item -Recurse C:\Windows\System32
```

System przestaje istnieć.

Nie każde polecenie wymaga administratora. Zwykłe operacje — przeglądanie plików, ping, ipconfig, winget search — działają bez podwyższonych uprawnień. Instalacja oprogramowania, zmiana ustawień systemowych, modyfikacja rejestru — tak, tu administrator jest potrzebny.

PowerShell ma wbudowane zabezpieczenie przed skryptami z internetu — **ExecutionPolicy**. Domyślnie blokuje uruchamianie niepodpisanych skryptów. Nie wyłączaj tego bez zrozumienia, co robisz.

Kopiowanie komend z AI — myślenie nie jest opcjonalne

Coraz więcej osób kopiuje komendy z czatów AI — ChatGPT, Claude, Gemini. Generatory wypluwają kod, który wygląda poprawnie. Nikt nie gwarantuje, że jest bezpieczny.

AI może wygenerować komendę, która:

- używa złej ścieżki — rm -rf /home zamiast /home/user/temp
- instaluje nie to oprogramowanie, co trzeba
- zawiera opcje, których nie rozumiesz — a one zmieniają znaczenie

Przed wklejeniem odpowiedz sobie na trzy pytania:

- Co robi to polecenie? Przeczytaj każdy człon od lewej do prawej.
- Czy znasz każdą flagę? -rf, -force, --delete-all — wiesz co oznaczają?
- Czy jesteś we właściwym katalogu? Sprawdź pwd przed uruchomieniem.

Jeśli komenda obsługuje --dry-run, użyj go. Testuj na maszynie wirtualnej albo w WSL, zanim uruchomisz na prawdziwym systemie.

POZIOM 2

Jak działa sudo w środku

sudo nie jest magiczne. To program, który sprawdza plik /etc/sudoers. W nim zapisane są reguły: kto może uruchomić jakie polecenie jako kto.

Plik sudoers ma własną składnię. Przykład:

```
root    ALL=(ALL:ALL) ALL
%admin  ALL=(ALL) ALL
janek   ALL=(ALL) NOPASSWD: /usr/bin/apt
```

Pierwsza linia: root może wszystko. Druga: grupa admin może wszystko, ale pyta o hasło. Trzecia: użytkownik janek może uruchomić apt bez hasła.

Czasowa ważność sudo — na Linuksie domyślnie 15 minut (timestamp_timeout=15). Na macOS domyślnie 0 — pyta za każdym razem. W obu systemach po przekroczeniu timeoutu system ponownie poprosi o hasło. To celowe: zmniejsza ryzyko, że ktoś inny wpisze komendę na pozostawionym terminalu.

Na Windows odpowiednikiem jest UAC (User Account Control). Różnica: UAC pyta za każdym razem, sudo na Linuksie zapamiętuje na 15 minut. Oba rozwiązania mają tę samą zasadę — nie chodź cały dzień z kluczem głównym w kieszeni.

ExecutionPolicy — jak PowerShell chroni przed skryptami

PowerShell ExecutionPolicy ma cztery główne wartości używane w praktyce:

Polityka	Co robi
Restricted	Żaden skrypt nie uruchomi się. Tylko pojedyncze polecenia. Domyślna na Windows Client.
RemoteSigned	Skrypty lokalne — tak. Skrypty z internetu — tylko podpisane cyfrowo.
AllSigned	Wszystkie skrypty muszą być podpisane — nawet te napisane na komputerze.
Bypass	Nic nie jest blokowane. Używana przez instalatory i skrypty uruchamiane programowo.

Sprawdź aktualną politykę:

```
Get-ExecutionPolicy
```

Zmiana dla bieżącego użytkownika:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Bez -Scope CurrentUser zmiana dotyczy całej maszyny i wymaga administratora. W praktyce **RemoteSigned** to rozsądny kompromis: własne skrypty działają, zdalne wymagają podpisu.

Dlaczego curl | bash jest ryzykowny

curl -sSL https://... | bash to łańcuch przekierowań. curl pobiera dane i wysyła je na standardowe wyjście. Rura (|) przekazuje je do bash, który wykonuje jako skrypt.

Problem: brak kontroli nad tym, co bash dostaje. Strona może zwrócić inny skrypt w zależności od dnia, godziny, adresu IP lub przeglądarki. To samo polecenie dziś instaluje program, jutro może wykraść dane.

Nawet zaufane serwisy bywają kompromitowane. W 2019 roku atakujący podmienił skrypt instalacyjny popularnego narzędzia na serwerze CDN — przez kilka godzin każdy, kto uruchomił curl | bash, instalował złośliwe oprogramowanie.

Słowo na koniec

Jedna nieprzeczytana komenda może zniszczyć system. Ale nie bój się terminala — bój się entera bez namysłu.

Każda z tych pięciu pułapek ma to samo antidotum: zatrzymaj się na sekundę przed enterem. Przeczytaj polecenie. Sprawdź katalog. Jeśli czegoś nie rozumiesz — nie wykonuj.

Terminal nie wybacza. Ale nagradza tych, którzy myślą.

Zanim Windows się obudzi — terminal, zanim system wstanie

W 1995 roku wkładało się dyskietkę z MS-DOS, pisało fdisk, potem format c: /s, potem sys c:. Trzy komendy i system stał. Trzy dekady później wkłada się pendrive, wciska Shift+F10 i wpisuje te same trzy komendy — tylko inaczej się nazywają.

Ekran jest czarny. System nie wstaje. Jedyna rzecz, która działa — klawiatura. W tym momencie zaczyna się prawdziwa diagnostyka, a nie ta z okienek z komunikatem "coś poszło nie tak". **Windows Recovery Environment** (WinRE) to tryb ratunkowy, który działa, zanim system się uruchomi. Na Macu — **macOS Recovery**. Obydwa dają dostęp do terminala w momencie, gdy system nie może wystartować — gdy nie ma już czego klikać.

POZIOM 1

Wejście do WinRE — trzy drogi

Do WinRE wchodzi się na trzy sposoby, w zależności od tego, w jakim stanie jest system.

Pierwszy — z bootowalnego pendrive'a instalacyjnego. Po uruchomieniu komputera z USB pojawia się ekran wyboru języka i przycisk "Zainstaluj teraz". Na tym ekranie naciska się Shift+F10. Otwiera się wiersz poleceń z uprawnieniami SYSTEM — wyższymi niż administrator w zwykłym Windows.

Drugi — z działającego Windows. W Windows 10: Settings → Update & Security → Recovery → Advanced startup → Restart now. W Windows 11: Settings → System → Recovery → Advanced

startup → Restart now. Komputer restartuje się do niebieskiego menu. Tam: Troubleshoot → Advanced options → Command Prompt.

Trzeci — automatycznie. Po trzech nieudanych próbach uruchomienia systemu Windows sam proponuje przejście do opcji naprawczych. Wybór: Advanced options → Command Prompt.

diskpart — czysty dysk od zera

Do czyszczenia dysku przed reinstalacją lub zmianą schematu partycji potrzebne jest **diskpart**. Diskpart działa w WinRE i pozwala całkowicie wyczyścić dysk ze wszystkich partycji, łącznie z tymi ukrytymi (systemową EFI, MSR, recovery).

Sekwencja dla czystej instalacji na UEFI:

```
diskpart
list disk
select disk 0
clean
convert gpt
exit
```

Przed clean — sprawdzić przez `list disk`, który dysk ma być czyszczony. Na maszynie z wieloma dyskami zdarza się wybrać niewłaściwy. **clean** usuwa wszystkie partycje bez ostrzeżenia. Dane nie znikają fizycznie — znika tablica partycji. Odtworzenie danych jest możliwe, ale wymaga specjalistycznego oprogramowania.

Po clean i convert gpt zamyka się okno cmd i wraca się do instalatora. Windows Setup sam tworzy partycje: EFI (100 MB FAT32), MSR (16 MB) i główną (reszta).

DISM — wypakowanie systemu

DISM (Deployment Image Servicing and Management) służy do obrazowania systemu. W WinRE jego główne zastosowanie to wypakowanie obrazu Windows na docelowy dysk bez używania standardowego instalatora.

Przed /Apply-Image trzeba przygotować dysk — utworzyć partycje. Dla UEFI potrzebne są partycja EFI (FAT32, 100 MB) i partycja główna (NTFS):

```
diskpart
select disk 0
clean
convert gpt
create partition efi size=100
```

```
format fs=fat32 quick
assign letter=S
create partition primary
format fs=ntfs quick
assign letter=C
exit
```

Składnia DISM:

```
DISM /Apply-Image /ImageFile:D:\sources\install.wim /Index:1 /ApplyDir:C:\
```

D: to pendrive z instalacją, **C:** to docelowy dysk. Litery w WinRE mogą różnić się od tych w normalnym systemie. Na pendrive może to być E: lub F:. Żeby sprawdzić, które litery mają dyski, używa się `diskpart → list volume`. Plik z obrazem systemu w katalogu sources może nazywać się `install.wim` lub `install.esd` — DISM działa tak samo na obu. Przed użyciem sprawdza się nazwę przez `dir D:\sources\install.*`.

/Index:1 wybiera wersję systemu z obrazu. W pliku install.wim (lub install.esd) może być kilka indeksów — mapowanie numerów na edycje różni się między obrazem konsumenckim a biznesowym. Przed użyciem DISM sprawdza się indeksy:

```
DISM /Get-WimInfo /WimFile:D:\sources\install.wim
```

Wynik pokazuje listę edycji z przypisanymi indeksami. Instalator Windows normalnie pyta o wybór wersji — DISM wymaga podania numeru.

Po zastosowaniu obrazu system jest wypakowany na C:, ale nie ma bootloadera. Do tego służy bcdboot.

bcdboot — budzik dla bootloadera

bcdboot kopiuje bootloadery z wypakowanego systemu Windows na partycję EFI i tworzy wpis w **BCD** (Boot Configuration Data). Dzięki niemu komputer wie, gdzie system jest zainstalowany.

Przed uruchomieniem bcdboot trzeba zamontować partycję EFI przez diskpart:

```
diskpart
select disk 0
list partition
select partition 1
assign letter=S
exit
```

Po zamontowaniu EFI jako S: uruchamia się bcdboot:

```
bcdboot C:\Windows /s S: /f UEFI
```

C:\Windows — katalog z zainstalowanym systemem. **/s S:** — partycja EFI. **/f UEFI** — typ firmware. Dla UEFI. Dla Legacy BIOS: /f BIOS.

bootrec — gdy system nie wstaje

bootrec to zestaw poleceń do naprawy rozruchu. Działa w WinRE i naprawia uszkodzone sektory bootu, MBR i BCD.

Trzy główne opcje:

```
bootrec /fixmbr
bootrec /fixboot
bootrec /rebuildbcd
```

/fixmbr nadpisuje Master Boot Record. Działa tylko w systemach z Legacy BIOS. W UEFI MBR nie jest używany — polecenie nic nie robi.

/fixboot zapisuje nowy sektor rozruchowy. Na UEFI może zwrócić `access denied` — oznacza to, że partycja EFI jest zamontowana z błędnymi uprawnieniami. Rozwiązanie: zamontować EFI przez diskpart z nową literą i spróbować ponownie.

/rebuildbcd skanuje wszystkie dyski w poszukiwaniu instalacji Windows i przebudowuje BCD. Zanim się go uruchomi, trzeba sprawdzić, czy partycja EFI jest zamontowana — bez tego bootrec może nie znaleźć żadnej instalacji i zwrócić `Total identified Windows installations: 0`. Montuje się ją przez diskpart: `select disk 0 → list partition → select partition 1 → assign letter=S`. Jeśli bootrec nie znajdzie instalacji, /rebuildbcd nie pomoże — wtedy bcdboot jest jedynym rozwiązaniem.

chkdsk i sfc w trybie offline

chkdsk sprawdza dysk pod kątem błędów systemu plików. W WinRE służy do naprawy uszkodzeń, które mogą uniemożliwiać uruchomienie systemu.

```
chkdsk C: /f
```

/f naprawia błędy logiczne systemu plików — metadane NTFS, indeksy, opóźnione znaczniki czasu. Nie skanuje powierzchni dysku w poszukiwaniu fizycznie uszkodzonych sektorów. Do skanowania sektorów potrzebny jest parametr `/r`, który łączy naprawę błędów logicznych z odczytem każdego sektora i oznaczeniem uszkodzonych jako złe.

sfc (System File Checker) w trybie offline skanuje pliki systemowe i zastępuje uszkodzone kopią z lokalnego cache:

```
sfc /scannow /offbootdir=C:\ /offwindir=C:\Windows
```

/offbootdir wskazuje partycję systemową, **/offwindir** — katalog Windows. Te parametry są potrzebne, bo WinRE nie ma zamontowanego bieżącego systemu.

macOS Recovery — Intel i Apple Silicon

macOS ma odpowiednik WinRE — **macOS Recovery**. Na Intel wchodzi się przez Cmd+R przy starcie. Klawisze trzeba przytrzymać, aż pojawi się Apple logo i pasek postępu pod spodem.

Na Apple Silicon (M1/M2/M3/M4) wchodzi się inaczej: przytrzymać przycisk zasilania, aż pojawi się napis Loading startup options. Kliknąć Options, potem Continue, wybrać konto administratora i wpisać hasło.

W obu przypadkach po załadowaniu Recovery pojawia się okno z opcjami. W górnym menu: Utilities → Terminal.

Terminal w Recovery na Macu — co można zrobić

Terminal w macOS Recovery ma ograniczony zestaw poleceń, ale kilka z nich ratuje sytuację:

resetpassword — jedno słowo, bez parametrów. Otwiera się interfejs do resetowania hasła. Wybiera się dysk, konto i wpisuje nowe hasło. Działa nawet na zaszyfrowanym dysku FileVault, o ile zna się hasło do konta Apple ID.

Lista dysków i partycji:

```
diskutil list
```

Wynik pokazuje wszystkie fizyczne dyski, partycje i woluminy. Na Apple Silicon dysk wewnętrzny to /dev/disk0. Partycja systemowa (/) i partycja danych (Data) są osobne — to standard APFS.

System Integrity Protection (SIP) — mechanizm ochrony jądra. W Recovery można go wyłączyć lub włączyć:

```
csrutil status
csrutil disable
csrutil enable
```

Wyłączenie SIP obniża bezpieczeństwo systemu. Przydaje się do modyfikacji chronionych katalogów albo instalacji rozszerzeń jądra. Po naprawie warto włączyć SIP z powrotem.

POZIOM 2

Jak działa bootowanie Windows

Łańcuch uruchamiania Windows na UEFI wygląda tak:

Procesor uruchamia firmware UEFI. Firmware sprawdza Secure Boot — jeśli jest włączony, weryfikuje podpis bootloadera. Uruchamia bootmgfw.efi z partycji EFI (FAT32, oznaczona jako "EFI System Partition"). Bootloader odczytuje BCD (Boot Configuration Data) — plik z konfiguracją bootu, który mówi, gdzie szukać systemu. Na podstawie BCD bootloader uruchamia winload.exe — właściwy kernel Windows.

BCD to nie jest zwykły plik tekstowy. To magazyn danych w formacie binarnym, podobny do rejestru. Przechowuje wpisy dla każdej zainstalowanej kopii Windows. Każdy wpis ma identyfikator GUID. Przykład:

```
bcdedit /enum
```

Wynik pokazuje listę wpisów. Domyślny wpis oznaczony jest **{default}**. Każdy zawiera ścieżkę do pliku systemu (osdevice), partycję (device) i opcje startu.

Gdy system nie znajduje bootloadera na partycji EFI, pojawia się czarny ekran z kursorem. Wtedy potrzebne jest bcdboot, które odtwarza całą strukturę od zera.

DISM — więcej niż wypakowanie

DISM to nie tylko /Apply-Image. W przedsiębiorstwach używa się go do całego cyklu życia obrazów systemowych:

- /Capture-Image — tworzy obraz (WIM) z istniejącej instalacji Windows. Przydaje się do przygotowania obrazu referencyjnego z oprogramowaniem
- /Add-Driver — dodaje sterowniki do obrazu przed wdrożeniem. Oszczędza czas, bo nie trzeba instalować sterowników po wdrożeniu
- /Enable-Feature — włącza funkcje Windows (Hyper-V, Containers, WSL) w obrazie offline
- /Get-CurrentEdition — pokazuje aktualną edycję systemu
- /Set-Edition — zmienia edycję (Home → Pro) bez reinstalacji

W normalnym systemie DISM działa też online — naprawia komponenty bez restartu:

```
DISM /Online /Cleanup-Image /RestoreHealth
```

To polecenie skanuje repozytorium komponentów (Component Store, C:\Windows\WinSxS) i naprawia uszkodzenia, ściągając zdrowe pliki z Windows Update. To pierwszy krok przed sfc /scannow, jeśli system jeszcze uruchamia się normalnie.

BCD i bcdedit

BCD to magazyn danych w formacie podobnym do rejestru Windows. Plik znajduje się na partycji EFI: EFI\Microsoft\Boot\BCD. Nie otwiera się go w Notatniku — tylko przez bcdedit.

Przydatne operacje bcdedit:

Polecenie	Efekt
bcdedit /enum	Wyświetla wszystkie wpisy w BCD
bcdedit /export C:\bcdbackup	Tworzy kopię zapasową BCD przed modyfikacją
bcdedit /set {default} recoveryenabled Yes	Włącza opcję odzyskiwania dla domyślnego wpisu
bcdedit /timeout 30	Ustawia czas oczekiwania na wybór systemu przy starcie
bcdedit /delete {GUID}	Usuwa wpis z BCD (wymaga znajomości GUID)

Opcja /set zmienia pojedynczy parametr. GUID w nawiasach klamrowych identyfikuje wpis. `{default}` to alias bieżącego domyślnego systemu. `{current}` to system, z którego uruchomiono bcdedit.

macOS Recovery — głębiej

macOS Recovery istnieje w dwóch wariantach. Recovery HD to lokalna partycja (około 650 MB), zawierająca podstawowe opcje: Terminal, Disk Utility, reinstalację systemu. Partycja jest tworzona automatycznie podczas instalacji macOS.

Internet Recovery (Option+Cmd+R na Intel) ładuje obraz recovery przez sieć. Działa, gdy lokalna partycja recovery jest uszkodzona albo gdy dysk został całkowicie wyczyszczony. Wymaga stabilnego połączenia z internetem — pobiera około 2 GB danych.

Na Apple Silicon recovery nie jest wbudowane w firmware — iBoot jest w pamięci ROM, ale samo środowisko odzyskiwania ładuje się z dedykowanego woluminu APFS na wewnętrznym SSD (Apple_APFS_Recovery). To podobna koncepcja co partycja recovery na Intel, ale szybsza, bo SSD jest bezpośrednio na płycie i komunikuje się przez NVMe bez dodatkowych pośredników.

Praktyczne scenariusze

Scenariusz: reinstalacja Windows z pendrive (szybka ścieżka)

Komputer ma uszkodzoną instalację Windows. Reinstalacja jest szybsza niż szukanie przyczyny:

Krok 1 — komputer uruchamia się z pendrive'a instalacyjnego (zmiana boot order w BIOS/UEFI przez F2, F12, Del lub Esc).

Krok 2 — na ekranie "Zainstaluj teraz" naciska się Shift+F10. Otwiera się wiersz poleceń.

Krok 3 — czyszczenie dysku:

```
diskpart
list disk
select disk 0
clean
convert gpt
exit
```

Krok 4 — zamyka się okno cmd i kontynuuje instalację. Windows Setup widzi czysty dysk i sam tworzy partycje.

Scenariusz: naprawa bootu

System pokazuje czarny ekran z kursorem lub "Bootmgr is missing". Przyczyna: uszkodzony BCD, brak bootloadera na partycji EFI.

Krok 1 — uruchamia się WinRE (Shift+F10 z USB lub z menu Advanced startup).

Krok 2 — sprawdza się, czy partycja EFI jest widoczna:

```
diskpart
list volume
```

Szuka się woluminu o systemie plików FAT32 i rozmiarze ~100 MB. Jeśli nie ma litery, przypisuje się: `select volume X → assign letter=S`.

Krok 3 — uruchamia się naprawę:

```
bootrec /rebuildbcd
bcdboot C:\Windows /s S: /f UEFI
```

Jeśli bootrec nie znajdzie instalacji, /rebuildbcd nie pomoże. Wtedy bcdboot jest jedynym rozwiązaniem — tworzy bootloader od nowa.

Scenariusz: Mac — reset hasła przez Recovery

Hasło do konta macOS przepadło. Bez niego nie ma dostępu do plików i systemu. Recovery pozwala je zmienić:

Krok 1 — komputer uruchamia się w Recovery (Intel: Cmd+R, Apple Silicon: przytrzymanie przycisku zasilania → Options).

Krok 2 — otwiera się Terminal z menu Utilities.

Krok 3 — wpisuje się jedno słowo:

```
resetpassword
```

Otwiera się okno Reset Password. Wybiera się dysk z systemem (macOS Base System lub Macintosh HD), konto użytkownika, wpisuje nowe hasło i potwierdza.

Jeśli dysk ma włączony FileVault, system poprosi o hasło odzyskiwania (recovery key) lub zalogowanie przez Apple ID. Bez tego reset hasła nie jest możliwy — dane na zaszyfrowanym dysku są dostępne tylko z kluczem szyfrowania.

Scenariusz: pechowy update

Update systemu przerwał się w połowie. System uruchamia się, ale działa niestabilnie — programy padają, ustawienia znikają.

Krok 1 — uruchamia się DISM w trybie online:

```
DISM /Online /Cleanup-Image /RestoreHealth
```

DISM skanuje Component Store (WinSxS) i naprawia uszkodzenia. Jeśli pliki źródłowe są dostępne lokalnie, naprawa trwa kilka minut. Jeśli nie, DISM ściąga zdrowe pliki z Windows Update.

Krok 2 — po DISM uruchamia się sfc:

```
sfc /scannow
```

sfc zastępuje uszkodzone pliki systemowe zdrowymi kopiami z cache lub z DISM. Jeśli DISM nie naprawił Component Store, sfc nie będzie miał z czego odtworzyć plików — wtedy potrzebny jest restart i DISM z obrazu instalacyjnego (źródło /Source).

> Kolejność ma znaczenie: najpierw DISM /RestoreHealth, potem sfc /scannow. DISM naprawia źródło (Component Store), sfc naprawia pliki. Odwrotna kolejność nie ma sensu — sfc nie ma z czego odtworzyć plików.

Dyskietka MS-DOS z lat 90. wkładała się do stacji, pisało fdisk i format, a po minucie system był gotowy. Pendrive z WinRE robi to samo — tylko dyskietka ma 1.44 MB, a install.wim ma 4 GB. To ten sam zestaw poleceń w innej skali. Terminal przed uruchomieniem systemu to nie awaryjne wyjście — to oryginalny interfejs komputera. Okienka przyjdą później, jeśli w ogóle.

Jeśli komputer nie wstaje, terminal nie zawodzi.

Pliki i foldery — bez myszy

Eksplorator plików to nakładka na system plików. Każde kliknięcie "kopiuj" i "wklej" to dwa żądania do jądra systemu. Terminal wysyła te żądania bez pośrednika. Kopiowanie, przenoszenie, zmiana nazwy, usuwanie — każda z tych operacji sprowadza się do jednego polecenia. Szybciej niż szukanie ikony.

Kopiowanie

Są trzy polecenia — każde dla innej powłoki. Różni je tylko nazwa:

Cel	CMD	PowerShell	macOS/Linux
Kopiuj	copy	Copy-Item	cp
Przenieś	move	Move-Item	mv
Usuń	del	Remove-Item	rm

Składnia jest zawsze ta sama: najpierw źródło, potem cel.

Windows CMD:

```
copy raport.docx archiwum\
```

Windows PowerShell:

```
Copy-Item raport.docx archiwum\
```

macOS:

```
cp raport.docx archiwum/
```

Wariant z kopiowaniem wielu plików naraz:

CMD:

```
copy *.txt C:\backup\
```

PowerShell:

```
Copy-Item *.txt C:\backup\
```

macOS:

```
cp *.txt ~/Backup/
```

Różnica w ukośnikach: Windows używa `\`, macOS i Linux `/`. To relikt historii — MS-DOS przejął backslasha od IBM, Unix poszedł w ukośnik.

Przenoszenie działa identycznie, tylko zmieniasz polecenie. Zmiana nazwy to przenoszenie w obrębie tego samego folderu:

```
mv stary.txt nowy.txt
```

System nie przepisuje danych. Zmienia tylko wpis w tablicy plików. Plik pod starą nazwą znika, pojawia się pod nową.

Usuwanie — gdzie jest kosz?

Wykonaj `del plik.txt` w CMD lub `rm plik.txt` w terminalu macOS. Plik znika. Nie trafia do kosza. System oznacza miejsce na dysku jako wolne. Dane fizycznie pozostają — aż do nadpisania przez inny plik.

PowerShell też usuwa na stałe. Do wysłania pliku do kosza potrzebna jest metoda .NET. Oto kompletny przykład:

```
# Zaladuj biblioteke VisualBasic
Add-Type -AssemblyName Microsoft.VisualBasic


# Usun plik do kosza
[Microsoft.VisualBasic.FileIO.FileSystem]::DeleteFile("C:\temp\stary_plik.txt",
"OnlyErrorDialogs", "SendToRecycleBin")
```

Jeśli chcesz usunąć cały folder do kosza, użyj `DeleteDirectory` zamiast `DeleteFile`:

```
[Microsoft.VisualBasic.FileIO.FileSystem]::DeleteDirectory("C:\temp\stary_folder",
"OnlyErrorDialogs", "SendToRecycleBin")
```

Tworzenie folderów

W każdej powłoce działa `mkdir`:

```
mkdir projekty
```

CMD przyjmuje też `md`. PowerShell preferuje `New-Item -ItemType Directory`. Wszystkie robią to samo: tworzą folder w bieżącej lokalizacji.

Do wybierania wielu plików według wzorca służą wildcardy.

Wzorce — wildcardy

To wzorce zastępcze. Pozwalają wybrać grupę plików bez wypisywania każdego z osobna:

Wzorzec	Znaczenie	Przykład
*	Dowolny ciąg znaków	*.txt → wszystkie pliki .txt
?	Jeden dowolny znak	dokument?.pdf → dokument1.pdf, dokumentA.pdf (nie dokument10.pdf)
[abc]	Jeden znak ze zbioru	[abc]* → pliki na a, b lub c
[0-9]	Jeden znak z zakresu	[0-9]* → pliki zaczynające się od cyfry
[!abc]	Jeden znak spoza zbioru	[!a]* → pliki nie na literę a

W CMD gwiazdka działa w pełni. ? i [] mają ograniczenia. PowerShell i macOS radzą sobie ze wszystkimi wzorcami.

Atrybuty plików

Każdy plik na Windows ma atrybuty. W CMD sprawdza się je tak:

```
attrib
```

Zobaczysz listę plików z oznaczeniami: `R` (read-only — tylko do odczytu), `H` (hidden — ukryty), `S` (systemowy). Aby zmienić:

```
attrib +h plik.txt    — ukryj plik
attrib -r plik.txt    — usuń ochronę przed zapisem
```

PowerShell daje dostęp przez `.Attributes` na obiekcie `Get-Item`:

```
(Get-Item plik.txt).Attributes
```

Aby wyświetlić ukryte pliki:

- **Windows:** `dir /ah`
- **macOS:** `ls -la`

Flaga `-la` w `ls` znaczy "long listing, all" — pokazuje wszystko, łącznie z plikami zaczynającymi się od kropki.

Różnice systemów plików

NTFS (Windows) nie rozróżnia wielkości liter, ale zapamiętuje, jaką zapisałeś. `Raport.txt` i `raport.txt` to ten sam plik.

APFS (macOS) domyślnie też nie rozróżnia. Możesz to zmienić, ale domyślnie — nie.

ext4 (Linux przez WSL) rozróżnia. `Raport.txt` i `raport.txt` to dwa różne pliki.

System	Znaki zakazane	Komentarz
Windows (NTFS)	\ / : * ? " < > |	Znaki zastrzeżone przez system plików
macOS (APFS)	/ i \0 (null)	Tylko dwa znaki są całkowicie zakazane
Linux (ext4)	\0 (null)	Null to jedyny zakazany znak

Przy przenoszeniu plików między systemami bezpieczniej trzymać się sprawdzonego zestawu znaków: litery, cyfry, myślnik, podkreślnik. To jedyny sposób, by uniknąć problemów niezależnie od systemu.

Skąd się wzięły wildcardy

Wzorce `*`, `?` i `[]` pochodzą z Unixa V6 (1975). Zaimplementował je *Ken Thompson* w powłoce `sh`. Wcześniej używał podobnej składni w edytorze `ed` — przeniósł ją do powłoki. Pomysł przyjął się i trafił do DOS-u, a stamtąd do Windows. macOS odziedziczył go po BSD.

PowerShell traktuje wildcardy inaczej. `-Filter` używa własnego silnika, nie przekazuje wzorca do systemu plików. Działa szybciej, ale `[]` obsługuje słabiej niż `-Path`.

rm -rf / — najsłynniejszy błąd

```
rm -rf /
```

To polecenie usuwa wszystko. Rozłóżmy je:

- `rm` — usuń (remove)
- `-r` — rekurencyjnie, także foldery w folderach
- `-f` — force, nie pytaj o potwierdzenie
- `/` — katalog główny, korzeń systemu

Razem: usuń cały system plików, od korzenia, bez ostrzeżenia.

Nowoczesne systemy się bronią. Linux (GNU coreutils) wymaga flagi `--no-preserve-root`, aby `rm` usunął `/`. Bez niej polecenie kończy się błędem. Na macOS (BSD) ta ochrona nie istnieje — system polega na SIP, ale to zupełnie inny mechanizm.

Efekt, gdyby się powiodło: system przestaje działać w sekundę. Uruchomione procesy blokują swoje pliki, więc nie znika wszystko — ale to wystarczy, by system był bezużyteczny. To nie jest polecenie do testowania. Nawet na maszynie wirtualnej.

Kosz z linii poleceń

Windows: PowerShell przez metodę .NET (opis wyżej). CMD — nie ma opcji.

macOS: `rm` usuwa na stałe. Aby przenieść plik do kosza, możesz zainstalować `trash` przez Homebrew:

```
brew install trash
trash plik.txt
```

Alternatywa — przenieś ręcznie do katalogu kosza:

```
mv plik.txt ~/.Trash/
```

To nieoficjalne, ale Finder odczyta plik jako znajdujący się w koszu.

Praktyczne scenariusze

Scenariusz: znajdź największe pliki na dysku

Dysk się kończy, a przyczyna nie jest oczywista. Terminal znajduje największe pliki w sekundę.

Windows (PowerShell):

```
Get-ChildItem -Path C:\ -Recurse -ErrorAction SilentlyContinue |
  Sort-Object Length -Descending |
  Select-Object -First 10 FullName, Length
```

Przeszukuje cały dysk C:, sortuje od największego, wycina 10 największych. `-ErrorAction SilentlyContinue` pomija foldery, do których nie masz dostępu.

macOS:

```
du -ah /Users/twojanazwa 2>/dev/null | sort -rh | head -20
```

`du -ah` zlicza rozmiar każdego pliku i folderu. `sort -rh` sortuje odwrotnie (największe na górze). `head -20` wycina 20 pierwszych.

Do sprawdzenia, które foldery są największe, wystarczy krótsze polecenie:

```
du -sh * | sort -rh | head -10
```

Scenariusz: zrób backup zdjęć

Kopiowanie zdjęć na pendrive — tylko nowych plików, bez nadpisywania istniejących — wymaga innego podejścia niż zwykłe kopiowanie:

Windows — Robocopy (wbudowany od Windows Vista):

```
robocopy C:\Users\twojanazwa\Pictures D:\backup-zdjec /MIR /R:2 /W:3
```

/MIR (mirror) kopiuje nowe i zmienione pliki, usuwa z celu pliki skasowane w źródle. **/R:2** i **/W:3** ograniczają ponowne próby — nie czeka w nieskończoność na odłączony dysk.

macOS (rsync — standard na każdym Uniksie):

```
rsync -avh --progress ~/Zdjęcia /Volumes/PENDRIVE/
```

-a (archive) zachowuje atrybuty i daty. **-v** (verbose) pokazuje co się dzieje. **--progress** pokazuje pasek postępu dla każdego pliku.

Po pierwszym backupie uruchom to samo polecenie — rsync skopiuje tylko nowe i zmienione pliki. Drugi raz trwa sekundy.

Sprawdź to sam

9. Utwórz plik. W PowerShell: `New-Item proba.txt`. W CMD: `echo test > proba.txt`. Na Macu: `touch proba.txt`. Sprawdź czy istnieje przez `ls`.

10. Skopiuj plik. Zrób kopię zapasową: `copy proba.txt proba_kopia.txt` (CMD), `Copy-Item proba.txt proba_kopia.txt` (PowerShell), `cp proba.txt proba_kopia.txt` (Mac). Potwierdź przez `ls` — masz dwa pliki.
11. Zmień nazwę. Przemianuj plik: `ren proba.txt test.txt` (CMD), `Rename-Item proba.txt test.txt` (PowerShell), `mv proba.txt test.txt` (Mac). `ls` pokazuje tylko `test.txt`.
12. Usuń plik. `del test.txt` (CMD), `Remove-Item test.txt` (PowerShell), `rm test.txt` (Mac). Uwaga — plik znika na stałe. Nie ma kosza.

Sieć z linii poleceń

Diagnozowanie sieci przez długie lata oznaczało jedno: logowanie do routera przez przeglądarkę albo telefon do działu IT z pytaniem "czy internet działa?". Terminal zmienia tę dynamikę. Każde z tych zadań sprowadza się do kilku klawiszy.

ping — czy komputer jest w sieci

ping testuje łączność. Wysyła mały pakiet danych na podany adres i czeka na odpowiedź.

Wpisz w terminalu:

```
ping 8.8.8.8
```

Adres `8.8.8.8` to publiczny DNS Google — jeden z najbardziej stabilnych publicznych DNS-ów. Jeśli dostajesz odpowiedzi, twój komputer ma połączenie z internetem. Jeśli nie — problem leży po twojej stronie (kabel, Wi-Fi, firewall).

Możesz pingować też nazwę domeny:

```
ping google.com
```

W tym przypadku ping najpierw zamienia `google.com` na adres IP (zapytanie DNS), a potem wysyła pakiety pod ten adres.

Windows (CMD i PowerShell): ping wysyła domyślnie 4 pakiety i kończy sam.

macOS: ping działa bez ograniczenia — będzie wysyłał pakiety, aż przerwiesz go skrótem **Ctrl + C**.

Przykładowy wynik na Windows:

```
Pinging 8.8.8.8 with 32 bytes of data:
```

```
Reply from 8.8.8.8: bytes=32 time=12ms TTL=117
Reply from 8.8.8.8: bytes=32 time=11ms TTL=117
Reply from 8.8.8.8: bytes=32 time=13ms TTL=117
Reply from 8.8.8.8: bytes=32 time=12ms TTL=117

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
Approximate round trip times:
    Minimum = 11ms, Maximum = 13ms, Average = 12ms
```

Co znaczy każda kolumna:

Pole	Co mówi
time=12ms	Czas podróży w jedną stronę i z powrotem. Im mniej, tym lepiej
TTL=117	Pozostałe skoki. Startuje od 128 (Windows) lub 64 (macOS/Linux)
Lost = 0 (0% loss)	Żaden pakiet nie zaginął — połączenie stabilne
Request timed out	Pakiet nie wrócił. Możliwe: blokada firewall, awaria, brak trasy

ipconfig / ifconfig — twoja karta sieciowa

Do sprawdzenia adresów komputera służy **ipconfig** na Windows.

```
ipconfig
```

Pojawia się lista interfejsów sieciowych: Ethernet, Wi-Fi, czasem Bluetooth. Dla każdego:

- **Adres IPv4** — twój adres w sieci lokalnej (np. `192.168.1.10`)
- **Maska podsieci** — zazwyczaj `255.255.255.0`
- **Brama domyślna** — adres routera, przez który wychodzisz do internetu

Przykładowy wynik:

```
Ethernet adapter Ethernet:
   IPv4 Address. . . . . . . . . . . : 192.168.1.10
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
   DNS Servers . . . . . . . . . . . : 8.8.8.8
                                       192.168.1.1
```

Więcej szczegółów daje:

```
ipconfig /all
```

Pojawią się serwery DNS, adres MAC (fizyczny) i informacje z DHCP.

macOS: użyj **ifconfig**. Wynik jest dużo obszerniejszy — zobaczysz też status łącza, MTU i statystyki pakietów. Szukaj interfejsu z nazwą `en0` lub `en1` — sprawdź po adresie IP, który z nich to Wi-Fi.

PowerShell ma własne polecenie:

```
Get-NetIPAddress
```

Zwraca obiekty — możesz filtrować, sortować, wyciągać konkretne właściwości.

nslookup — co kryje się za nazwą

nslookup odpytuje serwery DNS. Dzięki niemu sprawdzisz, pod jakim adresem IP działa strona.

```
nslookup google.com
```

Przykładowy wynik:

```
> nslookup google.com
Server:  dns.google
Address:  8.8.8.8

Non-authoritative answer:
Name:    google.com
Addresses:  2a00:1450:4010:c08::8b
          142.250.185.78
          142.250.185.78
```

Co znaczy każda część:

Linia	Co mówi
Server / Address	DNS, który odpowiedział (tu: 8.8.8.8, Google)
Non-authoritative answer	Odpowiedź z cache DNS, nie od serwera głównego
Adresy IPv4 i IPv6	Wszystkie IP przypisane do tej domeny

Polecenie działa identycznie na Windows i macOS. Jeśli dostajesz błąd "can't find", domena nie istnieje albo DNS nie ma dla niej rekordu.

Sprawdź openai.com:

```
nslookup openai.com
```

A teraz domena, która nie istnieje:

```
nslookup nieistniejacadomena123.pl
```

Zobaczysz `*** Nie można znaleźć nieistniejacadomena123.pl: Non-existent domain`. Różnica między istniejącą a nieistniejącą domeną jest natychmiast widoczna.

PowerShell ma też `Resolve-DnsName`, które zwraca obiekt z aliasami, adresami IPv4 i IPv6:

```
Resolve-DnsName google.com
```

tracert / traceroute — którędy idzie pakiet

tracert (Windows) i **traceroute** (macOS) pokazują trasę, jaką pakiet pokonuje do celu.

```
tracert google.com
```

Na macOS:

```
traceroute google.com
```

Wynik to lista **hopów** — każdy to jeden router po drodze. Kolumna pierwsza to numer skoku, druga i trzecia to czas odpowiedzi (trzy próby), czwarta to adres lub nazwa routera.

Jeśli przy którymś hopie pojawi się `*  *  *`, router nie odpowiedział — niekoniecznie błąd, niektóre routery ignorują pakiety ICMP.

curl — pobierz coś z internetu

curl to uniwersalny klient URL. Potrafi pobrać stronę, plik, wysłać dane — cokolwiek przez HTTP, HTTPS, FTP i kilkanaście innych protokołów.

Windows (CMD): curl jest wbudowany od Windows 10 (2018). Działa w CMD i PowerShell, ale w PowerShell wpisz jawnie `curl.exe`, bo samo `curl` to alias `Invoke-WebRequest`.

macOS: curl jest w systemie od zawsze.

Podstawowe użycie — wyświetl stronę w terminalu:

```
curl https://example.com
```

Kod HTML strony pojawia się bezpośrednio w oknie terminala. Aby zapisać wynik do pliku:

```
curl -o strona.html https://example.com
```

Pobranie pliku:

```
curl -o plik.zip https://przyklad.pl/plik.zip
```

Sprawdzenie nagłówków HTTP (bez pobierania treści):

```
curl -I https://google.com
```

W odpowiedzi serwer zwraca kod statusu (200 = OK, 404 = nie znaleziono, 301 = przekierowanie), typ treści i datę. To najszybszy sposób na sprawdzenie, czy strona działa.

Wysłanie danych POST (np. do API):

```
curl -X POST -d "nazwa=test&wartosc=123" https://httpbin.org/post
```

Podgląd całej komunikacji (SSL, nagłówki, wszystko):

```
curl -v https://example.com
```

Tekst płynie rurami — pipe i przekierowania

To najważniejsza koncepcja w terminalu. Program produkuje tekst. Drugi program go połyka. Łączy je | (pipe, rura).

W CMD wyświetl tylko pliki .txt:

```
dir | find ".txt"
```

W PowerShell:

```
Get-ChildItem | Where-Object { $_.Extension -eq ".txt" }
```

Na Macu:

```
ls | grep ".txt"
```

Przekierowanie > zapisuje output do pliku zamiast na ekran:

```
ping google.com > wynik.txt
```

W pliku `wynik.txt` zobaczysz wynik pingowania. >> dopisuje na końcu pliku:

```
ping google.com >> log.txt
```

< robi odwrotność — wczytuje zawartość pliku jako wejście dla programu:

```
sort < lista.txt
```

W PowerShell możesz łączyć rury w łańcuchy:

```
Get-Process | Where-Object { $_.WorkingSet -gt 100MB } | Sort-Object
WorkingSet -Descending | Select-Object -First 5
```

To: pobiera procesy, odcina te poniżej 100 MB, sortuje od największego, wycina 5 największych. Czytasz od lewej do prawej.

netstat — kto się łączy z twoim komputerem

netstat wyświetla aktywne połączenia sieciowe — które programy komunikują się z internetem i przez które porty.

Podstawowe użycie:

```
netstat -an
```

Opcja `-a` pokazuje wszystkie połączenia i porty nasłuchujące. `-n` wyświetla adresy i porty numerycznie, bez rozwiązywania nazw (działa szybciej).

Windows pozwala sprawdzić, który program otworzył dane połączenie:

```
netstat -b
```

Wymaga uprawnień administratora (uruchom CMD jako administrator).

W PowerShell odpowiednikiem jest:

```
Get-NetTCPConnection
```

Zwraca obiekty — status, port lokalny, port zdalny, proces. Możesz filtrować po stanie (`Where-Object State -eq "Established"`), żeby zobaczyć tylko aktywne połączenia.

---

ping — Mike Muuss i sonar

ping napisał Mike Muuss w grudniu 1983 roku. Potrzebował programu do diagnozowania problemów z siecią w laboratorium Army Ballistic Research Laboratory. Nazwa pochodzi z sonaru — tak jak sonar wysyła sygnał i czeka na echo, tak ping wysyła pakiet i czeka na odpowiedź. Muuss napisał całość w jednym wieczorze. ping zachował do dziś tę samą zasadę działania: ICMP Echo.

traceroute — Van Jacobson i TTL

traceroute powstał w latach 80. (pierwszy publiczny kod w 1988). Autor, Van Jacobson, wykorzystał pole **TTL** w nagłówku IP. Działa to tak: wysyłasz pakiet z TTL=1 — pierwszy router

odrzuca go i odsyła komunikat "czas wygasł". Dzięki temu poznajesz adres pierwszego routera. Potem wysyłasz z TTL=2 — drugi router odpowiada. I tak dalej, aż pakiet dotrze do celu.

curl — Daniel Stenberg i waluty

curl wywodzi się ze skryptu httpget, który napisał Rafael Sagula z Brazylii. W grudniu 1996 przejął go szwedzki programista Daniel Stenberg. W 1997 roku program nazywał się urlget, a w marcu 1998 ukazała się pierwsza stabilna wersja pod nazwą curl 4.0. Dziś curl jest standardem — wbudowanym w Windows, macOS i prawie każdą dystrybucję Linuksa. Szacuje się, że curl jest zainstalowany na ponad 20 miliardach urządzeń [DO WERYFIKACJI].

Porty — cyfrowe drzwi

Każde połączenie sieciowe używa portu — liczby od 0 do 65535. Port można porównać do drzwi: adres IP to adres budynku, port to numer drzwi wewnątrz. Standardowe porty:

Port	Protokół	Zastosowanie
21	FTP	Przesyłanie plików
22	SSH	Zdalny terminal
25	SMTP	Wysyłanie e-maili
53	DNS	Zapytania o adresy IP
80	HTTP	Zwykłe strony WWW
443	HTTPS	Szyfrowane strony WWW
3306	MySQL	Bazy danych
3389	RDP	Pulpit zdalny Windows

Żeby zobaczyć, które porty są otwarte na twoim komputerze, użyj `netstat -an`. Linia `0.0.0.0:80` oznacza: "nasłuchuję na porcie 80 na wszystkich interfejsach".

Testowanie portu — czy usługa nasłuchuje

Sam ping nie wystarczy. Ping może działać, ale strona może nie odpowiadać, bo port jest zamknięty. Do testowania portów służą inne polecenia:

Windows PowerShell:

```
Test-NetConnection google.com -Port 443
```

Wynik: `TcpTestSucceeded : True` — port otwarty. `False` — zamknięty lub firewall blokuje.

macOS:

```
nc -zv google.com 443
```

`nc` (netcat) to scyzoryk sieciowy. `-z` znaczy 'tylko skanuj, nie wysyłaj danych', `-v` to verbose.

Jeśli nie masz nc, możesz użyć `telnet`:

```
telnet google.com 443
```

Jeśli port jest otwarty, ekran robi się pusty. Naciśnij **Ctrl +]**, potem wpisz `quit` i Enter, żeby wyjść.

Praktyczne scenariusze

Scenariusz: sprawdź, dlaczego internet nie działa

Strony się nie ładują. Zanim telefon do dostawcy, warto przeprowadzić diagnozę krok po kroku.

Krok 1 — czy komputer ma adres IP?

```
ipconfig
```

Chodzi o linię `IPv4 Address`. Jeśli pojawia się adres zaczynający się od `169.254`, DHCP nie działa — router nie przydzielił adresu. Rozwiązanie:

```
ipconfig /release
ipconfig /renew
```

Krok 2 — ping do routera

```
ping 192.168.1.1
```

Brak odpowiedzi oznacza problem w sieci lokalnej (kabel, Wi-Fi, router).

Krok 3 — ping do internetu

```
ping 8.8.8.8
```

Jeśli krok 2 działa, a ten nie — router ma łączność z komputerem, ale nie z internetem. Wtedy restart routera.

Krok 4 — sprawdź DNS

```
nslookup google.com
```

Jeśli ping na 8.8.8.8 działa, ale strony nie otwierają się w przeglądarce — problemem jest DNS. Spróbuj:

```
ipconfig /flushdns
```

Jeśli to nie pomoże, zmień DNS na 8.8.8.8 (Google) w ustawieniach karty sieciowej.

Scenariusz: pobierz całą stronę internetową

Do ściągnięcia dokumentacji, regulaminu czy instrukcji nie trzeba otwierać przeglądarki. Terminal radzi sobie szybciej:

Windows (PowerShell):

```
Invoke-WebRequest -Uri https://example.com/strona.html -OutFile strona.html
```

Pobiera plik HTML na dysk. Po otwarciu w przeglądarce wygląda jak strona.

Jeśli chcesz wyciągnąć sam tekst, bez znaczników HTML:

```
(Invoke-WebRequest -Uri https://example.com).Content
```

macOS — curl (jest domyślnie):

```
curl -o strona.html https://example.com/strona.html
```

`curl` to standardowe narzędzie do transferu danych. **-o** zapisuje do pliku. **-O** (wielkie) zachowuje oryginalną nazwę pliku.

Pobierz plik, a nie stronę:

```
curl -O https://example.com/plik.pdf
```

Przydaje się do skryptów aktualizujących programy — zamiast klikać *Download*, pobierasz najnowszą wersję jedną komendą.

Do podglądu samych nagłówków HTTP (bez pobierania treści):

```
curl -I https://example.com
```

Zobaczysz kod odpowiedzi (200 = OK, 404 = nie znaleziono, 301 = przekierowanie) i serwer.

Sprawdź to sam

13. Sprawdź łączność. W terminalu wpisz `ping google.com`. Jeśli dostajesz odpowiedzi — internet działa. Na Windows ping skończy sam po 4 pakietach. Na Macu przerwij `Ctrl+C`.
14. Sprawdź swój adres IP. W CMD: `ipconfig`, w PowerShell: `Get-NetIPAddress`, na Macu: `ifconfig`. Znajdź linię IPv4 — to twój adres w sieci lokalnej.
15. Sprawdź DNS. Wpisz `nslookup wikipedia.org`. Zobaczysz adres IP Wikipedii. Porównaj z wynikiem dla `nslookup google.com` — każda domena ma inne IP.
16. Prześledź trasę. Wpisz `tracert google.com` (Windows) lub `traceroute google.com` (Mac). Zobaczysz przez które routery przechodzi pakiet do Google.
17. Pobierz nagłówki HTTP. Wpisz `curl -I https://example.com`. Zobaczysz odpowiedź serwera: kod 200 oznacza że strona działa.

Pierwszy skrypt — przestań robić to samo ręcznie

Trzy, cztery, pięć poleceń dziennie. Te same. Jutro powtórka. Pojutrze też.
Po tygodniu każdy zaczyna szukać asystenta, który robi to za niego.
Nazywa się **skrypt**.

> W latach 90. i 2000. wszystko szło w stronę GUI. Okienka, ikony, kreatory — tak miała wyglądać przyszłość komputerów.
> W tle działała jednak jedna rzecz, której nie dało się zastąpić okienkami: automatyzacja w firmach, skrypty na serwerach, administracja masowa. GUI jest dobre dla człowieka. Terminal jest dobry dla powtarzalności.
> CMD przetrwał nie dlatego, że był dobry. Przetrwał dlatego, że był wbudowany w świat, który już działał. I który bez terminala przestałby działać z dnia na dzień.

POZIOM 1

Co to jest skrypt

Skrypt to plik tekstowy, w którym zapisujesz polecenia jedno po drugim.
System czyta je od góry do dołu i wykonuje każde po kolei.
Bez skryptu: cd backup, mkdir 2025-01-15, copy pliki.txt . — ręcznie, każde polecenie osobno.
Ze skryptem: te trzy linijki lądują w pliku backup.bat, a całość odpala się jednym poleceniem.

Działa to jak przepis kulinarny. Są składniki i kolejne kroki. Nie trzeba myśleć przy każdym kroku „co teraz" — wykonuje się to, co napisano. Skrypt to przepis na konkretną czynność przy komputerze.

Batch (.bat) na Windows

Najstarszy format skryptów Windows. Plik z rozszerzeniem .bat to lista poleceń cmd.exe.

Zacznij od pustego pliku. Wpisz:

```
@echo off
echo Witaj w skrypcie batch
pause
```

- @echo off — wyłącza wyświetlanie samych komend na ekranie. Bez tego każda linia wyświetli się, zanim się wykona.
- echo Witaj w skrypcie batch — wypisuje tekst na ekran.
- pause — zatrzymuje skrypt i czeka, aż naciśniesz dowolny klawisz.

Zapisz jako witaj.bat, kliknij dwukrotnie. Otworzy się czarne okno z napisem.

Batch używa zmiennych w formacie %NAZWA%. Oto kilka przykładów:

```
@echo off
echo Twoja nazwa uzytkownika: %USERNAME%
echo Nazwa komputera: %COMPUTERNAME%
echo Sciezka systemowa: %WINDIR%
pause
```

`%USERNAME%`, `%COMPUTERNAME%` i `%WINDIR%` to zmienne systemowe — Windows ustawia je podczas startu. Można też zrobić własną:

```
@echo off
set SCIEZKA=C:\MojePliki
echo Katalog docelowy: %SCIEZKA%
pause
```

Polecenie set tworzy zmienną na czas działania skryptu. Po zamknięciu okna znika.

Przykład praktyczny — skrypt do backupu. Otwórz notatnik, wklej to, zapisz jako backup.bat:

```
@echo off
REM Skrypt do codziennego backupu
set DATA=%DATE:~-4%%DATE:~-10,2%%DATE:~-7,2%
echo Tworze kopie zapasowa z dnia %DATA%
mkdir D:\backup\%DATA%
copy C:\MojePliki\*.* D:\backup\%DATA%
```

```
echo Gotowe! Pliki skopiowane do D:\backup\%DATA%
pause
```

Jak użyć: wrzuć backup.bat na pulpit, kliknij dwukrotnie. Skrypt tworzy folder z dzisiejszą datą w D:\backup i kopiuje tam wszystkie pliki z C:\MojePliki.

Jedno kliknięcie. Reszta dzieje się sama.

Uwaga: wycinki %DATE% zależą od formatu daty w systemie. Na polskim Windows data ma postać DDD DD.MM.RRRR. Wtedy %DATE:~-4% daje rok, a %DATE:~-10,2% — miesiąc. Na angielskim Windows (MM/DD/RRRR) indeksy są inne. Dostosuj do swojego formatu.

PowerShell (.ps1) na Windows

PowerShell to nowocześniejsza powłoka. Jej skrypty mają rozszerzenie .ps1.

Podstawy wyglądają znajomo, jeśli znasz polecenia PowerShell:

- Write-Host "tekst" — wypisuje tekst
- Write-Output "tekst" — wysyła tekst do potoku (Write-Host wypisuje tylko na ekran, Write-Output przekazuje dalej w potoku)
- Zmienna: $nazwa

Różnica w zmiennych:

```
$user = $env:USERNAME
Write-Host "Czesc, $user"
```

$env:USERNAME odczytuje zmienną środowiskową USERNAME. W PowerShell zmienne systemowe są dostępne przez $env:.

Zapisz jako witaj.ps1, uruchom w terminalu:

```
.\witaj.ps1
```

Próba uruchomienia witaj.ps1 kończy się komunikatem o błędzie. PowerShell ma wbudowany mechanizm, który zatrzymuje niepodpisane skrypty.

ExecutionPolicy — dlaczego PowerShell nie chce uruchomić skryptu:

```
.\witaj.ps1
File cannot be loaded because running scripts is disabled on this system.
```

PowerShell domyślnie blokuje wszystkie skrypty. To celowy zabieg bezpieczeństwa — żeby nie uruchomić skryptu przypadkiem.

Rozwiązanie: ustaw politykę na RemoteSigned.

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

RemoteSigned znaczy: skrypty napisane lokalnie na danym komputerze mogą działać. Skrypty ściągnięte z internetu — tylko jeśli są podpisane cyfrowo. -Scope CurrentUser oznacza, że zmiana dotyczy bieżącego użytkownika, nie innych kont na tym samym komputerze.

Bash (.sh) na macOS

Na Macu domyślną powłoką jest bash (lub zsh w nowszych wersjach — składnia jest ta sama). Skrypty mają rozszerzenie .sh.

Pierwsza linia każdego skryptu bash:

```
#!/bin/bash
```

To shebang (od „shell bang"). Mówi systemowi: „ten plik ma uruchomić program /bin/bash i przekazać mu resztę pliku jako polecenia".

Druga sprawa — uprawnienia. Plik musi mieć prawo wykonania:

```
chmod +x skrypt.sh
```

chmod +x dodaje flagę execute (wykonaj). Bez tego system potraktuje plik jako zwykły tekst.

Potem uruchamiasz:

```
./skrypt.sh
```

Kropka i slash (./) mówią: weź plik z bieżącego folderu. To celowe — bash nie szuka skryptów w bieżącym folderze, żeby przypadkiem nie uruchomić złośliwego pliku.

Przykład — backup w bashu:

```
#!/bin/bash
DATA=$(date +%Y-%m-%d)
mkdir -p ~/backup/$DATA
cp ~/Dokumenty/* ~/backup/$DATA/
echo "Kopia zapasowa utworzona w ~/backup/$DATA"
```

Plik backup.sh wymaga trzech kroków: zapisania, nadania uprawnień (chmod +x) i uruchomienia (./backup.sh).

Porównanie: echo w trzech powłokach

Batch:

```
echo Witaj %USERNAME%
```

PowerShell:

```
Write-Host "Witaj $env:USERNAME"
```

Bash:

```
echo "Witaj $USER"
```

Koncept ten sam. Różni się składnia i sposób odczytu zmiennych. Batch używa procentów (%), PowerShell — dolara z prefiksem env:, bash — samego dolara.

POZIOM 2

ExecutionPolicy (PowerShell) na Windows

PowerShell ma siedem głównych wartości polityki wykonania:

Polityka	Działanie
Restricted	Żaden skrypt nie uruchomi się. Tylko pojedyncze polecenia. Domyślna na Windows.
RemoteSigned	Skrypty lokalne — tak. Skrypty z internetu — tylko podpisane.
AllSigned	Wszystkie skrypty muszą być podpisane. Nawet te napisane na twoim komputerze.
Unrestricted	Wszystkie skrypty można uruchomić. Ostrzeżenie przy skryptach zdalnych.
Bypass	Nic nie jest blokowane. Żadnych ostrzeżeń. Używana przez skrypty uruchamiane programowo.
Undefined	Nie ustawiono. Działa polityka z wyższego zakresu.
Default	Windows Client: Restricted. Windows Server: RemoteSigned.

Sprawdź swoją politykę:

```
Get-ExecutionPolicy
```

Zmiana dla bieżącego użytkownika:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Zmiana wymaga uprawnień administratora, chyba że użyjesz -Scope CurrentUser.

Shebang — jak system rozpoznaje skrypt

#! na początku pliku to magiczna liczba (magic number). System operacyjny czyta pierwsze dwa bajty pliku. Jeśli widzi #!, wie, że to skrypt. Reszta pierwszej linii to ścieżka interpretera.

Możliwe warianty:

```
#!/bin/bash
#!/usr/bin/env python3
#!/bin/zsh
```

/usr/bin/env python3 to sprytniejsza wersja — szuka interpretera w ścieżkach systemowych zamiast zakładać konkretną lokalizację.

Skrypt a wirusy — dlaczego .ps1 nie startuje dwuklikiem

Skryptów PowerShell nie da się uruchomić przez dwuklik. System nie ma domyślnego skojarzenia „otwórz .ps1 w PowerShell". To celowe zabezpieczenie.

Gdyby dwuklik działał, wystarczyłoby, że ktoś podeśle plik fajne_zdjecie.jpg.ps1. Windows domyślnie nie pokazuje rozszerzeń — ofiara widzi fajne_zdjecie.jpg, klika, a PowerShell wykonuje kod.

Bash na Macu działa inaczej — chmod +x to świadoma decyzja. Użytkownik mówi systemowi: „ten plik ma być wykonywalny". Dwuklik na .sh w Finderze otwiera terminal i pyta o potwierdzenie.

Różne podejścia do tego samego problemu — ale cel ten sam: nie dopuścić do przypadkowego uruchomienia kodu.

Argumenty — przekaż dane do skryptu

Skrypt, który robi to samo za każdym razem, jest mało przydatny. Argumenty pozwalają mu działać na różnych danych.

Batch (CMD):

```
echo Pierwszy argument: %1
echo Drugi argument: %2
```

Uruchom: skrypt.bat Ala ma kota → wyświetli oba argumenty.

PowerShell:

```
param($name, $age)
Write-Host "Mam na imię $name i mam $age lat."
```

Uruchom: .\skrypt.ps1 -name Jan -age 30

Bash:

```
#!/bin/bash
echo "Nazwa skryptu: $0"
echo "Pierwszy argument: $1"
echo "Wszystkie argumenty: $@"
echo "Liczba argumentów: $#"
```

Uruchom: ./skrypt.sh Jan Kowalski

Funkcje — nie powtarzaj się

Gdy ta sama logika pojawia się w kilku miejscach, wyciąga się ją do funkcji. Raz napisana, wołana wielokrotnie.

PowerShell:

```
function Backup-File {
    param($path)
    Copy-Item $path "$path.backup"
    Write-Host "Backup utworzony: $path.backup"
}

Backup-File -path "raport.docx"
```

Bash:

```
#!/bin/bash
backup_file() {
    cp "$1" "$1.backup"
    echo "Backup utworzony: $1.backup"
}

backup_file "raport.docx"
```

Batch ma funkcje przez `CALL` — ale to bardziej skok do etykiety niż prawdziwa funkcja. W praktyce Batch nie ma funkcji jak PowerShell czy bash.

Logowanie błędów — co poszło nie tak

Programy mają dwa strumienie wyjścia: `stdout` (standard output — normalny wynik) i `stderr` (standard error — błędy). Domyślnie oba trafiają na ekran. Można je rozdzielić.

Zapisz tylko błędy do pliku:

```
dir nieistniejacyfolder 2> error.log
```

2> przekierowuje stderr (strumień nr 2) do pliku. stdout (strumień nr 1) dalej idzie na ekran.

Rozdziel normalny output od błędów:

```
dir . > output.txt 2> error.log
```

W PowerShell odpowiednikiem jest `2>` (działa tak samo) lub `*>` do przekierowania wszystkich strumieni:

```
Get-ChildItem -Path C:\Nieistniejacy 2> bledy.log
Get-ChildItem *> wszystko.log  # wszystkie strumienie
```

W bashu:

```
ls /nieistniejacy 2> error.log
ls /nieistniejacy 2>&1  # błędy razem z normalnym outputem
```

Sprawdź to sam

18. Sprawdź ExecutionPolicy. W PowerShell wpisz `Get-ExecutionPolicy`. Zobaczysz wartość — prawdopodobnie `Restricted` (domyślna). To znaczy, że PowerShell nie pozwoli uruchomić żadnego skryptu.
19. Ustaw RemoteSigned. W PowerShell (jako administrator) wpisz `Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`. Potwierdź klawiszem `Y`. Od teraz możesz uruchamiać własne skrypty.
20. Napisz skrypt. Otwórz notatnik (albo `notepad witaj.ps1` z poziomu PowerShell). Wpisz: `'Hello World'`. Zapisz jako `witaj.ps1`.
21. Uruchom skrypt. W PowerShell wpisz `.\witaj.ps1`. Jeśli wszystko poszło dobrze, zobaczysz `Hello World`. Jeśli dalej widzisz błąd, wróć do punktu 2.

Winget - instalowanie bez klikania

Instalacja programu na Windows wyglada zwykle tak: przegladarka, Google, strona producenta, przycisk download. Plik laduje na dysku, dwuklik, **Next ? Next ? Finish**. Po dziesieciu takich rundach znika p-l godziny na klikanie tych samych okienek.

Winget omija caly lancuch: przegladarke, wyszukiwarke, strone producenta, kreator instalacji. Jedno polecenie wystarczy.

POZIOM 1

Co to jest winget

Winget to menedzer pakiet-w - program, kt-ry sam znajduje, pobiera i instaluje aplikacje za ciebie. Na Linuksie od dekad robi to **apt** (Advanced Package Tool). Na macOS **brew** (Homebrew). Na Windows dlugo nie bylo niczego wbudowanego. Az Microsoft w 2021 roku udostepnil winget. W Windows 11 jest preinstalowany. W Windows 10 wymaga aktualizacji **App Installer** z Microsoft Store. Sterujesz nim poleceniem `winget`.

R-znica miedzy instalacja przez przegladarke a przez winget:

Instalacja mysza	Instalacja terminalem
Google ? strona ? download ? next ? next ? finish ? gotowe	winget install firefox
Zajmuje 3-5 minut	Zajmuje 5 sekund

Instalacja mysza	Instalacja terminalem
Musisz recznie odznaczac adware w installerach	Czysta instalacja, zero smieci
Aktualizujesz kazdy program osobno	winget upgrade --all aktualizuje wszystko naraz

Terminal nie wyswietli paska postepu z animacja, ale zrobi to samo - szybciej i bez zbednych pytan.

Podstawowe komendy

Piec polecen wystarczy, zeby obsluzyc 95% codziennych zadan winget.

Operacja	Komenda	Opis
Szukanie	winget search nazwa	Znajduje program w repozytorium
Instalacja	winget install nazwa	Pobiera i instaluje bez okienek
Aktualizacja	winget upgrade --all	Sprawdza dostepne aktualizacje i aktualizuje wszystkie programy naraz
Lista	winget list	Wypisuje nadzorowane programy
Odinstalowanie	winget uninstall nazwa	Usuwa program szybciej niz Panel

Przyklady z zycia

Otw-rz terminal (CMD lub PowerShell - winget dziala w obu) i wpisz:

```
winget install firefox
```

Po kilku sekundach Firefox jest na dysku. To samo dla 7-Zip:

```
winget install 7zip
```

VS Code:

```
winget install vscode
```

PowerToys - zestaw narzedzi Microsoftu dla zaawansowanych uzytkownik-w:

```
winget install powertoys
```

Spotify:

```
winget install spotify
```

Kazde z tych polecen robi dokladnie to, co reczna instalacja - tylko bez zastanawiania sie, skad pobrac i kt-ra wersje wybrac.

Aktualizacja wszystkiego - jedna komenda

Najpierw sprawdz, co wymaga aktualizacji:

```
winget upgrade
```

Pojawia sie lista program-w z nowszymi wersjami. Kolumny: nazwa, zainstalowana wersja, dostepna wersja, zr-dlo.

Aby zaktualizowac wszystkie programy naraz:

```
winget upgrade --all
```

winget pobiera i instaluje nowsze wersje jeden po drugim. Nie wymaga potwierdzen - dziala bez okienek.

Jesli chcesz zaktualizowac tylko jeden program:

```
winget upgrade firefox
```

Winget sam znajduje najnowsza stabilna wersje i zastepuje stara. Bez kreator-w, bez klikania.

Rozwiazywanie problem-w z wyszukiwaniem

Po wpisaniu `winget install firefox` pojawia sie blad: "Multiple packages found" albo "No package found". Winget nie wie, kt-ry Firefox mial byc zainstalowany - w repozytorium jest ich kilka: wersja polska, angielska, ESR (dluga stabilna).

Zamiast zgadywac, sprawdz najpierw, co jest dostepne:

```
winget search firefox
```

Wynik pokazuje wszystkie pasujace pakiety z ich identyfikatorami. Na przyklad `Mozilla.Firefox`, `Mozilla.Firefox.ESR`, `Mozilla.Firefox.Polish`. Teraz widac, kt-ry wybrac.

Trzy flagi, kt-re rozwiewaja watpliwosci

`--exact` - szuka dokladnie takiej nazwy, jaka podales. Zadnych domysl-w, zadnych dopasowan czesciowych.

```
winget install --exact firefox
```

`--id` - instalacja po identyfikatorze. To najpewniejszy spos-b. Identyfikatory sa jednoznaczne - nie ma dw-ch pakiet-w o tym samym ID.

```
winget install --id Mozilla.Firefox
```

`--name` - instalacja po nazwie wyswietlanej. Przydaje sie, gdy nie pamietasz identyfikatora, ale znasz pelna nazwe programu.

```
winget install --name "Mozilla Firefox"
```

Kt-rej flagi uzywac? W 90% przypadk-w `winget install firefox` dziala bez problemu. Gdy zwraca blad, siegasz po `--id`. To najkr-tsza droga do celu.

Przyklad z niejednoznaczna nazwa

Wpisz `winget search node`. Wynik-w jest wiele: `OpenJS.NodeJS`, `OpenJS.NodeJS.LTS`, ale tez Node.js w r-znych wariantach i zupelnie inne pakiety, kt-re maja w opisie slowo "node". Samo `winget install node` zwr-ci blad "multiple packages found". Rozwiazanie: `winget install --id OpenJS.NodeJS` - i Node.js laduje na dysku w najnowszej wersji.

Import i export - reinstalacja w dw-ch komendach

Przed formatowaniem dysku albo przejsciem na nowy komputer wystarczy jedno polecenie:

```
winget export -o C:\Backup\packages.json
```

W pliku `packages.json` laduje lista wszystkich program-w, kt-re instalujesz przez winget - kazdy z identyfikatorem, zr-dlem i wersja.

Fragment takiego pliku:

```
{
  "Sources": [
    {
      "Name": "winget",
      "Packages": [
        { "PackageIdentifier": "Mozilla.Firefox" },
        { "PackageIdentifier": "Microsoft.PowerToys" },
        { "PackageIdentifier": "7zip.7zip" }
      ]
    }
  ]
}
```

Po reinstalacji systemu, zamiast klikac w przegladarce, wystarczy otworzyc terminal i wpisac:

```
winget import -i C:\Backup\packages.json
```

Winget odtwarza cala liste. Program po programie. Kazdy pobiera z oficjalnego zr-dla i instaluje w trybie cichym. Bez potwierdzania, wybierania katalogu, odznaczania adware.

Gdy jakis pakiet nie moze sie zainstalowac

Na nowym systemie niekt-re programy moga byc niedostepne - starsza wersja zniknela z repozytorium, pakiet wymaga innej architektury. Winget przerywa przy pierwszym bledzie. Rozwiazaniem jest flaga --ignore-unavailable:

```
winget import -i C:\Backup\packages.json --ignore-unavailable
```

Ta flaga kaze pominac niedostepne pakiety i kontynuowac z reszta.

A co z programami, kt-rych nie ma w winget? Dopisujesz je do listy recznie. Po imporcie instalujesz brakujace w inny spos-b - pobierasz ze strony producenta albo uzywasz alternatywnego menedzera (**Scoop**, **Chocolatey**). Plik JSON mozna edytowac w Notatniku i dodac wpis dla pakietu instalowanego normalnie recznie.

Import i export to najwieksza supermoc winget. Przy reinstalacji oszczedza godziny.

POZIOM 2

Skad winget wie, co zainstalowac

Winget nie zgaduje. Jego baza to **Windows Package Manager Community Repository** - repozytorium na GitHubie (microsoft/winget-pkgs), do kt-rego kazdy moze dodac opis pakietu. Kazdy wpis (manifest) zawiera:

- nazwe programu
- identyfikator (np. Microsoft.PowerToys)
- URL do oficjalnego instalatora
- hash pliku do weryfikacji
- informacje, czy instalacja wymaga admina

Gdy wpiszesz `winget install firefox`, winget przeszukuje manifesty i znajduje `Mozilla.Firefox`. Potem pobiera instalator z oficjalnego serwera Mozilli, weryfikuje sume kontrolna i uruchamia instalacje w trybie cichym. Ciche przelaczniki (`/S`, `/VERYSILENT` lub inne) pochodza z manifestu - kazdy pakiet ma odpowiednie parametry dla swojego instalatora.

Identyfikatory

Czasem samo `winget install firefox` nie wystarczy - nazwa moze byc niejednoznaczna lub w repozytorium jest kilka pakiet-w o podobnej nazwie. Wtedy uzywasz pelnego identyfikatora:

```
winget install Microsoft.PowerToys
```

Format to `Producent.NazwaPakietu`. Identyfikatory znajdziesz przez `winget search`. Sa jednoznaczne - nie pomylisz PowerToys od Microsoftu z czyms o podobnej nazwie.

Blokowanie aktualizacji

Czasem nie chcesz aktualizowac konkretnego programu. Nowa wersja zmienia interfejs, usuwa funkcje albo psuje kompatybilnosc. **Winget pin** blokuje pakiet - przypina go do obecnej wersji.

Dodanie blokady:

```
winget pin add --id Mozilla.Firefox
```

Od tej pory `winget upgrade --all` pomija Firefoksa. Nie aktualizuje go, nie pyta o zgode, zwyczajnie go ignoruje.

Uwaga: domyslny pin blokuje tylko masowa aktualizacje przez `winget upgrade --all`. Jesli wpiszesz `winget upgrade Mozilla.Firefox` wprost, pakiet i tak sie zaktualizuje. Aby zablokowac r-wniez bezposrednie aktualizacje, dodaj flage `--blocking`:

```
winget pin add --id Mozilla.Firefox --blocking
```

Wtedy ani `--all`, ani bezposrednie wywolanie nie ruszy Firefoksa.

Podglad zablokowanych pakiet-w:

```
winget pin list
```

Lista pokazuje przypiete pakiety z identyfikatorem i wersja, w kt-rej zostaly zablokowane.

Usuniecie blokady:

```
winget pin remove --id Mozilla.Firefox
```

Po usunieciu pinu program zn-w podlega aktualizacjom.

Po co to komu? Deweloper, kt-rego aplikacja dziala tylko z Java 8, blokuje Jave przed aktualizacja do nowszej wersji. Zesp-l utrzymujacy starsze narzedzie CLI napisane dla konkretnej wersji Node.js przypina Node.js, by `winget upgrade --all` nie zepsul srodowiska budowania.

Administrator IT w firmie zatwierdza aktualizacje pakiet-w dopiero po testach - pin trzyma je w znanej, stabilnej wersji.

Diagnostyka: winget --info

Gdy winget zachowuje sie nieoczekiwanie, pierwsze polecenie ratunkowe:

```
winget --info
```

Pokazuje informacje diagnostyczne o samym wingecie. Wynik zawiera kilka p-l:

- **Windows Package Manager** - wersja winget, np. `v1.7.10582`. Przydaje sie przy zglaszaniu bledu na GitHubie lub sprawdzeniu, czy zainstalowana jest najnowsza wersja.
- **Windows.Desktop** - wersja systemu, np. `v10.0.22631.2861`. Niekt-re funkcje winget dzialaja tylko na Windows 11.
- **System Architecture** - architektura procesora, np. `X64`. Wazne przy instalacji pakiet-w, kt-re maja osobne wersje dla ARM i x64.
- **Microsoft.DesktopAppInstaller** - pakiet App Installer ze Sklepu Microsoft, kt-ry dostarcza winget do systemu.

Przykladowy output:

```
Windows Package Manager v1.7.10582
Copyright (c) Microsoft Corporation. All rights reserved.

Windows: Windows.Desktop v10.0.22631.2861
System Architecture: X64
Package: Microsoft.DesktopAppInstaller v1.22.10582.0
```

Jesli wysylasz zgloszenie na forum technicznym, dolacz wynik `winget --info`. To jak karta pacjenta - bez niej nikt nie postawi diagnozy.

Zarzadzanie zr-dlami

Winget nie szuka pakiet-w w pr-zni. Laczy sie ze zr-dlami - repozytoriami, kt-re udostepniaja manifesty. Domyslnie podlaczone sa trzy zr-dla: **winget** (repozytorium spolecznosciowe), **msstore** (Microsoft Store) i **winget-font** (repozytorium czcionek).

```
winget source list
```

Nazwa	Zr-dlo
winget	https://cdn.winget.microsoft.com/cache

msstore	https://storeedgefd.dsx.mp.microsoft.com/v9.0
winget-font	https://cdn.winget.microsoft.com/fonts

Winget - gl-wne repozytorium spolecznosciowe. To samo, kt-re znajduje sie na GitHubie (microsoft/winget-pkgs). Ponad 10 000 pakiet-w, wszystkie przeszly weryfikacje. Instalacja z tego zr-dla dziala w trybie cichym, nie wymaga logowania.

Winget-font - repozytorium czcionek Microsoftu. Pojawilo sie w 2024 roku i umozliwia instalacje czcionek przez winget, np. `winget install --source winget-font "Cascadia Code"`. Nie wymaga logowania.

Msstore - Microsoft Store. Niekt-re pakiety z msstore moga wymagac zalogowania na konto Microsoft, szczeg-lnie platne. Nie wszystkie aplikacje ze Sklepu sa dostepne przez winget, a instalacja moze wymagac interakcji. Do tego zr-dla odwolujesz sie przez `--source msstore`:

```
winget install --id 9NBLGGH4NNS1 --source msstore
```

Dodawanie i usuwanie zr-del:

```
winget source add --name myrepo --arg https://mojerepozytorium.pl

winget source remove --name myrepo
```

Przydaje sie w firmach, kt-re maja wlasne wewnetrzne repozytorium pakiet-w. Administrator dodaje zr-dlo i wszyscy korzystaja z winget zamiast recznie instalowac firmowe oprogramowanie.

Case study: reinstalacja Windows

Od teorii do praktyki. Scenariusz: formatowanie dysku i instalacja Windows od zera. Bez winget - popoludnie z przegladarka. Z winget - dwadziescia minut.

Krok 1: Przed formatem

Otw-rz terminal (administrator) i wykonaj:

```
winget export -o C:\Backup\packages.json
```

Przenies plik `packages.json` na zewnetrzny dysk, pendrive albo do chmury. Wazy kilka kilobajt-w.

Krok 2: Po reinstalacji

Zainstaluj Windows. Zainstaluj aktualizacje. Otw-rz terminal. Jesli winget nie jest dostepny (starszy Windows 10), pobierz App Installer ze Sklepu Microsoft.

Przywr-c liste:

```
winget import -i D:\packages.json
```

Winget przechodzi przez liste i instaluje kazdy pakiet po kolei.

Krok 3: Gdy cos nie dziala

Nie wszystkie pakiety przetrwaja reinstalacje. Program m-gl zostac usuniety z repozytorium, zmienic identyfikator albo wymagac starszej wersji frameworka. Winget raportuje blad i przerywa.

Rozwiazanie:

```
winget import -i D:\packages.json --ignore-unavailable
```

Pakiety, kt-re sa dostepne, zainstaluja sie normalnie. Niedostepne zostana pominiete. Po imporcie sprawdz log i zobacz, kt-re wypadly.

Krok 4: Uzupelnienie brakujacych program-w

Czesc oprogramowania nie trafi do winget nigdy. Adobe Creative Suite. Specjalistyczne sterowniki. Programy napisane przez kolege z pracy w 2005 roku. Te instalujesz recznie.

Aby nie zgubic tych program-w przy nastepnej reinstalacji, stw-rz drugi plik - swoja wlasna liste. Zwykly tekst w Notatniku:

```
# Programy spoza winget - do recznej instalacji
Adobe Photoshop 2025 - pobierz z konta Adobe
Sterownik drukarki Kyocera - strona producenta
Narzedzie do fakturowania - instalator na pendrive
System rezerwacyjny - dostep w sieci firmowej
```

Efekt koncowy: zamiast 45 minut na pobieranie i klikanie kazdego instalatora z osobna, spedzasz 5 minut na uruchomieniu `winget import` i 10 minut na recznym dokonczeniu reszty.

Alternatywy

Winget pojawil sie w 2021 roku. Zanim Microsoft dodal go do systemu, na rynku istnialy dwa popularne menedzery firm trzecich.

Cecha	Winget	Chocolatey	Scoop
Rok debiutu	2021	2011	2015
Komenda	winget	choco	scoop
Pakiety	~10 000	~10 000+	~5 000+
Wymaga admina	czesto	czesto	nie

Cecha	Winget	Chocolatey	Scoop
Instalacja	wbudowany w Win11	Set-ExecutionPolicy + skrypt	iex + skrypt
Najlepszy do	codziennych program-w	starszych aplikacji	narzedzi dev
Windows 11	preinstalowany	brak	brak

Kt-ry wybrac: jesli chcesz w pelni automatyczna konfiguracje nowego komputera bez przegladarki, wypr-buj Scoop. Jesli potrzebujesz starszych, rzadkich aplikacji - Chocolatey. Jesli chcesz rozwiazania wbudowanego w system, bez instalowania dodatkowego oprogramowania - winget.

A na macOS?

Na macOS tym samym zajmuje sie **Homebrew** (komenda `brew`). Instalujesz przez terminal:

```
brew install firefox
```

To samo podejscie, inna powloka. Homebrew ma wiecej pakiet-w niz winget i istnieje od 2009 roku. Wr-cimy do niego w Rozdziale 10.

Praktyczne scenariusze

Scenariusz: zainstaluj nowy komputer od zera

Nowy laptop nie wymaga godziny klikania Next ? Next ? Finish. Jeden skrypt instaluje wszystko od razu.

Krok 1 - otw-rz PowerShell jako Administrator

Windows + X ? Windows PowerShell (Admin) lub terminal (Admin)

Krok 2 - zainstaluj menedzera Winget (jesli Windows 10 bez aktualizacji):

```
Add-AppxPackage -Path https://aka.ms/getwinget
```

Krok 3 - stw-rz plik z lista program-w:

```
@echo off
rem instaluj wszystko.cmd
```

```
winget install Google.Chrome
winget install 7zip.7zip
winget install VideoLAN.VLC
winget install Spotify.Spotify
winget install Microsoft.VisualStudioCode
winget install Docker.DockerDesktop
winget install OBSProject.OBSStudio
winget install Notepad++.Notepad++
winget install Git.Git
winget install Python.Python.3.12
winget install WhatsApp.WhatsApp
winget install Valve.Steam
winget install Discord.Discord
```

Skrypt dziala w tle. Po powrocie z kawy komputer jest gotowy do pracy.

Krok 4 - zaktualizuj wszystko na raz:

```
winget upgrade --all
```

Ta komenda aktualizuje wszystkie programy zainstalowane przez Winget. *Uruchamiaj raz w miesiacu.*

Wariant dla zaawansowanych - zapisz liste zainstalowanych program-w do pliku:

```
winget export -o C:\Users\twojanazwa\Desktop\programy.json
```

Na nowym komputerze zaimportuj: `winget import -i C:\Users\twojanazwa\Desktop\programy.json`

Sprawdz to sam

22. Otw-rz terminal i wpisz `winget search firefox`. Zobaczysz liste pasujacych pakiet-w - w tym `Mozilla.Firefox` w kilku wariantach (polski, angielski, ESR).
23. Jesli chcesz, zainstaluj PowerToys: `winget install powertoys`. Nie musisz niczego potwierdzac.
24. Sprawdz, co zainstalowal Microsoft: `winget list | findstr "Microsoft"` w CMD, albo `winget list | Select-String Microsoft` w PowerShell.

Terminal na Macu — Unix w garniturze

Pod interfejsem graficznym macOS — za oknami, za ikonami, za wszystkim co widać na ekranie — pracuje Unix. Terminal to okno do tej warstwy: bezpośrednie, bez filtra graficznego.

Czas je otworzyć.

POZIOM 1

Jak otworzyć Terminal na Macu

Są trzy drogi. Każda działa.

Finder — najdłuższa, ale zapamiętasz na stałe. Otwórz Findera. W pasku menu: Idź → Narzędzia. Albo skrót **Shift+Cmd+U**. Zobaczysz folder Narzędzia. W nim Terminal. Dwuklik.

Spotlight — najszybsza. Wciśnij **Cmd+Spacja**, wpisz `terminal`, Enter. Terminal staje przed tobą w ułamku sekundy.

Launchpad — wizualna. Kliknij Launchpad w Docku (albo ściśnij kciuki i złącz palce na trackpadzie). Znajdź folder **Inne**, w nim Terminal.

Spotlight to najszybsza droga. Po tygodniu używa się go do wszystkiego — wyszukiwania plików, otwierania aplikacji, kalkulatora. Terminal wpisuje się w ten nawyk.

Gdy otworzysz Terminal pierwszy raz, zobaczysz białe okno z czarnym tekstem. Kursor miga po znaku **$** (lub **%** w **zsh**). To twój **prompt** — znak, że powłoka czeka na polecenie.

Bash czy zsh — którą masz

Każdy Mac ma domyślnie zainstalowaną **powłokę** (shell) — program, który interpretuje polecenia i przekazuje je systemowi.

Do macOS Catalina (2019) domyślną powłoką był **bash** — Bourne Again Shell. To standard na Linuksie od dekad. Brian Fox napisał go w 1989 roku jako wolne oprogramowanie. Od macOS Catalina Apple zmienił domyślną powłokę na **zsh** (Z Shell).

Powód? Licencyjny. Bash na Macu to wersja 3.2 na licencji **GPL v2** (GNU General Public License). Nowsze wersje (4.0+) przeszły na **GPL v3** — bardziej restrykcyjną licencję, której Apple wolał uniknąć. Zamiast modernizować starego bash 3.2, Apple wymienił domyślną powłokę na zsh, który używa licencji MIT — luźniejszej, bardziej przyjaznej dla Apple. Koniec historii. Żadnej dramatycznej zmiany.

Bash — Bourne Again Shell. Działa od 1989. Jest na każdym Linuksie i na Macu sprzed Cataliny. Składnia: zmienne przez `$NAZWA`, pętle, ify, funkcje.

zsh — Z Shell. Powstał w 1990 roku (rok po bashu). Na pierwszy rzut oka identyczny. W codziennym użytkowaniu różnice są kosmetyczne:

- lepsze autouzupełnianie (wpisz `cd /u/lo/ap` i tab — zsh podpowie `/usr/local/apache2`)
- lepszy globbing (`ls **/*.txt` działa w zsh bez włączania dodatkowych opcji)
- inny wygląd prompta (na zsh często widać `~ %` zamiast `bash-3.2$`)
- modułowa konfiguracja przez frameworki (Oh My Zsh)

Której używać? Mac po 2019 ma domyślnie zsh. I tak powinien zostać. Do rzeczy opisanych w tej książce obie powłoki pracują identycznie. Starszy Mac z bashem? Też dobrze.

Sprawdź sam: **`echo $SHELL`**. Powie ci, czego używasz.

Różnice między CMD/PS na Windows a Terminal na Macu

Przy przejściu z Windows kilka rzeczy rzuca się w oczy od razu. Oto zestawienie:

Obszar	Windows (CMD/PS)	macOS (Terminal)
Dyski	C:\, D:\ — litery dysków	/ — jeden korzeń
Wielkość liter	Nie rozróżnia (NTFS)	APFS domyślnie nie, ale komendy tak

Obszar	Windows (CMD/PS)	macOS (Terminal)
Admin	Kliknij prawym → Uruchom jako administrator	sudo przed poleceniem
Pomoc	help (CMD) lub Get-Help (PS)	man komenda
Separator ścieżek	\ (backslash)	/ (slash)
Katalog domowy	%USERPROFILE% lub C:\Users\nazwa	/Users/nazwa lub ~
Kończenie procesu	Ctrl+C kopiuje?	Ctrl+C przerywa, Cmd+C kopiuje
Zmienne środowiskowe	%NAZWA% lub $env:NAZWA	$NAZWA

Struktura katalogów na Macu

Po przejściu z Windows pierwsze co rzuca się w oczy — brak liter dysków. Zamiast `C:\Program Files` masz **`/Applications`**. Zamiast `C:\Windows` — `/System`. Zamiast `C:\Users\twojanazwa` — `/Users/twojanazwa`.

`/` (korzeń) — to odpowiednik `C:\`. Wszystko zaczyna się stąd. `/` to najwyższy poziom systemu plików. Wpisz `cd /`, a pokażą ci się katalogi główne. Wykonaj `ls /`, zobaczysz wszystko co system wystawia na tym poziomie.

`/Applications` — tu leżą programy. Odpowiednik `C:\Program Files`. `ls /Applications` pokaże wszystkie zainstalowane programy. Każda aplikacja na Macu to folder z rozszerzeniem `.app`.

`/System` — system operacyjny. Nie ruszaj go. Odpowiednik `C:\Windows`. Zawiera jądro, biblioteki systemowe, frameworki Apple. Wejdziesz tu z `sudo` i coś skasujesz — Mac może przestać działać.

`/Users` — pliki użytkowników. Odpowiednik `C:\Users`. Każdy użytkownik ma swój katalog: `/Users/twojanazwa`. W środku: `Desktop`, `Documents`, `Downloads`, `Library` (konfiguracja programów), `Pictures`, `Music`, `Movies`.

`~` (tylda) — skrót do twojego katalogu domowego. Zamiast pisać `/Users/marek`, piszesz `~`. To samo co `%USERPROFILE%` na Windows. Działa wszędzie: `cd ~`, `ls ~/Desktop`, `open ~/Downloads`.

`/tmp` — pliki tymczasowe. System i programy wyrzucają tu śmieci. Przy restarcie zawartość jest czyszczona. Odpowiednik `C:\Windows\Temp` i `%TEMP%`.

`/etc` — konfiguracja systemu. Pliki ustawień: lista użytkowników (`/etc/passwd`), konfiguracja sieci (`/etc/hosts`). Żeby zmienić coś w `/etc`, potrzebujesz `sudo`.

```
cd /             # jesteś w korzeniu
ls /Applications # lista programów
cd ~/Desktop     # jesteś na pulpicie
```

Pierwsze komendy na Macu

`ls`, `cd`, `pwd` działają identycznie jak na Windows — opisaliśmy je w Rozdziale 2. Oto komendy, które poznasz dopiero na Macu:

Komenda	Działanie	Przykład
open .	Otwiera bieżący folder w Finderze	open . — otwiera Finder w tym folderze
open -a Safari	Uruchamia program Safari	open -a TextEdit notatka.txt
command -v python3	Sprawdza czy program istnieje	command -v git → /usr/bin/git
pwd	Pokazuje ścieżkę (z /)	pwd → /Users/twojanazwa
say tekst	Mówi na głos (TTS)	say "Hello" — głośnik mówi
caffeinate	Nie usypia Maca podczas działania	caffeinate -i 3600 — godzina bez snu

`open .` to komenda, której będziesz używać najczęściej. Nawigujesz w terminalu, trafiasz na folder z plikami. Zamiast klikać przez Findera — wpisujesz `open .` i Finder otwiera się w tym miejscu. Kropka oznacza "bieżący katalog".

`command -v` to zamiennik `where` z Windows. Jeśli program jest zainstalowany, zobaczysz ścieżkę. Jeśli nie — brak outputu.

`Uwaga: command -v python3` może nie znaleźć Pythona, dopóki nie zainstalujesz Command Line Tools (`xcode-select --install`).

Skróty klawiszowe w Terminalu

Terminal rządzi się własnymi skrótami. Część działa jak w każdym programie, część zaskakuje — zwłaszcza gdy przychodzisz z Windows.

Skrót	Działanie	Uwaga
Ctrl+C	Przerywa bieżący proces (SIGINT)	To nie kopiuje! Cmd+C kopiuje
Ctrl+D	Kończy strumień (EOF)	W pustej powłoce zamyka Terminal
Ctrl+Z	Zawiesza proces (SIGTSTP)	Wznawiasz przez fg lub bg
Tab	Autouzupełnianie	W zsh poprawia literówki
Ctrl+A	Skok na początek linii	Odpowiednik Home
Ctrl+E	Skok na koniec linii	Odpowiednik End
Ctrl+U	Czyści linię od kursora	Kasuje całość gdy kursor na końcu
↑/↓	Historia poleceń	Zapamiętuje setki komend
Cmd+T	Nowa karta	Jak w przeglądarce
Cmd+W	Zamknij kartę	Bez ostrzeżenia
Cmd++ / Cmd+-	Powiększ / pomniejsz	Zmienia czcionkę
Cmd+K	Wyczyść ekran	Jak clear — historia przewijania zostaje

Ctrl+C to najważniejszy skrót i najczęstsze źródło frustracji. W terminalu **Ctrl+C** nie kopiuje. Wysyła sygnał **SIGINT** (SIGnal INTerrupt) do działającego procesu. Program się zatrzymuje, wracasz do prompta. Zapamiętaj: **Cmd+C to kopiowanie**, Ctrl+C to zabijanie procesu.

POZIOM 2

Dlaczego macOS ma terminal

Ta historia ma konkretny początek: 1997 rok.

W 1997 roku Apple kupił NeXT — firmę Steve'a Jobsa, którą założył po odejściu z Apple w 1985. NeXT produkował komputery i system operacyjny NeXTSTEP. NeXTSTEP zbudowano na jądrze Mach i warstwie BSD (Berkeley Software Distribution) — jednej z wersji Unixa.

Gdy Apple kupił NeXT, dostał w pakiecie gotowy, nowoczesny system z jądrem Unix. Apple przerobił wygląd, dodał warstwę graficzną (Aqua), przemianował na **Mac OS X** (2001). Pod spodem został Unix. Terminal i wszystkie komendy unixowe przeszły bez zmian.

To nie jest emulacja, nie jest warstwa kompatybilności. To bezpośredni spadek po NeXTSTEP → BSD → Unix z Bell Labs. Twój Mac ma certyfikat **UNIX 03** (od 2007 roku) — oficjalne potwierdzenie, że spełnia standard Single UNIX Specification. Terminal w systemie Windows nie ma takiego certyfikatu i nigdy nie miał.

zsh vs bash — głębiej

Na poziomie podstawowym różnicy nie widać. Głębiej różnice istnieją, ale codziennej pracy nie zmieniają.

Autouzupełnianie. Bash uzupełnia komendy, nazwy plików, zmienne. zsh robi to samo, ale lepiej: poprawia literówki, podpowiada argumenty (`git chec` → Tab → `git checkout`), rozwijane menu gdy pasuje kilka opcji. W bashu dostajesz beep i listę. W zsh strzałką wybierasz z listy.

Globbing. `ls ~/**/*.txt` — rekurencyjne wyszukanie plików we wszystkich podkatalogach. W zsh działa domyślnie. W bashu 3.2 na Macu (do Cataliny) opcja `globstar` nie istnieje — `**` w ogóle nie zadziała, bash zwróci błąd "no matches found". W bashu 4.0+ trzeba włączyć `shopt -s globstar`. Różnica jest więc konkretna: zsh obsługuje rekurencyjny globbing od razu, bash nie.

Prompt. W bashu standardowy prompt wygląda tak: `bash-3.2$`. W zsh: `hostname ~ %`. Oba możesz dowolnie zmieniać (`PS1` w bashu, `PROMPT` w zsh).

W codziennej pracy: różnica między toyotą a hondą. Oba jeżdżą, oba mają klimatyzację, oba dowożą. Różnice widać w detalach — po roku, nie po tygodniu.

sudo — skąd się wzięło i jak działa

sudo nie jest magiczne. To program o nazwie `sudo`, który robi jedną rzecz: uruchamia podane polecenie jako inny użytkownik (domyślnie root). Działa od 1980 roku (najpierw na Uniwersytecie SUNY Buffalo, potem w standardzie BSD).

Mechanizm:

1. Wpisujesz `sudo rm plik.txt`

2. sudo sprawdza `/etc/sudoers` — plik konfiguracyjny, który określa, kto i co może uruchomić przez sudo

3. Jeśli nazwa użytkownika jest w grupie admin (lub ma wpis w sudoers), system pyta o **własne** hasło (nie hasło roota)

4. Po uwierzytelnieniu sudo może zapamiętać hasło na kilka minut (timestamp_timeout w /etc/sudoers) — nie trzeba wpisywać go przy każdym wywołaniu. Na macOS domyślnie pyta za każdym razem (timestamp_timeout=0).

5. Polecenie wykonuje się jako root

sudo -s — otwiera powłokę roota. Działa w bashu i zsh. Prompt zmienia się na `#` (hash) zamiast `$`. Jesteś rootem. Każde polecenie działa bez pytania. Nie zostań w tym trybie. Zamknij przez `exit`.

Różnica względem Windows: w Windows podnosisz uprawnienia dla całego okna terminala (uruchom jako administrator). Na Macu podnosisz dla konkretnego polecenia. To bezpieczniejsze — nie siedzisz cały czas jako admin, tylko wskakujesz na chwilę.

Zmienne środowiskowe i pliki startowe

Gdy otwierasz terminal, uruchamia się proces, który dziedziczy zestaw **zmiennych środowiskowych** (environment variables). Przechowują one informacje o systemie: gdzie szukać programów, jaki jest katalog domowy, jakiej powłoki się używa.

`echo $PATH` pokazuje zmienną PATH. To lista katalogów oddzielonych dwukropkami, w których system szuka programów. Gdy wpisujesz `ls`, system przeszukuje każdy katalog z PATH. Znajduje `/bin/ls` i uruchamia go. Na Windows PATH rozdzielają średniki (`;`). Na Macu dwukropki (`:`).

`echo $HOME` — twój katalog domowy. Odpowiednik `%USERPROFILE%` z Windows.

`echo $SHELL` — ścieżka do twojej powłoki. Zobaczysz `/bin/zsh` albo `/bin/bash`.

Skąd się biorą te zmienne? Terminal przy starcie ładuje pliki konfiguracyjne. To zwykłe pliki tekstowe, które wykonują się jak skrypt.

Dla **zsh** (domyślna powłoka na Macu od 2019):

- **.zprofile** — uruchamia się raz, przy logowaniu. Służy do ustawień, które mają zadziałać tylko raz. W większości przypadków zostawiasz go pustego.
- **.zshrc** — uruchamia się przy każdym otwarciu Terminala. To plik, który będziesz edytować najczęściej. Tu dodajesz aliasy (`alias ll='ls -la'`), zmienne PATH, funkcje pomocnicze.

- **.zshenv** — uruchamia się zawsze, także przy skryptach nieinteraktywnych. Rzadko go edytujesz.

Dla **bash** (starsze Maci, do Cataliny):

- **`.bash_profile`** — odpowiednik `.zprofile`.
- **`.bashrc`** — odpowiednik `.zshrc`.
- **`.bash_login`** — alternatywa dla `.bash_profile`.

Gdzie te pliki leżą? W twoim katalogu domowym. `~/.zshrc`, `~/.bashrc`. Są ukryte (kropka na początku). W Finderze nie zobaczysz ich domyślnie. W terminalu zobaczysz przez `ls -la ~`.

Przykład: dodaj katalog do PATH. Masz skrypty w `~/scripts` i chcesz je uruchamiać bez podawania ścieżki.

```
echo 'export PATH=$PATH:~/scripts' >> ~/.zshrc
source ~/.zshrc
```

Pierwsza komenda dopisuje linię do pliku `.zshrc`. Druga ładuje zmiany od razu, bez zamykania terminala. Od teraz `~/scripts` jest w PATH i możesz wpisać `nazwa_skryptu` zamiast `~/scripts/nazwa_skryptu`.

Dostosowanie Terminala

Domyślny Terminal — biały ekran, czarny tekst. Działa, ale po godzinie zaczyna męczyć oczy. Możesz to zmienić.

Terminal → Settings → Profiles. Tu znajdziesz gotowe schematy kolorów (profile). Każdy profil to zestaw: kolor tła, kolor tekstu, czcionka, przezroczystość, kursor. W macOS Monterey i starszych: Terminal → Preferences → Profiles.

Profile domyślne: **Basic** (biały ekran, czarny tekst), **Pro** (ciemny, zielony tekst — lepszy dla oczu przy dłuższej pracy), **Homebrew** (ciemny, granatowo-zielony), **Novel** (kremowe tło, brązowy tekst — styl vintage), **Ocean**, **Man Page**, **Red Sands**.

Możesz importować własne schematy. **Solarized** (ciemny zielono-żółty, zaprojektowany pod kątem czytelności) i **Catppuccin** (pastelowy, popularny wśród deweloperów) to dwa popularne wybory. W sieci znajdziesz pliki `.terminal`.

Poza kolorami możesz zmienić: czcionkę (wyszukaj czcionki programistyczne: Menlo, Fira Code, JetBrains Mono), przezroczystość tła. Możesz też wyłączyć dźwięk dzwonka — nikt za nim nie tęskni.

iTerm2 — alternatywa dla domyślnego Terminala

iTerm2 to darmowy zamiennik domyślnego Terminala. Do pobrania z iterm2.com. Ma funkcje, których Apple nie dodał do swojego terminala: podział okna na panele (Split Panes), wyszukiwanie wsteczne (Cmd+F, potem znajdź), profiles, lepsze kopiowanie (zaznacz tekst, automatycznie trafia do schowka).

Dla początkującego domyślny Terminal wystarczy. iTerm2 przydaje się przy pracy na wielu serwerach — SSH po lewej, logi na górze po prawej, edytor na dole, wszystko w jednym oknie. Na razie wystarczy wiedzieć, że taka opcja istnieje. Wraca się do niej, gdy domyślny terminal zaczyna uwierać.

Jak znaleźć odpowiedź samemu — man, apropos, whatis

Na Macu (i każdym Uniksie) dokumentacja jest wbudowana w system. Bez Google, bez przeglądarki. Terminal i komenda man — to wystarczy.

man — instrukcja obsługi

`man` (manual) wyświetla dokumentację komendy. Wpisz:

```
man ls
```

Zobaczysz kilkanaście ekranów tekstu. Przewijaj strzałkami lub **PgUp/PgDn**. Szukaj frazy: naciśnij /, wpisz szukaną frazę, Enter. **n** — następne trafienie. **q** — wyjście.

Struktura manuala jest zawsze taka sama: nazwa, składnia, opis, flagi, przykłady, pliki, zobacz też. Gdy poznasz ten układ, czytasz każdą komendę od razu.

Sprawdź sam:

```
man ssh
```

Zobaczysz, że ssh ma kilkadziesiąt flag. Nie musisz ich wszystkich znać — wystarczy, że wiesz, gdzie je znaleźć, gdy zajdzie potrzeba.

apropos — szukaj, gdy nie znasz nazwy

Nie pamiętasz, jak się nazywa komenda do sprawdzania sieci? Wpisz:

```
apropos network
```

System przeszuka wszystkie manuale i wypisze komendy związane z siecią: `ping`, `ifconfig`, `netstat`, `tcpdump` — i kilkadziesiąt innych.

`apropos` szuka w opisach komend. Działa jak wyszukiwarka — wpisujesz temat, dostajesz listę narzędzi. To odpowiednik `Get-Command` z PowerShell (opisanego w Rozdziale 1).

whatis — szybki opis

Chcesz wiedzieć, co robi dana komenda, bez czytania całego manuala?

```
whatis ls
```

Dostajesz jedno zdanie: "ls -- list directory contents". Tyle. Czasem to wystarczy.

Jeśli nie masz manuala (błąd "No entry for ..."), brakuje bazy. Zainstaluj przez:

```
sudo mandb
```

Rada na koniec: zanim wkleisz komendę z internetu, wpisz `man` i sprawdź, co robi każda flaga. Ludzie nie kradną kont, bo nie znasz komend — ludzie kradną konta, bo wklejasz komendy, których nie rozumiesz. `man` to pierwsza linia obrony.

Sprawdź to sam

25. Otwórz Terminal. **Cmd+Spacja**, wpisz terminal, Enter. Zobaczysz okno z promptem `%` lub `$`.
26. Sprawdź swoją powłokę. Wpisz: `echo $SHELL`. Zobaczysz `/bin/zsh` (nowszy Mac) lub `/bin/bash` (starszy).
27. Otwórz Dokumenty w Finderze. Wpisz: `open ~/Documents`. Finder wyskoczy w tym folderze. To najszybszy sposób na przejście z terminala do okienka.
28. Zajrzyj do manuala. Wpisz: `man ls`. Przeczytaj opis. Przewiń strzałkami. Wyjdź klawiszem `q`.
29. Sprawdź czy Python jest. Wpisz: `command -v python3`. Jeśli zobaczysz ścieżkę — Python jest. Jeśli nic nie widać — brak CLT. Zainstaluj: `xcode-select --install`.

Nawigacja po Macu bez myszy — cd, ls, pwd, find

Przed tobą Terminal na Macu. Czarny ekran, biały tekst, migający kursor. Polecenie **ls** wyświetla listę plików — tak samo jak na Windows. Ale kolory nie działają. To pierwsza różnica, którą warto znać.

POZIOM 1

ls na Macu — te same komendy, inne kolory

ls na Macu pochodzi z BSD (Berkeley Software Distribution), nie z GNU jak na Linuxie. Na co dzień różnica sprowadza się do jednego: **flagi kolorów**.

Na Linuxie włączasz kolory przez **ls --color**. Na Macu ta flaga nie istnieje. Używasz **ls -G**. Reszta działa tak samo:

- **ls -l** — szczegółowa lista: **uprawnienia**, liczba dowiązań, **właściciel**, grupa, **rozmiar**, **data modyfikacji**, nazwa pliku
- **ls -la** — to samo plus **ukryte pliki** (te z kropką na początku nazwy, np. .bash_profile)
- **ls -lh** — rozmiary w formacie czytelnym dla człowieka: 4,2 KB zamiast 4096
- **ls -lt** — sortowane od najnowszego

Przykład outputu:

```
ls -laG
```

```
drwxr-xr-x  12 user  staff    384 May 31 10:00 .
drwxr-xr-x   5 user  staff    160 May 30 08:00 ..
-rw-r--r--   1 user  staff   1024 May 31 09:00 notatka.txt
drwxr-xr-x   3 user  staff     96 May 29 14:00 Projekty
```

Połącz flagi dowolnie: **ls -laG** daje szczegółową listę z ukrytymi plikami i kolorami.

```
ls -laG
```

cd i pwd — po co komu pwd

cd zmienia katalog. Działa identycznie jak na Windows — **cd Documents** przenosi do **Documents**, **cd ..** cofa poziom wyżej.

pwd (print working directory) wypisuje gdzie jesteś. Nie używasz go często, bo prompt zwykle pokazuje ścieżkę. Przydaje się gdy:

- masz otwarte trzy okna terminala i gubisz które jest którym
- skrypt wpisuje **cd** i tracisz orientację
- pracujesz przez SSH na zdalnej maszynie

Rozdział 2 pokazał podstawy nawigacji na Windows. Na Macu są trzy skróty, które oszczędzają czas:

- **cd -** — wraca do **poprzedniego** katalogu (nie nadrzędnego, a ostatniego w którym byłeś)
- **cd ~** — przenosi do katalogu domowego (/Users/twojanazwa)
- **cd /** — przenosi do korzenia systemu plików

```
pwd
cd -
```

Ścieżki: / vs \ i ~ (tylda)

Windows używa backslashe (C:\Users\twojanazwa). Mac używa forward slashe (/Users/twojanazwa/Documents). Jeden korzeń / zamiast liter dysków.

Tylda (~) to skrót do katalogu domowego. Wpisz **cd ~** — jesteś w /Users/twojanazwa. Działa też z **ls**: **ls ~/Documents** wypisze zawartość bez wchodzenia.

Możesz też odwołać się do katalogu domowego innego użytkownika: **ls ~michal/Documents**.

Dokańczanie tabulatorem działa lepiej niż w CMD. Wpisz **cd Do** i naciśnij Tab. Terminal sam uzupełni do **Documents**. Jeśli jest kilka pasujących opcji, naciśnij Tab dwa razy — zobaczysz listę. Działa dla plików, katalogów, a nawet komend.

find — szukanie plików

Finder na Macu ma pasek wyszukiwania. Z terminala robisz to samo przez **find**.

```
find /Users -name "*.pdf"
```

To przeszukuje cały katalog **Users** (wraz z podkatalogami) i wypisuje wszystkie pliki .pdf.

Przydatne flagi — zestawienie:

Flaga	Znaczenie	Przykład
-type f	Tylko pliki, nie katalogi	find ~ -type f
-type d	Tylko katalogi	find ~ -type d
-name "*.pdf"	Szukaj po nazwie	find ~ -name "*.pdf"
-size +10M	Pliki większe niż 10 MB	find ~ -size +10M
-mtime -7	Zmodyfikowane ostatnie 7 dni	find ~ -mtime -7
-maxdepth 3	Maksymalnie 3 katalogi w głąb	find ~ -maxdepth 3 -name *.txt

Łączysz je bez ograniczeń:

```
find ~ -type f -name "*.jpg" -size +1M
```

To znajdzie wszystkie zdjęcia JPG większe niż 1 MB w twoim katalogu domowym.

locate to szybsza alternatywa — przeszukuje prekompilowaną bazę danych zamiast rzeczywistego dysku. Na Macu baza nie istnieje od razu. Musisz ją zbudować:

```
sudo /usr/libexec/locate.updatedb
```

Potem **locate** "**faktura**" działa natychmiast. Wadą jest to, że baza domyślnie odświeża się raz w tygodniu przez launchd, więc może nie znaleźć plików dodanych dzisiaj.

POZIOM 2

mdfind — spotlight z terminala

Spotlight to silnik wyszukiwania macOS, który indeksuje treść plików, metadane, a nawet zawartość PDF-ów. Z terminala używasz go przez **mdfind**.

```
mdfind -name "faktura"
```

Wyniki są natychmiastowe — Spotlight przeszukał już dysk w tle. Działa też filtrowanie po typie:

```
mdfind -onlyin ~/Documents -name "faktura"
```

-onlyin ogranicza zakres do konkretnego katalogu, co przyspiesza wyszukiwanie gdy wiesz gdzie szukać.

Różnica między find a mdfind

Cecha	find	mdfind	locate
Szybkość	Wolne — przeszukuje dysk w czasie rzeczywistym	Natychmiastowe — korzysta z indeksu Spotlight	Natychmiastowe — korzysta z bazy danych
Aktualność	100% — widzi każdy nowy plik	Opóźnione — indeks aktualizuje się w tle	Opóźnione — baza odświeża się raz w tygodniu
Szuka w treści	Nie	Tak — PDF, DOC, tekst	Tylko nazwy
Wymaga przygotowania	Nie	Nie — Spotlight działa zawsze	Tak — sudo /usr/libexec/locate.updatedb
Najlepszy do	Plików dodanych przed chwilą	Szukania po treści dokumentów	Szybkich lookupów znanych plików

Dowiązania symboliczne

Symlink (dowiązanie symboliczne) to plik-wskaźnik. Wskazuje na inny plik lub katalog. Na Windows nazywa się **mklink**. Na Macu tworzysz go przez **ln**:

```
ln -s /Users/twojanazwa/Documents/skrypty.sh ~/szybki_dostep.sh
```

Pierwszy argument to cel (dokąd wskazuje), drugi to nazwa dowiązania (co tworzysz). **ls -l** pokazuje strzałką, gdzie prowadzi:

```
lrwxr-xr-x  1 twojanazwa  staff  42 May 31 12:00 szybki_dostep.sh ->
/Users/twojanazwa/Documents/skrypty.sh
```

Usuwasz symlink jak zwykły plik (**rm nazwa**). Oryginał zostaje nietknięty.

Sprawdź to sam

30. Sprawdź gdzie jesteś. W Terminalu wpisz `pwd`. Zobaczysz ścieżkę — np. `/Users/twojanazwa`.

31. Wylistuj ukryte pliki. Wpisz `ls -la`. Zobaczysz wszystko w bieżącym katalogu, także pliki z kropką (np. `.zshrc`).

32. Znajdź plik po nazwie. Wpisz `find ~ -name "*.pdf"`. Znajdzie wszystkie pliki PDF w twoim katalogu domowym.

33. Znajdź pliki starsze niż 7 dni. Wpisz `find ~ -type f -mtime +7`. Zobaczysz pliki, których nie ruszałeś od tygodnia. Przydaje się do wiosennych porządków.

Uprawnienia — dlaczego Mac ci odmawia

./skrypt.sh — i terminal odpowiada: **Permission denied**. Próba zapisu w **/usr/local** kończy się komunikatem: **Operation not permitted**. To nie blokada. Mac pilnuje reguł, które Unix ma w DNA od pięćdziesięciu lat. Czas je poznać.

POZIOM 1

Dlaczego Mac mówi "Permission denied"

System uprawnień Unix narodził się w Bell Labs na początku lat 70. Powstał dla maszyn, z których korzystało wielu ludzi naraz — każdy potrzebował własnych plików i nie mógł ruszać cudzych. macOS, choć ma grafikę, mysz i touch bar, pod spodem dalej używa tych samych reguł.

Każdy plik i katalog ma trzy atrybuty: **właściciela** (kto go stworzył), **grupę** (zbiór użytkowników, np. **staff**) i **prawa** (co każdy może z nim zrobić).

W terminalu wpisz:

```
ls -la ~
```

Zobaczysz wiersze takie jak ten:

```
-rw-r--r--@ 1 mzdrowy staff   1024 Jan 1 12:00 plik.txt
```

Rozłóż go na części. Pierwszy znak to typ pliku: **-** zwykły plik, **d** katalog, **l** dowiązanie symboliczne. Dalej idzie dziewięć znaków w trzech grupach po trzy:

rw- — prawa właściciela (mzdrowy): czytaj, pisz, brak wykonania

r-- — prawa grupy (staff): tylko czytaj

r-- — prawa innych: tylko czytaj

Każda grupa to trzy pozycje: **r** (read — odczyt), **w** (write — zapis), **x** (execute — wykonanie). Brak uprawnienia znaczy **-**.

Znak **@** na końcu oznacza rozszerzone atrybuty — na przykład flagę **quarantine**.

chmod — zmiana uprawnień

Masz skrypt, który nie chce działać. Terminal mówi **Permission denied** przy próbie **./skrypt.sh**. Brakuje litery **x**.

```
chmod +x skrypt.sh
```

chmod (change mode) zmienia prawa. **+x** dodaje wykonanie dla wszystkich. Możesz operować też na konkretnych grupach: **chmod u+x** daje execute tylko właścicielowi, **chmod go-w** zabiera zapis grupie i innym.

Zapis liczbowy jest precyzyjniejszy. Opiera się na trzech wartościach:

4 = read (r)

2 = write (w)

1 = execute (x)

Wartości sumuje się dla każdej grupy. Jedna cyfra na grupę.

```
chmod 755 skrypt.sh
```

7 (4+2+1) daje właścicielowi rwx, **5** (4+1) daje grupie r-x, **5** daje innym r-x. To standard dla programów.

Dla plików tylko do odczytu:

```
chmod 644 notatka.txt
```

6 (4+2) to rw- dla właściciela, **4** (r--) dla grupy i innych. Takie prawa ma większość twoich plików.

Blokada pliku przed wszystkimi? **chmod 000 plik** — zero we wszystkich grupach. Nawet właściciel go nie odczyta, dopóki nie zmieni praw.

chown — zmiana właściciela

Czasem musisz oddać plik komuś innemu. Do tego służy **chown** (change owner).

```
sudo chown root plik.txt
```

Bez **sudo** nie zmienisz właściciela — tylko root ma prawo oddawać pliki. Po tym poleceniu plik należy do roota. Nie ruszysz go bez sudo.

Żeby go odzyskać:

```
sudo chown mzdrowy plik.txt
```

Możesz zmienić właściciela i grupę naraz:

```
sudo chown mzdrowy:staff plik.txt
```

Grupa **staff** to domyślna grupa kont użytkowników na macOS.

Uprawnienia a programy

Żeby uruchomić program z terminala, plik musi mieć bit **x**. Bez niego **./program** zwróci **Permission denied**. Dotyczy to zarówno plików binarnych, jak i skryptów bash, python czy ruby.

Skrypt bez **chmod +x** uruchomisz przez **bash skrypt.sh** — bash czyta plik i interpretuje linijki, system nie uruchamia go bezpośrednio.

macOS dokłada własną warstwę ochrony. Ściągnąłeś aplikację z internetu? System blokuje ją komunikatem: program nie może zostać otwarty, ponieważ pochodzi od nieznanego dewelopera. To **Gatekeeper** — system Apple sprawdzający sygnatury.

Flaga blokady nazywa się **quarantine** (kwarantanna). Sprawdzisz ją przez **xattr** — narzędzie do odczytu i zapisu rozszerzonych atrybutów:

```
xattr -l ~/Pobrane/*.dmg
```

xattr wyświetla atrybuty pliku. Widzisz **com.apple.quarantine**? System dostał sygnał: ten plik przyszedł z internetu. Usuniesz flagę tak:

```
xattr -d com.apple.quarantine program
```

Po tej operacji uruchomisz program normalnie. Bierzesz na siebie odpowiedzialność.

POZIOM 2

setuid/setgid — program działający w imieniu innych

Czasem program musi działać z prawami kogoś innego. Weź **passwd** — zmiana hasła. Program zapisuje do zastrzeżonej bazy haseł systemu, która należy do roota. Zwykły użytkownik nie ma tam dostępu. A jednak każde konto może zmienić własne hasło.

Rozwiązanie: bit **setuid** (Set User ID). Program z **chmod u+s** uruchamia się z prawami właściciela pliku, nie osoby wywołującej.

```
chmod u+s program
```

Wystarczy sprawdzić:

```
ls -l /usr/bin/passwd
```

Zobaczysz **-rwsr-xr-x** — litera **s** w miejscu **x** dla właściciela. To setuid. Program działa jako root, nawet gdy uruchomi go zwykły użytkownik.

Bit **setgid** (**chmod g+s**) zmienia grupę efektywną. Dla katalogów setgid sprawia, że nowe pliki dziedziczą grupę katalogu. Przydaje się w katalogach współdzielonych.

Uwaga: setuid na skryptach bash to klasyczne ryzyko bezpieczeństwa. Współczesne systemy (w tym macOS) ignorują bit setuid na skryptach interpretowanych. Dotyczy tylko plików binarnych.

ACL — gdy rwx to za mało

Trzy grupy, trzy prawa. Co w sytuacji, gdy dostęp do pliku ma mieć tylko dwoje konkretnych ludzi, a reszta zespołu — nie?

Wchodzą **ACL** (Access Control Lists). Na macOS wsparcie ACL jest domyślnie włączone.

Listę ACL wyświetla:

```
ls -le plik.txt
```

Znak **+** po prawach oznacza, że plik ma wpisy ACL.

Dodaj regułę:

```
chmod +a "user ania allow write" plik.txt
```

Ania dostaje prawo zapisu, niezależnie od grupy i reszty świata. Możesz mieszać **allow** i **deny**, ustawiać prawa dla użytkowników i całych grup.

ACL dają elastyczność kosztem czytelności. **ls -le** pokaże całą listę. W praktyce na prywatnym Macu rzadko ich potrzebujesz.

chflags — ukryte blokady macOS

macOS ma jeszcze jedną warstwę: flagi plików przez **chflags** (change flags).

```
chflags hidden plik.txt
```

Plik znika z Findera. Terminal dalej go widzi. **chflags nohidden** przywraca widoczność.

```
chflags uchg plik.txt
```

Flaga **uchg** (user immutable) zabezpiecza plik przed kasowaniem i modyfikacją — nawet ty, jako właściciel, nie ruszysz go bez cofnięcia flagi. **chflags nouchg** zwalnia blokadę.

Flagi działają obok praw Unix, nie zamiast nich.

macOS a Windows — różnica filozofii

Windows od NTFS (1993) ma ACL jako podstawowy system. Każdy plik ma listę precyzyjnie określającą: użytkownik X ma prawo Y. Potężne i skomplikowane.

Unix poszedł inną drogą: trzy bity, trzy grupy, koniec. ACL dodano później jako rozszerzenie — stąd znak **+** w **ls -l**. Do dziś większość systemów Unix i Linux działa na czystym rwx i rzadko sięga po ACL.

Na macOS dwoistość jest wyraźna: są podstawowe **chmod** na co dzień i ACL oraz flagi na wyjątki. Używaj pierwszego, gdy możesz. Drugiego, gdy musisz.

Sprawdź to sam

1. **ls -la ~** — przejrzyj uprawnienia plików w katalogu domowym. Każdy wiersz to jedna historia.

2. **chmod 600 ~/.ssh/id_rsa** — klucz prywatny SSH musi mieć takie prawa (tylko właściciel czyta i pisze). Przy szerszych prawach SSH odmówi współpracy.

3. **xattr -l ~/Pobrane/** — znajdź plik z flagą quarantine i zdecyduj, czy chcesz go odblokować.

Homebrew — sklep, którego Apple nie chciał zrobić

Apple ma App Store. Tysiące aplikacji z ikonami, oknami, sliderami. Wszystko ładnie opakowane. Ale co z programami bez GUI? Z kompilatorem, menedżerem plików, klientem FTP bez okienek? Na to Apple sklepu nie zrobił. Zrobił go Max Howell w 2009 roku i nazwał **Homebrew**.

POZIOM 1

Czym jest Homebrew

Homebrew to menedżer pakietów — program, który znajduje, pobiera i instaluje inne programy za użytkownika. Na Linuksie od dekad robi to **apt** (Advanced Package Tool). W Windows robi to **winget** (Rozdział 6). Na macOS robi to `brew`.

Różnica między instalacją przez przeglądarkę a przez Homebrew:

- Przeglądarka: Safari → wpisanie nazwy → znalezienie strony → szukanie linku do pobrania → rozpakowanie DMG → przeciągnięcie do Aplikacji → często wpisanie hasła. I tak dla każdego programu osobno.
- Homebrew: `brew install nazwa`. Koniec.

Terminal nie wyświetli animowanego paska postępu, ale zrobi to samo — szybciej i bez ciągłego pytania o hasło.

Homebrew nie wymaga uprawnień administratora. Wszystko instaluje do własnego katalogu: `/usr/local` na Macach Intel, `/opt/homebrew` na **Apple Silicon** (M1 i nowsze). Systemu nie rusza — nie ma ryzyka, że coś się popsuje w macOS.

Instalacja

Instalacja zaczyna się od jednego polecenia w Terminalu:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

To polecenie ściąga skrypt instalacyjny z GitHuba i uruchamia go. Skrypt sam instaluje **Command Line Tools for Xcode** — zestaw programów deweloperskich Apple (kompilator `clang`, `git`, `make` i inne). Nie potrzeba całego Xcode, tylko tych programów. Skrypt ogarnia to za użytkownika.

Na **Apple Silicon** (M1/M2/M3/M4) po instalacji trzeba dodać Homebrew do zmiennej `PATH`. Skrypt informuje o tym na końcu — zwykle to dwie komendy do wklejenia:

```
echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprofile
eval "$(/opt/homebrew/bin/brew shellenv)"
```

Te dwie linie dodaje się od razu — żeby terminal wiedział, gdzie szukać `brew`.

Podstawowe komendy

Siedem poleceń wystarczy, żeby ogarnąć 95% tego, co można zrobić z Homebrew.

Szukanie

```
brew search wget
```

Przeszukuje repozytorium Homebrew. Jeśli dokładna nazwa nie jest znana, wystarczy wpisać fragment — `brew search` znajdzie wszystko, co pasuje.

Instalowanie

```
brew install wget
```

Pobiera i instaluje. Żadnych okienek. Żadnych pytań o zgodę na dostęp do plików.

Aktualizacja listy pakietów

```
brew update
```

Odświeża lokalną listę dostępnych pakietów — mówi `brew`, co nowego pojawiło się w repozytorium. Warto to zrobić przed każdą większą instalacją.

Aktualizacja pakietów

```
brew upgrade
```

Sprawdza wszystkie zainstalowane programy i aktualizuje te, które mają nowszą wersję. Jedno polecenie zamiast ręcznego sprawdzania każdego programu z osobna.

Lista zainstalowanych

```
brew list
```

Pokazuje wszystko, co zostało zainstalowane przez `brew install`.

Odinstalowanie

```
brew uninstall wget
```

Usuwa pakiet. Czysto, bez zostawiania resztek w `~/Library/Application Support`.

Informacje o pakiecie

```
brew info wget
```

Pokazuje wersję, zależności, datę wydania i link do strony domowej. Przydaje się przed instalacją.

Aktualizacja i czyszczenie

Homebrew nie aktualizuje programów automatycznie. Trzeba zrobić to ręcznie — ale to jedna komenda.

Lista przestarzałych pakietów:

```
brew outdated
```

Pojawia się lista programów z zainstalowaną i dostępną wersją.

Aktualizacja wszystkich pakietów:

```
brew upgrade
```

Homebrew pobiera nowsze wersje wszystkich programów, które na to zasługują. Aktualizacja pojedynczego pakietu też jest możliwa:

```
brew upgrade wget
```

Po aktualizacji zostają stare wersje. Czas na czyszczenie:

```
brew cleanup
```

cleanup usuwa stare wersje i tymczasowe pliki. Można sprawdzić, ile miejsca zajmują:

```
brew cleanup --dry-run
```

--dry-run tylko pokazuje, co zostanie usunięte — niczego nie kasuje. Warto uruchomić go przed właściwym cleanup, żeby zobaczyć, czego się spodziewać.

Caski — aplikacje z GUI

Homebrew instaluje głównie programy terminalowe. Ale co z Firefoksem, VS Code albo Spotify? Do instalacji aplikacji z GUI przez terminal służą **caski**.

Cask to wersja Homebrew dla aplikacji z GUI. Różnicę robi flaga `--cask`:

```
brew install --cask firefox
brew install --cask visual-studio-code
brew install --cask spotify
```

To samo co ściąganie ze strony i przeciąganie do Aplikacji — tylko że w jednej linii. Homebrew znajduje oficjalne źródło, pobiera DMG, montuje, kopiuje i sprząta.

W skrócie:

- **Formula** — pakiet CLI (terminalowy), instalujesz bez flagi: `brew install wget`
- **Cask** — aplikacja z GUI, instalujesz z `--cask`: `brew install --cask firefox`

Jedno brew, dwie kategorie. Terminal obsługuje obie.

POZIOM 2

Homebrew na Apple Silicon

Na Macach z procesorem Intel Homebrew instaluje wszystko do `/usr/local/bin/`. Na Apple Silicon (M1, M2, M3, M4) instaluje do `/opt/homebrew/bin/`. Dlaczego?

Apple Silicon to architektura **arm64**. Intel to **x86_64**. Te dwie architektury nie mieszają się — program skompilowany dla Intela nie działa natywnie na Apple Silicon i odwrotnie. Homebrew na Apple Silicon kompiluje wszystko dla arm64. Działa szybciej i zużywa mniej baterii.

Homebrew można zainstalować w dwóch wersjach na jednym komputerze (Intel przez Rosettę), ale nie jest to konieczne. Wersja arm64 obsługuje wszystko — jeśli jakiś program nie ma wersji na Apple Silicon, macOS uruchomi go automatycznie przez Rosettę.

brew bundle — jeden plik, cały komputer

Przeprowadzka na nowego Maca nie wymaga instalowania 30 programów ręcznie przez tydzień. Jeden plik, jedno polecenie.

Najpierw na starym komputerze:

```
brew bundle dump
```

To polecenie tworzy plik `Brewfile` w bieżącym katalogu. Zawiera listę wszystkich zainstalowanych pakietów — formuły, caski, a nawet aplikacje z Mac App Store.

Potem na nowym komputerze:

```
brew bundle install
```

Homebrew czyta `Brewfile` i instaluje wszystko po kolei. 30 programów w 10 minut. Skrypt działa w tle — po powrocie z kawy komputer jest gotowy.

Plik `Brewfile` to zwykły tekst. Ląduje na GitHub, w repozytorium z konfiguracją, na pendrive.

brew doctor — coś nie działa?

Homebrew bywa kapryśny. Po aktualizacji macOS, po zmianie ścieżek, po nieudanej instalacji — czasem `brew` zaczyna marudzić. Wtedy:

```
brew doctor
```

`brew doctor` skanuje instalację i wypisuje ostrzeżenia: brakujące zależności, nieaktualne ścieżki, problemy z uprawnieniami. Większość z nich można naprawić przez `brew doctor` — sam podaje komendy do wklejenia.

Jeśli `brew` przestaje działać, pierwszy krok: `brew doctor`. Drugi krok: zastosować się do tego, co mówi. W 90% przypadków to wystarczy.

Sprawdź to sam

34. `brew search python` — znajdź pakiety związane z Pythonem. Zobacz, ile ich jest: `python` (formula) vs `python@3.11` (starsza wersja) vs caski z Pythonem w nazwie.

35. `brew info wget` — jeśli masz zainstalowany, zobacz wersję i zależności. Jeśli nie masz, zobacz, czy chcesz go zainstalować. Wget to klasyczny program do ściągania plików z sieci — przydaje się częściej, niż myślisz.
36. Jeśli nie masz jeszcze Homebrew, ale chcesz zobaczyć, jak działa instalacja czegoś przydatnego: `brew install --cask rectangle`. Rectangle to menedżer okien — przeciągasz okno do krawędzi, a ono dopasowuje się do połowy ekranu. Działa w tle, nie wymaga konfiguracji. Po instalacji znajdziesz go w Launchpadzie jak każdą inną aplikację.

Skrypty bash na Macu

Terminal. Polecenia. Niektóre sekwencje wracają codziennie — te same flagi, te same ścieżki, ta sama kolejność. Trzeci, czwarty, piąty raz. Wtedy pojawia się myśl: czas napisać skrypt.

W systemie Windows skrypty pisze się w batch i PowerShell. Na Macu w **bash** (lub **zsh** od macOS Catalina). Składnia bliższa staremu Uniksowi. Bash istnieje od 1989 — to, co działa na Macu, ruszy też na Linuxie i w WSL.

POZIOM 1

Pierwszy skrypt bash na Macu

Skrypt to zwykły plik tekstowy. Dostaje rozszerzenie `.sh`, w pierwszej linii ląduje **shebang** — informację dla systemu, który interpreter ma uruchomić plik.

W edytorze wystarczy wpisać dwie linie:

```
#!/bin/bash
echo "Witaj, świecie!"
```

Plik zapisuje się jako `hello.sh`.

Shebang (`#!`) musi być pierwszą linią pliku — dosłownie pierwszym znakiem. Trzy warianty:

- `#!/bin/bash` — konkretna ścieżka do basha. Działa na każdym Macu.
- `#!/usr/bin/env bash` — szuka basha w PATH. Bardziej przenośne na Linuxie. Na Macu obie formy działają tak samo.
- `#!/bin/zsh` — dla skryptów w zsh. Na Macu zsh jest domyślną powłoką od 2019 roku.

Nawet jeśli domyślną powłoką jest zsh, skrypt z shebangiem `#!/bin/bash` i tak ruszy w bashu.

Plik nie uruchomi się sam. Potrzebuje prawa do wykonania (**execute**):

```
chmod +x hello.sh
```

Do uruchomienia:

```
./hello.sh
```

Kropka i ukośnik (`./`) to celowy zabieg. System nie szuka skryptów w bieżącym katalogu ze względów bezpieczeństwa.

Bez shebangu system zgadnie interpreter po bieżącej powłoce. Lepiej nie polegać na zgadywaniu.

Zmienne i argumenty

Skrypt przyjmuje argumenty z linii poleceń. Każdy ląduje w zmiennej pozycyjnej: **`$1`** (pierwszy), **`$2`** (drugi). **`$@`** zwraca wszystkie argumenty naraz.

Zmodyfikowana wersja `hello.sh`:

```
#!/bin/bash
echo "Witaj, $1!"
```

Testowanie:

```
./hello.sh Krzysztof
```

Wynik: `Witaj, Krzysztof!`

`$?` przechowuje kod wyjścia ostatniego polecenia. Zero to sukces, każda inna wartość to błąd.

Zmienne środowiskowe są dostępne od razu: **`$HOME`** to katalog domowy (`/Users/twojanazwa`), **`$USER`** to nazwa użytkownika. Wczytanie danych od użytkownika przez **`read`**:

```
#!/bin/bash
read -p "Podaj nazwe: " nazwa
echo "Witaj, $nazwa!"
```

`read -p` wyświetla prompt i czeka na wejście. Wynik trafia do zmiennej `nazwa`.

Instrukcje warunkowe

Automatyzacja zaczyna się od warunków. Testowanie istnienia pliku:

```
if [ -f "$1" ]; then
```

```
    echo "Plik $1 istnieje"
fi
```

`[ -f "$1" ]` to test — zwraca prawdę, jeśli `$1` wskazuje na istniejący plik. Analogicznie **`[ -d "$1" ]`** sprawdza, czy to katalog.

Porównanie stringów:

```
if [ "$1" = "help" ]; then
    echo "Uzycie: skrypt.sh [nazwa]"
fi
```

Nawiasy kwadratowe wymagają odstępów — `[ "$1" = "help" ]` działa, `["$1"="help"]` nie.

Skrócone formy: **`&&`** uruchamia drugie polecenie tylko, gdy pierwsze się udało (kod wyjścia 0). **`||`** uruchamia, gdy pierwsze się nie udało (kod wyjścia ≠ 0).

Przy bardziej rozbudowanej logice lepiej użyć jawnego `if`.

Pętle

Pętla **`for`**, by przejść po wszystkich plikach:

```
for f in *.txt; do
    echo "Znaleziono: $f"
done
```

Bash rozwija `*.txt` na listę plików przed pętlą.

Czytanie linii z pliku:

```
while read line; do echo "$line"; done < plik.txt
```

`while read line` czyta plik linia po linii. Przydaje się do logów i konfiguracji.

Pętla po zakresie liczb:

```
for i in {1..10}; do echo "Licznik: $i"; done
```

`{1..10}` generuje sekwencję od 1 do 10. Działa też z literami (`{a..z}`).

Uwaga — krok w zakresie (`{1..10..2}`) wymaga basha 4.0+. Apple dostarcza na Macu bash 3.2, więc `{1..10..2}` zwróci dosłownie `{1..10..2}` zamiast sekwencji. Na Macu działają tylko zakresy bez kroku, np. `{1..10}`.

POZIOM 2

Różnice bash: Mac (BSD) vs Linux (GNU)

Mac używa narzędzi z projektu BSD, Linux — z GNU. Składnia różni się w kluczowych miejscach.

`sed -i` — edycja pliku w miejscu. Na Linuxu: `sed -i 's/stary/nowy/g' plik.txt`. Na Macu potrzebny jest pusty string: `sed -i '' 's/stary/nowy/g' plik.txt`. Bez tego pojawi się błąd.

`date` — Linux: `date -d "2026-06-01"`. Mac: `date -j -f "%Y-%m-%d" "2026-06-01"`. Warto sprawdzić `man date` na systemie docelowym.

`find` — opcja `-printf` (GNU) nie działa na Macu. Zamiast tego `-exec` lub `-print0` z `xargs`.

Jeśli skrypt ma działać na obu, system sprawdza się przez **`uname`** — zwraca `Darwin` na macOS, `Linux` na Linuxie:

```
if [[ "$(uname)" == "Darwin" ]]; then
    SED_OPTS=(-i '')
else
    SED_OPTS=(-i)
fi
```

Debugowanie

Skrypt nie działa? Pomaga flaga **`-x`**:

```
bash -x hello.sh "test"
```

Bash wypisze każdą komendę przed wykonaniem, poprzedzoną znakiem `+`. Widać wartości zmiennych, wybraną gałąź warunku, wyniki komend. Najszybszy sposób na znalezienie błędu.

Debugowanie tylko fragmentu skryptu: **`set -x`** (włącza śledzenie) i **`set +x`** (wyłącza).

Dla bezpieczeństwa na początku skryptu stosuje się **`set -eu`** (lub rozdzielnie: `set -e` i `set -u`). `-e` kończy skrypt przy pierwszym błędzie (kod wyjścia ≠ 0). `-u` traktuje użycie niezdefiniowanej zmiennej jako błąd i kończy skrypt. Razem zapobiegają cichym błędom, które ciężko znaleźć.

Traps: obsługa Ctrl+C

Gdy użytkownik wciśnie Ctrl+C, system wysyła sygnał `SIGINT` (numer 2). Domyślnie skrypt kończy się natychmiast. Jeśli trzeba posprzątać przed wyjściem, służy do tego `trap`. W funkcji czyszczącej `exit` kończy skrypt z odpowiednim kodem:

```
#!/bin/bash
cleanup() {
    echo "Sprzątanie..."
    rm -f /tmp/tempfile_$$.txt
    exit 1
}
trap cleanup INT

echo "Naciśnij Ctrl+C"
sleep 100
```

`trap cleanup INT` znaczy: gdy przyjdzie sygnał INT (Ctrl+C), wykonaj `cleanup`. Można przechwycić też `SIGTERM` i `EXIT` — odpala się przy każdym zakończeniu skryptu.

Zadania cykliczne

Na Linuxie i starszych wersjach macOS zadania cykliczne uruchamia się przez cron. Na współczesnym Macu Apple promuje launchd — system init, który zarządza procesami i harmonogramem. Cron nadal działa, ale launchd daje więcej kontroli.

Cron: wystarczy uruchomić `crontab -e` i dodać linię:

```
0 6 * * * /Users/twojanazwa/scripts/poranny.sh
```

Format: minuta, godzina, dzień miesiąca, miesiąc, dzień tygodnia, polecenie. Gwiazdka = każdy. `0 6 * * *` = codziennie o 6:00.

Launchd używa plików XML (`.plist`) w `~/Library/LaunchAgents/`. Przykład (`local.poranny.plist`):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist>
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>local.poranny</string>
    <key>ProgramArguments</key>
    <array><string>/Users/twojanazwa/scripts/poranny.sh</string></array>
```

```
    <key>StartCalendarInterval</key>
    <dict>
        <key>Hour</key><integer>6</integer>
        <key>Minute</key><integer>0</integer>
    </dict>
</dict>
</plist>
```

Ładowanie: `launchctl load ~/Library/LaunchAgents/local.poranny.plist`.

Przy podstawowych zadaniach konfiguracja crontabu jest szybsza. Launchd daje logowanie, restart przy błędzie, monitorowanie zasobów.

Sprawdź to sam

37. Stwórz `hello.sh`:

```
#!/bin/bash
echo "Witaj, $1"
```

Daj `chmod +x hello.sh`. Uruchom bez argumentu i z `./hello.sh Kasia`.

38. Zapisz jako `licznik.sh`:

```
#!/bin/bash
for i in {1..5}; do echo "Licznik: $i"; done
```

Uruchom. Zmień zakres na `{1..10}`.

39. Uruchom `bash -x hello.sh "test"` — zobacz debug krok po kroku.

40. Napisz skrypt, który przyjmuje nazwę pliku i sprawdza, czy istnieje. Jeśli nie, wyświetla komunikat i kończy się kodem błędu (`exit 1`).

SSH — cudze maszyny, własne palce

Są dwa komputery. Jeden stoi na biurku, drugi gdzieś w chmurze albo w piwnicy serwerowni. Żeby coś na nim zrobić, trzeba się do niego dostać. Nie ma monitora, nie ma myszy. Jest tylko sieć i protokół, który robi za oczy i ręce.

SSH (Secure Shell) to szyfrowane połączenie z innym komputerem. Terminal na lokalnej maszynie, komenda, i nagle echo wraca z tamtego komputera. Każdy wysłany znak jest szyfrowany. Każda odpowiedź, która wraca, też. Ktoś pomiędzy nie zobaczy ani hasła, ani pliku, ani komendy.

POZIOM 1

Czym jest SSH

SSH to protokół. Działa na porcie 22. Używa kryptografii klucza publicznego — maszyna klienta i serwer uzgadniają szyfr, zanim wyślą choćby znak danych. Standardowo stosuje się algorytmy RSA (2048 lub 4096 bitów) albo **Ed25519** (krzywa Edwardsa — daje krótsze klucze, jest szybszy i bezpieczniejszy). Ten drugi jest dziś zalecany.

Do czego służy? Do zdalnego terminala. `ls` wpisane lokalnie, a lista plików pochodzi z tamtego komputera. Skrypt uruchomiony zdalnie, konfiguracja edytowana, serwis zrestartowany — wszystko jak przy własnym monitorze.

Łączenie

Schemat jest jeden:

```
ssh user@adres
```

`user` to nazwa konta na zdalnej maszynie. `adres` to IP lub nazwa domeny.

```
ssh root@192.168.1.100
```

Po naciśnięciu Enter klient SSH łączy się na porcie 22. Serwer odpowiada komunikatem:

```
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
ED25519 key fingerprint is SHA256:xyz...
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

To **fingerprint klucza hosta** — odcisk klucza serwera. Należy go sprawdzić z tym, który podał administrator. Jeśli się zgadza, wpisuje się `yes`. Klient zapamiętuje klucz w `~/.ssh/known_hosts`. Przy następnym połączeniu porównuje go automatycznie. Jeśli klucz się zmieni — ostrzeże. To zabezpieczenie przed atakiem Man-in-the-Middle.

Jeśli serwer słucha na niestandardowym porcie, dodaj `-p`:

```
ssh -p 2222 user@server.example.com
```

Po połączeniu podaje się hasło. Ale jest lepszy sposób — klucze SSH.

Klucze SSH (zamiast hasła)

Hasło można podejrzeć, odgadnąć, przechwycić. Klucz — nie. To para plików: jeden prywatny (przechowywany tylko u siebie), drugi publiczny (trafia na serwer). Działają jak zamek i klucz. Serwer szyfruje wyzwanie kluczem publicznym, klient odszyfrowuje prywatnym. Jeśli pasuje — wpuszcza bez hasła.

Generuje się je tak:

```
ssh-keygen -t ed25519
```

Program zapyta o lokalizację (domyślnie `~/.ssh/id_ed25519`) i hasło chroniące klucz — passphrase (można zostawić puste, ale bezpieczniej je wpisać). Powstają dwa pliki:

- `~/.ssh/id_ed25519` — klucz prywatny. **Nikt poza właścicielem nie może go czytać.** Zabezpiecz go: `chmod 600 ~/.ssh/id_ed25519`.
- `~/.ssh/id_ed25519.pub` — klucz publiczny. Można go pokazywać, wysyłać, wklejać.

Żeby serwer rozpoznawał klienta, trzeba mu podrzucić klucz publiczny. Ręczne wklejanie do `~/.ssh/authorized_keys` działa, ale jest powolne. Szybsza droga:

```
ssh-copy-id user@192.168.1.100
```

Hasło podaje się ostatni raz. Klient kopiuje klucz publiczny na serwer i dodaje go do pliku autoryzowanych kluczy. Od tej pory połączenia działają bez hasła.

Pliki przez SSH

SSH służy też do przesyłania plików. Oto dwie komendy, które przydają się codziennie.

scp (Secure Copy) — kopia pliku:

```
scp dokument.txt user@192.168.1.100:~/
```

Wysyła `dokument.txt` do katalogu domowego użytkownika na serwerze. W drugą stronę:

```
scp user@192.168.1.100:~/dokument.txt .
```

Kropka oznacza bieżący katalog. Plik trafia z serwera na lokalną maszynę.

rsync — synchronizacja. Działa jak scp, ale porównuje pliki i przesyła tylko różnice:

```
rsync -avz folder/ user@192.168.1.100:~/folder/
```

`-a` zachowuje uprawnienia i daty, `-v` pokazuje postęp, `-z` kompresuje dane. Przy drugim uruchomieniu przesyła tylko nowe i zmienione pliki.

POZIOM 2

~/.ssh/config

Wpisywanie `ssh user@192.168.1.100 -p 2222` za każdym razem męczy. Plik `~/.ssh/config` rozwiązuje ten problem. Definiuje się aliasy:

```
Host serwer
    HostName 192.168.1.100
    User root
    Port 2222
    IdentityFile ~/.ssh/id_ed25519
```

Potem wystarczy:

```
ssh serwer
```

Resztę klient czyta z konfiguracji. Można mieć dziesiątki wpisów — każdy z własnym adresem, portem, kluczem i tunelem.

Tunelowanie

SSH to także tunel. Przekierowuje port z lokalnej maszyny na zdalną i odwrotnie. Przydaje się, gdy potrzebny jest dostęp do panelu administracyjnego, który słucha tylko na localhost serwera.

```
ssh -L 8080:localhost:80 user@server
```

Przekierowuje port 8080 na lokalnej maszynie na port 80 na serwerze. Przeglądarka pod `http://localhost:8080` pokazuje to, co serwer ma na porcie 80. Bez tunelu SSH to połączenie byłoby zablokowane.

SSH na Windows

Do niedawna Windows potrzebował osobnego programu (PuTTY). Od Windows 10 wersja 1809 OpenSSH jest dostępny jako funkcja opcjonalna. Włącza się go przez Ustawienia → Funkcje opcjonalne → Dodaj funkcję → OpenSSH Client. Działa w CMD, PowerShell i Terminalu Windows. Wszystkie komendy z tego rozdziału (`ssh`, `scp`, `ssh-keygen`) są dostępne od ręki.

Agent forwarding

Wyobraźmy sobie: połączenie z maszyną A, a z niej skok do maszyny B. Żeby nie kopiować klucza prywatnego na A (zła praktyka), używa się przekazywania agenta. Lokalna maszyna trzyma klucz, a maszyna A tylko przekazuje zapytania do agenta na lokalnej maszynie.

```
ssh -A user@maszyna-a
```

Potem z maszyny A łączy się do B:

```
ssh user@maszyna-b
```

Serwer B widzi klienta z A, ale autoryzacja przechodzi przez lokalną maszynę. Klucz prywatny nigdy jej nie opuszcza.

Uwaga: przełącznik `-A` otwiera drzwi. Jeśli ktoś przejmie maszynę A, będzie mógł używać agenta. Należy używać tylko do zaufanych maszyn.

SSH do GitHuba — pierwszy prawdziwy przykład

Większość ludzi pierwszy raz używa SSH nie do serwera, a do GitHuba. Zamiast loginu i hasła przy każdym pushu używa się klucza.

Krok 1 — wygenerowanie klucza (jeśli nie ma):

```
ssh-keygen -t ed25519 -C "twoj@email.com"
```

Podmienia się `twoj@email.com` na swój adres email. Passphrase można zostawić pusty (Enter) albo wpisać hasło chroniące klucz.

Krok 2 — skopiuj klucz publiczny:

```
cat ~/.ssh/id_ed25519.pub
```

Zaznacza się cały output (od `ssh-ed25519` do końca) i kopiuje do schowka.

Krok 3 — dodaj klucz na GitHubie:

Otwórz github.com/settings/keys. Kliknij "New SSH key". Wklej klucz. Zapisz.

Krok 4 — przetestuj połączenie:

```
ssh -T git@github.com
```

Wynik: `Hi twojanazwa! You've successfully authenticated.` Gotowe.

Teraz repozytoria klonuje się przez SSH:

```
git clone git@github.com:uzytkownik/repozytorium.git
```

Żadnych haseł. Klucz załatwia sprawę.

Praktyczne scenariusze

Scenariusz: połącz się z Raspberry Pi

Raspberry Pi (lub inny komputer z Linuxem) w sieci lokalnej. Połączenie przez SSH — bez monitora, bez klawiatury, bez myszy.

Krok 1 — znajdź adres IP Raspberry Pi

Jeśli adres Pi nie jest znany, sprawdza się router (192.168.1.1 w przeglądarce) albo używa skanowania sieci:

```
nmap -sn 192.168.1.0/24
```

Szukaj hosta z otwartym portem 22 i nazwą `raspberrypi`.

Krok 2 — połącz się (domyślny login i hasło):

```
ssh pi@192.168.1.XXX
```

Domyślne hasło: **raspberry**. Zmień je od razu po pierwszym logowaniu:

```
passwd
```

Krok 3 — skonfiguruj Raspberry Pi bez monitora (headless setup):

Przed pierwszym uruchomieniem utwórz na karcie SD plik `ssh` (pusty, bez rozszerzenia) w partycji boot. To włącza serwer SSH.

Krok 4 — skopiuj klucz publiczny, żeby nie wpisywać hasła:

```
ssh-copy-id pi@192.168.1.XXX
```

Od tej pory logowanie odbywa się bez hasła. *To samo robi się na serwerach VPS, GitHub i GitHub Actions.*

Krok 5 — przydatne komendy na Raspberry Pi:

```
vcgencmd measure_temp        # temperatura procesora
vcgencmd get_throttled       # czy Pi dławi wydajność
free -h                      # pamięć RAM
df -h /                      # miejsce na karcie SD
htop                         # menedżer procesów
sudo apt update && sudo apt upgrade -y  # aktualizacja systemu
```

Raspberry Pi to dobry serwer do nauki: można na nim postawić Nginx, Pi-hole (blokada reklam w sieci) albo własne repozytorium Gita.

Sprawdź to sam

41. `ssh-keygen -t ed25519` — wygenerowanie pary kluczy (jeśli jeszcze nie ma).
42. `ls -la ~/.ssh` — podgląd kluczy. Są tam `id_ed25519` i `id_ed25519.pub`? Jeśli są też stare klucze RSA, również będą widoczne.
43. Jeśli jest serwer albo drugi komputer w sieci: `ssh localhost` — połączenie z samym sobą. Wymaga uruchomionego serwera SSH. Na Windows włącza się OpenSSH Server w funkcjach opcjonalnych. Na Macu — "Ustawienia systemowe" → "Ogólne" → "Udostępnianie" → włącza się "Zdalne logowanie".

Git — wehikuł czasu dla plików

Są trzy pliki. Projekt, nad którym ktoś siedział dwa tygodnie. Nagle jeden znika — przypadkowe usunięcie, nadpisanie, pomyłka. Bez Gita to godziny pracy do odtworzenia z pamięci, jeśli w ogóle. Z Gitem — jedno polecenie i plik wraca sprzed tygodnia. Git to nie fanaberia programistów. To wehikuł czasu dla każdego, kto pracuje z plikami tekstowymi.

Git śledzi każdą zmianę. Pamięta, kto i kiedy zmienił który plik. Do dowolnego momentu historii można wrócić. Ten rozdział pokazuje tyle, ile trzeba, żeby zacząć. Git działa lokalnie — GitHub to opcjonalny dodatek, który pełni rolę zdalnego backupu i narzędzia do współpracy.

POZIOM 1

Czym jest Git

Git to system kontroli wersji. Każda zmiana w plikach zostaje zapamiętana. Można zobaczyć, co dokładnie się zmieniło między wersjami. Można cofnąć się do dowolnego punktu w historii.

Różni się od zwykłej kopii zapasowej. Kopia zapasowa to seria zdjęć — widać, co było, ale nie widać, co zmieniło się między zdjęciami. Git działa jak dziennik zmian. Dla każdej linijki tekstu widać, kiedy została dodana, zmodyfikowana albo usunięta.

Git został stworzony przez Linusa Torvaldsa w 2005 roku do zarządzania kodem źródłowym jądra Linux. Do dziś pozostaje najpopularniejszym systemem kontroli wersji. Każdy projekt open source na GitHubie go używa.

Pierwsze repozytorium

Repozytorium to projekt objęty kontrolą wersji. Żeby zacząć, wystarczy jeden plik i jedna komenda.

Zainicjuj

```
cd ~/projekt
git init
```

To tworzy ukryty katalog `.git` — tu Git trzyma całą historię. Reszta folderów zostaje nietknięta.

Dodaj plik do śledzenia

```
git add README.md
git add .    # wszystkie pliki w bieżącym katalogu
```

Polecenie `git add` przenosi pliki do *staging area* — obszaru przejściowego. To lista zmian przygotowanych do zatwierdzenia.

Zatwierdź zmianę

```
git commit -m 'Pierwszy commit: dodano README'
```

`commit` to punkt kontrolny. Można do niego wrócić w każdej chwili. Każdy commit ma swój niepowtarzalny identyfikator (hash), autora, datę i opis. `-m` to skrót od `--message` — treść opisu.

Codzienna praca

Sprawdź status

```
git status
```

Pokazuje, które pliki są zmienione, które nowe, które gotowe do commita. To najczęściej używane polecenie Gita.

Zobacz, co się zmieniło

```
git diff
```

Pokazuje dokładnie, które linijki zostały dodane (zielone) i usunięte (czerwone). Przydaje się przed commitem, żeby sprawdzić, czy zmiany są prawidłowe.

Historia

```
git log --oneline
```

Lista commitów — każdy z identyfikatorem (hash), autorem, datą i opisem. Bez --oneline log pokazuje pełne informacje: hash, autora, datę, cały komunikat.

Cofnij ostatnią zmianę — jeszcze przed commitem

```
git checkout -- README.md
```

Przywraca plik do stanu z ostatniego commita. **Uwaga**: nie da się tego cofnąć. Git nie przechowuje zmian, które nie trafiły do staging area lub commita.

Cofnij ostatniego commita

```
git reset --soft HEAD~1
```

`--soft` — cofa commita, ale zostawia zmiany w plikach. Można je poprawić i zrobić nowy commit. `--hard` — cofa commita i usuwa zmiany. Nieodwracalne. `HEAD~1` oznacza "jeden commit przed bieżącym". `HEAD~2` cofnie o dwa.

GitHub — zdalne repozytorium

Git działa w pełni lokalnie. GitHub to serwer, który przechowuje kopię repozytorium. Pozwala synchronizować pracę między różnymi maszynami i współpracować z innymi osobami.

Połącz lokalne repozytorium z GitHub

```
git remote add origin git@github.com:twojanazwa/projekt.git
git branch -M main
git push -u origin main
```

Do zdalnych repozytoriów potrzebny jest klucz SSH. `git push` wysyła wszystkie commity na GitHub. `-u` (--set-upstream) zapamiętuje, że gałąź main ma domyślnie odpowiadać origin/main. Potem wystarczy samo `git push`.

Pobierz zmiany z GitHub

```
git pull
```

Pull ściąga zmiany z serwera i scala je z lokalnymi. **Przy pracy na kilku maszynach**: pull przed pracą, push po pracy.

Jeśli na serwerze i lokalnie zmieniono ten sam fragment pliku — Git zgłosi konflikt. Wymaga ręcznego rozwiązania, ale zdarza się rzadko przy pracy jednoosobowej.

POZIOM 2

Gałęzie (branches)

Gałąź to osobna linia rozwoju. Można eksperymentować bez ryzyka — zmiany w gałęzi nie wpływają na główną linię, dopóki nie zostaną scalone.

`git branch` pokazuje listę gałęzi. Gwiazdka oznacza bieżącą.

Nowa gałąź:

```
git branch eksperyment
git checkout eksperyment
# skrót: git checkout -b eksperyment
```

Po zmianach i commitach na gałęzi eksperyment — powrót do main i scalenie:

```
git checkout main
git merge eksperyment
```

Scalenie (merge) przenosi zmiany z gałęzi eksperyment do main. Git radzi sobie z tym automatycznie, gdy zmiany nie nachodzą na siebie. W przeciwnym razie trzeba pomóc ręcznie.

Po scaleniu gałąź można usunąć:

```
git branch -d eksperyment
```

.gitignore — czego nie śledzić

Plik `.gitignore` w katalogu głównym repozytorium mówi Gitowi, które pliki i foldery ignorować. Hasła, klucze API, pliki tymczasowe, katalog `node_modules` — to wszystko powinno trafić do .gitignore, żeby przypadkiem nie wyciekło.

Przykładowy .gitignore:

```
# Hasła i konfiguracja lokalna
.env
config.local.ini


# Pliki tymczasowe
*.log
*.tmp


# Katalog zależności
node_modules/
```

Wzorce działają podobnie do glob: `*` oznacza dowolny ciąg znaków, `katalog/` ignoruje cały katalog. Plik .gitignore sam podlega kontroli wersji — commit go śledzi.

Konfiguracja i aliasy

Git ma trzy poziomy konfiguracji: systemowy, globalny (bieżący użytkownik) i lokalny (repozytorium). Najczęściej używa się globalnego:

```
git config --global user.name 'Twoje Imię'
git config --global user.email 'twoj@email.com'
```

Bez tego Git nie pozwoli zrobić commita. Nazwa i email trafiają do każdego commita jako autor.

Aliasy skracają często używane komendy:

```
git config --global alias.st status
git config --global alias.ci commit
git config --global alias.co checkout
git config --global alias.br branch
```

Potem wystarczy `git st` zamiast `git status`, `git ci -m 'opis'` zamiast `git commit -m 'opis'`.

Szybki przepis: backup konfiguracji

Pliki konfiguracyjne (.bashrc, settings.json, notatki) — można je śledzić w Git i mieć kopię na GitHubie. Oto cały przepis od początku do końca:

```
cd ~
git init
git add .bashrc .gitconfig
git commit -m 'Backup konfiguracji z 2026-05-31'
git remote add origin git@github.com:twojanazwa/dotfiles.git
git push -u origin main
```

Na GitHubie trzeba wcześniej utworzyć puste repozytorium o nazwie `dotfiles`. Resztę załatwia kilka komend. Późniejsza aktualizacja to tylko `git add . && git commit -m 'opis' && git push`.

Czego nie robić

`git commit` bez `-m` — otworzy edytor (zwykle vi). Bez znajomości vi można w nim utknąć. Zawsze używaj `-m "opis"`.

`git push` bez `git pull` przed — jeśli ktoś wypchnął zmiany na serwer, push zostanie odrzucony. Najpierw pull, potem push.

`git add .` bez zastanowienia — doda wszystko, łącznie z plikami z hasłami, kluczami API, plikami tymczasowymi. Używaj `.gitignore` i sprawdzaj `git status` przed każdym dodaniem.

`git reset --hard` bez sprawdzenia — usuwa zmiany bez ostrzeżenia. Najpierw `git status`, potem reset.

Sprawdź to sam

44. Zainicjuj repozytorium w nowym folderze. Stwórz plik README.md z opisem. Wykonaj `git add` i `git commit`.
45. Zmień plik. Uruchom `git diff`. Zobacz, co się zmieniło.
46. Otwórz `git log --oneline`. Zobacz historię.

47. Stwórz nową gałąź (`git checkout -b test`). Zrób commit. Wróć do main (`git checkout main`). Zauważ, że zmiany zniknęły. Scal gałąź (`git merge test`).
48. Skonfiguruj alias: `git config --global alias.lg 'log --oneline --graph'`. Uruchom `git lg`.

Linux wewnątrz Windows

Terminal Windows. CMD działa, PowerShell działa. Tu nagle okazuje się, że program istnieje tylko na Linuksie. `grep` z flagą `-P` dla perl-compatible regex. `jq` do JSON. `rsync` do synchronizacji plików. Albo serwer Node.js, który ma działać w środowisku identycznym z produkcyjnym.

Do tej pory były trzy opcje: maszyna wirtualna (VirtualBox, VMware), dual boot (przeładowanie systemu) albo **WSL** — *Windows Subsystem for Linux*.

WSL to nakładka, która uruchamia Linuksa bezpośrednio w Windows. Bez wirtualnych pulpitów, bez przeładowywania komputera, bez konfiguracji sieci. Komenda `wsl` otwiera terminal Linuksa w oknie Windows.

POZIOM 1

Czym jest WSL

WSL (*Windows Subsystem for Linux*) to komponent systemu Windows, który pozwala uruchomić linuksową dystrybucję (Ubuntu, Debian, Fedora, Kali i inne) jako aplikację Windows. Nie wymaga do tego maszyny wirtualnej ani drugiego dysku.

Microsoft wypuścił WSL 1 w 2016 roku. Działał przez tłumaczenie wywołań systemowych Linuksa na wywołania Windows. W 2019 roku pojawił się **WSL 2**, który zamiast tego uruchamia prawdziwe jądro Linuksa w lekkiej maszynie wirtualnej. Różnica jest fundamentalna — WSL 2 ma pełną kompatybilność z jądrem Linuksa, przez co działa szybciej przy operacjach na plikach i obsługuje więcej programów.

Instalacja

Otwórz PowerShell lub CMD jako administrator i wpisz:

```
wsl --install
```

To jedno polecenie robi cztery rzeczy: włącza wymagane składniki Windows, pobiera jądro WSL 2, ustawia WSL 2 jako domyślny i instaluje domyślną dystrybucję (Ubuntu). Po restarcie komputera Linux jest gotowy do pracy.

> Linux nie zaczął się jako produkt korporacji. W 1991 roku Linus Torvalds, student w Helsinkach, miał prozaiczny problem: system, którego używał, był dla niego zbyt zamknięty.
> Nie planował rewolucji. Chciał mieć coś, co działa po jego myśli. Opublikował kod w internecie, a inni zaczęli go rozwijać razem z nim.
> Linux nie powstał jako konkurencja dla Windows. Powstał dlatego, że ktoś nie chciał czekać na cudze decyzje.

Minimalne wymagania: Windows 10 w wersji 2004 (build 19041) lub nowszy, albo dowolna wersja Windows 11. Wymagany jest też procesor z obsługą wirtualizacji (SLAT). Większość komputerów z ostatnich 7–8 lat to spełnia.

Dostępne dystrybucje sprawdza się tak:

```
wsl --list --online
```

Lista zawiera: Ubuntu, Debian, Kali Linux, Fedora, Alpine, openSUSE i inne. Aby zainstalować konkretną:

```
wsl --install -d Debian
```

WSL instaluje się raz. Potem można dodać dowolną liczbę dystrybucji i przełączać się między nimi.

Używanie WSL

Po instalacji Ubuntu (lub wybrana dystrybucja) pojawia się w Menu Start. Pierwsze uruchomienie poprosi o nazwę użytkownika i hasło — to konto linuksowe, niezależne od konta Windows.

Można też wejść do Linuksa z dowolnego terminala Windows:

```
wsl
```

Po wpisaniu polecenia uruchamia się shell Linuksa (domyślnie bash). Działa wszystko: `ls`, `cd`, `grep`, `awk`, `sed`, `ssh`, `apt`, `git`. To jest system Linux.

Można też wykonać pojedyncze polecenie bez wchodzenia do shell:

```
wsl -- ls -la /home
```

To przydaje się w skryptach — nie trzeba wchodzić do WSL, żeby coś sprawdzić lub uruchomić.

Pliki Windows są dostępne w WSL pod ścieżką `/mnt/c/Users/twojanazwa`. Dysk C: to `/mnt/c`, dysk D: to `/mnt/d`. Ale WSL ostrzega: operacje na plikach Windows z poziomu WSL są wolniejsze niż na plikach linuksowych. Projekty należy trzymać w katalogu domowym WSL (`/home/twoja_nazwa/`), a do Windows wrzucać tylko gotowe pliki.

Aby otworzyć bieżący katalog w Eksploratorze Windows:

```
explorer.exe .
```

Działa — WSL może wywoływać programy Windows, a Windows może wywoływać programy WSL. Most działa w obie strony.

Integracja z Windows

WSL nie jest odizolowany. Programy Windows widzą sieć WSL i odwrotnie — `localhost` działa w obie strony. Serwer uruchomiony w WSL na porcie 3000 otwiera się w przeglądarce Windows pod `http://localhost:3000`.

Dla użytkowników Visual Studio Code dostępne jest rozszerzenie **Remote — WSL** (kod: `ms-vscode-remote.remote-wsl`). Z poziomu WSL można wtedy wpisać:

```
code .
```

i VS Code otworzy się w trybie WSL — widzi pliki linuksowe, używa linuksowego shella w terminalu, instaluje rozszerzenia po stronie WSL. Dla programisty to najwygodniejszy sposób pracy: edytor w Windows, środowisko w Linux.

Można też mieszać polecenia Windows i Linux w jednym skrypcie. W PowerShell działa `wsl -- grep -r 'error' /var/log`, w bashu WSL uruchamia się `powershell.exe -Command Get-Process`. Komunikacja przez stdin/stdout działa.

POZIOM 2

WSL 1 vs WSL 2 — który wybrać

WSL 2 używa pełnego jądra Linuksa uruchomionego w lekkiej maszynie wirtualnej zarządzanej przez Hyper-V. To daje pełną kompatybilność syscalli — programy linuksowe działają w WSL 2 dokładnie tak jak na fizycznym Linuksie.

WSL 1 działał przez tłumaczenie syscalli („babel fish" między jądrami). Był szybszy przy starcie i używał mniej RAM, ale nie obsługiwał wszystkich funkcji jądra. Nie działały w nim FUSE, Docker (choć dało się go uruchomić przez obejścia), ani programy wymagające specyficznych syscalli.

W praktyce: dla początkujących z WSL odpowiedni jest WSL 2. Jest domyślną wersją od Windows 10 2004. Przy starszym sprzęcie bez obsługi wirtualizacji SLAT pozostaje WSL 1.

Wersję sprawdza się:

```
wsl --list --verbose
```

Wersję zmienia się dla konkretnej dystrybucji:

```
wsl --set-version Ubuntu 2
```

Obsługa usług

Od wersji 0.67.6 (wrzesień 2022) można włączyć **systemd** — menedżer systemu i usług standardowy w większości dystrybucji Linuksa. Bez niego WSL uruchamiał tylko init o minimalnej funkcjonalności. Z systemd dostępne są `systemctl`, `journalctl`, automatyczne uruchamianie usług.

Aby włączyć systemd, edytuj plik `/etc/wsl.conf` w dystrybucji WSL:

```
[boot]
systemd=true
```

Potem zrestartuj WSL:

```
wsl --shutdown
wsl
```

Od tej pory `systemctl list-units` pokaże wszystkie usługi. Działa `snapd`, działają serwery baz danych jako demony.

Zarządzanie zasobami

WSL 2 zużywa RAM — domyślnie może wykorzystać do 50% dostępnej pamięci systemu lub 8 GB (zależnie od wersji). Gdy WSL nie jest potrzebny, wyłącza się go:

```
wsl --shutdown
```

To zwalnia całą pamięć zajmowaną przez lekką maszynę wirtualną WSL 2. Nie trzeba czekać, aż WSL sam się zamknie.

Można też ograniczyć maksymalny RAM dla WSL. W tym celu tworzy się plik `%USERPROFILE%\.wslconfig`:

```
[wsl2]
memory=4GB
processors=2
swap=2GB
```

Po zapisie: `wsl --shutdown` i nowe uruchomienie WSL użyje tylko 4 GB RAM, 2 rdzenie procesora i 2 GB swapu.

Przenoszenie dystrybucji

Domyślnie WSL instaluje dystrybucje na dysku systemowym (C:). Gdy kończy się miejsce, przenosi się dystrybucję:

```
wsl --export Ubuntu D:\wsl_backup\ubuntu.tar
wsl --unregister Ubuntu
wsl --import Ubuntu D:\wsl\ubuntu\ D:\wsl_backup\ubuntu.tar
```

Pierwsze polecenie tworzy archiwum dystrybucji. Drugie usuwa oryginał. Trzecie importuje dystrybucję na dysk D.

Po imporcie ustawia się domyślnego użytkownika (bo zostaje zgubiony):

```
ubuntu config --default-user twoja_nazwa
```

Adresy IP i sieć

WSL 2 używa translacji adresów (NAT) dla swojej maszyny wirtualnej. Oznacza to, że adres IP WSL zmienia się po każdym restarcie. Nie warto konfigurować stałych adresów IP — `localhost` w Windows przekierowuje na WSL automatycznie. Dla aplikacji webowych używa się `localhost` i IP nie ma znaczenia.

Gdy potrzebne jest połączenie z WSL z innych komputerów w sieci, stosuje się port forwarding w PowerShell (z uprawnieniami administratora):

```
netsh interface portproxy add v4tov4 listenaddress=0.0.0.0 listenport=3000
connectaddress=127.0.0.1 connectport=3000
```

Gdzie są pliki WSL?

To pytanie zadaje każdy początkujący. Ubuntu jest zainstalowane w WSL. Gdzie fizycznie leżą pliki?

Z Windows do WSL: W Eksploratorze Windows wpisz w pasku adresu:

```
\\wsl$
```

Pojawia się lista dystrybucji. Po kliknięciu `Ubuntu` widać cały system plików Linuksa: `/home`, `/etc`, `/var` — wszystko.

Można też otworzyć katalog domowy WSL bezpośrednio:

```
\\wsl$\Ubuntu\home\twojanazwa
```

Z WSL do Windows: Dyski Windows są zamontowane w `/mnt/`.

```
$ cd /mnt/c
$ ls
Program Files   Users   Windows   ...
```

/mnt/c to dysk C:. **/mnt/d** to dysk D:. Działa pełne czytanie i pisanie.

Ważna zasada: pliki należy trzymać w systemie, w którym są używane. Przy pracy w WSL pliki trzyma się w katalogu WSL `~/projekty`. Jeśli potrzebne są z Windows, kopiuje się przez `\\wsl$`. Operacje na plikach Windows z WSL (`/mnt/c`) są zauważalnie wolniejsze.

Sprawdź to sam

49. Otwórz PowerShell i wpisz `wsl --list --verbose`. Wynik pokaże, czy WSL jest zainstalowany, w jakiej wersji i jakie dystrybucje.
50. Jeśli WSL nie jest zainstalowany: `wsl --install`. Po restarcie uruchom Ubuntu z Menu Start.

51. W shellu WSL wpisz: `lsb_release -a` (wynik pokaże wersję dystrybucji), potem `exit` (powrót do Windows).

52. Przetestuj most: w WSL wpisz `explorer.exe .` — otworzy się Eksplorator w bieżącym katalogu.

53. Stwórz plik w katalogu domowym WSL: `touch ~/test.txt`, sprawdź, czy plik jest widoczny w Windows pod `\\wsl$\Ubuntu\home\twoja_nazwa\test.txt`.

Co dalej — gdzie iść z terminalem

CMD, PowerShell, bash na Macu, SSH, WSL. Nawigacja po dysku, wyszukiwanie plików, diagnoza sieci, skrypty, instalacja bez myszy. Każdy rozdział tej książki dokładał kolejną warstwę.

To dopiero początek. Terminal to nie tylko administracja — to codzienne środowisko programistów, inżynierów i wszystkich, którzy wolą wydawać polecenia zamiast klikać w domysły.

Poniżej siedem ścieżek, które naturalnie wynikają z tej wiedzy. Każda to konkretna umiejętność do dodania do warsztatu.

Git — wersjonowanie i historia

Git jest niezbędny przy pracy z plikami tekstowymi — kodem, konfiguracją, dokumentacją.

Każdy plik to linia czasu. Można wrócić do dowolnej wersji, zobaczyć kto zmienił co i kiedy, łączyć zmiany z innymi osobami.

Git to rozwinięcie pomysłu z backupu (Scenariusz w Rozdz. 3) **w system**: nie tylko kopiowanie plików, ale zapamiętywanie każdej zmiany z opisem.

Od czego zacząć:

- git init, git add, git commit — trzy komendy na start
- git log — przeglądaj historię
- git diff — zobacz co się zmieniło
- GitHub — zdalne repozytorium, backup i współpraca

Polecam: `git-scm.com/book/pl` — Pro Git po polsku, za darmo.

Docker — aplikacje w kontenerach

Docker pozwala uruchomić aplikację w izolowanym środowisku — bez instalowania bibliotek systemowych, bez konfliktów wersji.

"Na moim komputerze działa" to problem, który Docker rozwiązuje raz na zawsze.

Znajomość terminala z tej książki jest wystarczająca, by uruchomić pierwszy kontener:

```
docker run -it ubuntu bash
```

To polecenie ściąga Ubuntu i otwiera bash. Świeży system, bez instalacji, bez ryzyka.

Co dalej:

- docker pull, docker run, docker ps — podstawy
- Dockerfile — definiuj własne środowisko
- docker-compose — uruchom kilka kontenerów naraz

Python — skrypty, które myślą

PowerShell i bash wystarczają do komend. Do prawdziwych programów potrzebny jest język programowania. Python jest naturalnym następcą.

Podstawy są już znane: zmienne, pętle, warunki — z Rozdz. 5 i 11. Python działa podobnie, ale ma więcej mocy:

- Przetwarzanie plików CSV, JSON, Excel
- Automatyzacja stron internetowych (Selenium)
- Analiza danych (pandas)
- Skrypty do backupu, raportów, monitorowania

Jak zacząć:

```
python --version  # sprawdź czy masz
python -c "print('Hello, terminal')"
```

Linux — prawdziwa moc terminala

Windows z WSL (Rozdz. 12) to przedsmak. Pełny Linux — Ubuntu, Debian, Fedora — to system, w którym terminal jest pierwszym obywatelem.

Na Linuxie wszystkie komendy z tej książki działają bez modyfikacji. Dodatkowo dochodzą:

- systemd — zarządzanie usługami (systemctl)
- cron — planowanie zadań
- logi — journalctl, /var/log
- pakiety — apt, dnf, pacman
- sieć — iptables, firewalld, netstat

Nie trzeba od razu instalować Linuxa na komputerze. **Wystarczy maszyna wirtualna (VirtualBox) albo Raspberry Pi.**

Automatyzacja — ręcznie tylko gdy trzeba

Skrypty z Rozdz. 5 i 11 to dopiero wstęp. Prawdziwa automatyzacja to:

- Cron (Linux/macOS) — harmonogram zadań
- Task Scheduler (Windows) — odpowiednik crona
- Watchdog — automatyczne reagowanie na zmiany plików
- CI/CD — GitHub Actions, Jenkins — automatyzacja testów i wdrożeń

Przykład: codzienny backup o 3:00 nad ranem, wysłany na zewnętrzny serwer. Raz napisany, działa latami.

Serwery — komputer w internecie

SSH (Rozdz. 13) otworzył drzwi do zdalnej administracji. Co dalej?

- Nginx lub Apache — postaw własną stronę
- Let's Encrypt — darmowy certyfikat HTTPS
- Fail2ban — ochrona przed atakami
- Docker na serwerze — izolowane aplikacje
- Monitoring — htop, nload, netdata

Najtańszy serwer VPS kosztuje kilkanaście złotych miesięcznie. Za tyle — własny serwer w internecie, zarządzany przez terminal.

Polecam: **DigitalOcean, Linode, Vultr** — mają tutoriale krok po kroku i darmowe kredyty na start.

AI w terminalu — nowa granica

Modele językowe (jak ten, który napisał część tej książki) działają z linii poleceń. Bez przeglądarki, bez GUI.

- ShellGPT — zadawaj pytania po polsku, dostawaj komendy
- Ollama — uruchom model lokalnie, bez internetu
- GitHub Copilot w CLI — podpowiada komendy w terminalu
- AI-powered skrypty — generuj kod z opisu

Przykład:

```
sgpt "znajdź największe pliki w bieżącym folderze"
# ShellGPT odpowiada: du -sh * | sort -rh | head -10
```

To nie zastąpienie nauki — to przyspieszenie. *Im więcej się rozumie, tym lepsze pytania się zadaje.*

Jedna rada na koniec

Nikt nie opanowuje wszystkiego naraz. Wystarczy jedna ścieżka na miesiąc.

Pliki — Git. Automatyzacja — Python. Sieć — Docker i serwery. System od środka — Linux.

Terminal to nie narzędzie. To sposób myślenia.

Zaczyna się od jednej komendy. Każdy był kiedyś początkujący. Nawet ten, który dziś pisze skrypty w śnie.

Linux i Windows różnią się nie komendami. Różnią się tym, dla kogo były projektowane na początku.

Linux — dla ludzi, którzy chcą grzebać w środku systemu. Windows — dla ludzi, którzy chcą, żeby system po prostu działał.

Terminal w obu przypadkach jest konsekwencją tego wyboru, a nie jego początkiem.

Koniec książki "Pod skórą systemu"

Dodatek A: Tabela porównawcza (CMD / PowerShell / macOS)

W całej książce pokazywałem ci polecenia dla dwóch światów: Windows i macOS. Tu znajdziesz je wszystkie w jednej tabeli. Nie musisz przerzucać rozdziałów w poszukiwaniu składni — otwierasz ten dodatek, znajdujesz operację, czytasz wiersz.

Kolumna CMD i PowerShell dotyczą Windows — różnica między nimi polega na składni i filozofii. CMD operuje na tekście, PowerShell na obiektach. macOS podaje komendy dla terminala BSD, który wywodzi się w prostej linii z Unixa z Bell Labs. Każda kolumna pokazuje to samo zadanie — wykonane w trzy różne sposoby.

Komendy w pigułce

Operacja	CMD (Windows)	PowerShell (Windows)	Terminal (macOS)
Wyświetl pliki	dir	Get-ChildItem / ls	ls
Zmień katalog	cd	Set-Location / cd	cd
Pokaż ścieżkę	cd (bez arg.)	Get-Location / pwd	pwd
Kopiuj plik	copy	Copy-Item / cp	cp
Przenieś plik	move	Move-Item / mv	mv
Usuń plik	del	Remove-Item / rm	rm
Utwórz katalog	mkdir	New-Item / mkdir	mkdir

Operacja	CMD (Windows)	PowerShell (Windows)	Terminal (macOS)
Usuń katalog	rmdir /s	Remove-Item -Recurse	rm -r
Pokaż zawartość pliku	type	Get-Content / cat	cat
Edytuj plik	notepad	notepad	nano (lub vim)
Szukaj w plikach	findstr	Select-String / sls	grep
Zmień uprawnienia	icacls	icacls	chmod
Zmień właściciela	takeown / icacls	takeown	chown
Administrator	Uruchom jako admin	Uruchom jako admin	sudo
Dokumentacja	help	Get-Help	man
Gdzie jest program	where	Get-Command	command -v
Procesy	tasklist	Get-Process	ps aux
Zabij proces	taskkill	Stop-Process	kill
Sieć (ping)	ping	Test-NetConnection	ping
Sieć (adres IP)	ipconfig	Get-NetIPAddress	ifconfig
Sieć (traceroute)	tracert	Test-NetConnection -TraceRoute	traceroute
Data	date /t	Get-Date	date
Historia komend	doskey /history	Get-History	history
Wyczyść ekran	cls	Clear-Host / cls	clear
Zainstaluj program	winget install nazwa	winget install nazwa	brew install nazwa
Sprawdź miejsce na dysku	wmic logicaldisk get size,freespace	Get-PSDrive	df -h

Operacja	CMD (Windows)	PowerShell (Windows)	Terminal (macOS)
Sprawdź kto jest zalogowany	query user	Get-WmiObject Win32_ComputerSystem | select UserName	who
Wyjście	exit	exit	exit

Aliasy — pułapka i ułatwienie

PowerShell rozpoznaje komendy z Unixa (`ls`, `cp`, `mv`, `rm`, `cat`, `pwd`, `clear`). Nie są to jednak te same programy co na macOS. `ls` w PowerShell woła `Get-ChildItem` — funkcję, która zwraca obiekty .NET, nie strumień tekstu. Wynik wygląda podobnie, ale działa inaczej.

Przykład: `ls | Export-Csv pliki.csv` działa w PowerShell, bo `ls` zwraca obiekty z właściwościami (Name, Length, LastWriteTime). Na macOS `ls` zwraca tekst — musisz go przetworzyć inaczej.

CMD nie ma aliasów unixowych. dir to dir, nie ls. Jeśli wpiszesz ls w CMD, dostaniesz błąd.

Uprawnienia — przepaść między Windows a Unix

Operacje na uprawnieniach (`icacls` vs `chmod` / `chown`) to jedne z najbardziej rozbieżnych pozycji w tabeli. Windows zarządza prawami przez listy ACL (Access Control List), które zapisujesz w formie dziedziczenia dla użytkowników i grup. `icacls` modyfikuje te listy.

macOS (Unix) używa bitów uprawnień (rwx dla właściciela, grupy, innych) oraz ACL jako rozszerzenia. chmod 755 plik zmienia bity. chown zmienia właściciela. Żadna z tych komend nie istnieje natywnie na Windows.

Jeśli pracujesz w obu systemach, musisz zapamiętać dwie składnie dla tej samej operacji. To jeden z obszarów, gdzie skrypty nie są przenośne.

Sieć — trzy podejścia

`ping` działa identycznie w CMD, PowerShell i macOS. Ale `ipconfig` nie istnieje na macOS (użyj `ifconfig`), a `ipconfig` w PowerShell to alias do starego programu CMD, nie do cmdletu.

PowerShell ma własne cmdlety sieciowe — `Get-NetIPAddress` daje więcej informacji niż `ipconfig`, ale wymaga modułu NetTCPIP.

Traceroute ma różne nazwy: `tracert` na Windows, `traceroute` na macOS. PowerShell łączy obie funkcje w `Test-NetConnection -TraceRoute` — ale to cmdlet, więc zwraca obiekt, nie tekst.

Administrator i sudo

Na macOS jedno sudo przed komendą podnosi uprawnienia do roota. Na Windows nie ma jednego przycisku — musisz uruchomić cały terminal jako administrator (prawym przyciskiem → Uruchom jako administrator) albo podnieść uprawnienia dla konkretnych operacji przez icacls lub takeown.

PowerShell obsługuje Start-Process -Verb RunAs do uruchomienia procesu z poziomu admina, ale to wymaga odpowiedzi na okno UAC. Żadna z metod nie działa tak bezpośrednio jak sudo.

Gdzie szukać pomocy

Na macOS wpisujesz `man ping` i czytasz manual. W CMD wpisujesz `help` (dla komend wewnętrznych, jak `dir` czy `cd`) albo `ping /?` (dla programów zewnętrznych — `help` ich nie obsługuje). W PowerShell wpisujesz `Get-Help Test-NetConnection`. Każdy system ma inny mechanizm, ale cel ten sam — dokumentacja w terminalu, bez przeglądarki.

CMD zwraca help w konsoli i zamyka się po pierwszej stronie. PowerShell domyślnie pokazuje fragment i zaprasza do przewijania. macOS używa less — przewijasz strzałkami, wychodzisz klawiszem q.

Dodatek B: 40 komend na start

Praca w konsoli ma swoje skróty myślowe. Znasz je, działasz. Nie znasz, tracisz czas. Ta lista zbiera 40 komend, które pokrywają 95% codziennej pracy — od nawigacji po instalowanie oprogramowania.

40 podstawowych komend

Szybka ściąga. Każda komenda ma krótki opis — tyle, żebyś wiedział, kiedy jej użyć.

Podstawy — nawigacja i pliki

ls / `dir` — wypisuje zawartość katalogu. `ls` działa w PowerShell i macOS, `dir` w CMD i PowerShell.

cd — zmienia bieżący katalog. `cd ..` (poziom wyżej), `cd ~` (katalog domowy), ścieżka absolutna.

pwd — wyświetla ścieżkę bieżącego katalogu. Na pytanie "gdzie jestem?" odpowiada: jesteś tu.

```
pwd
```

mkdir — tworzy nowy katalog. `mkdir nazwa` w bieżącym, `mkdir -p sciezka/do/katalogu` tworzy całą ścieżkę naraz.

rmdir — usuwa pusty katalog. Dla niepustego potrzebujesz `rm -r` na macOS lub `rmdir /s` w CMD.

```
rmdir stary_folder
```

cp / `copy` — kopiuje pliki. `cp zrodlo cel` na macOS, `copy zrodlo cel` w CMD, `Copy-Item` w PowerShell.

mv / `move` — przenosi lub zmienia nazwę pliku. Składnia: `mv zrodlo cel`.

rm / `del` — usuwa pliki. `rm` na macOS, `del` w CMD. `rm -r` usuwa cały katalog. Usuniętego nie znajdziesz w koszu.

Eksploracja — podgląd i wyszukiwanie

cat — wyświetla zawartość pliku. Działa na macOS (`cat plik.txt`) i w PowerShell (`Get-Content` lub alias `cat`). W CMD odpowiednikiem jest `type`.

less — stronicuje długi tekst. Przewijasz strzałkami lub spacją, wychodzisz klawiszem `q`. Na macOS wbudowany, na Windows przez Git Bash lub WSL.

head — pokazuje pierwsze 10 linii pliku. `head -n 20 plik` zmienia liczbę linii. W PowerShell: `Get-Content plik -TotalCount 10`.

tail — pokazuje ostatnie 10 linii. `tail -f plik` śledzi dopisywany plik w czasie rzeczywistym — kluczowe przy logach.

find — szuka plików. Na macOS: `find . -name "*.txt"`. Na Windows: `dir /s *.txt` w CMD lub `Get-ChildItem -Recurse` w PowerShell.

grep — szuka tekstu w plikach. `grep "fraza" plik.txt`. Na Windows odpowiednikiem jest `Select-String` w PowerShell lub `findstr` w CMD.

wc — liczy linie, słowa i znaki. `wc -l` liczy tylko linie, `wc -w` tylko słowa. Działa na macOS i w PowerShell.

System — procesy i informacje

ps — wyświetla listę procesów. Na macOS: `ps aux` pokazuje wszystkie procesy. W PowerShell: **Get-Process**. W CMD: `tasklist`.

kill — wysyła sygnał do procesu. `kill 1234` kończy proces o numerze 1234. `kill -9 1234` robi to natychmiast. Na Windows: **taskkill** /PID 1234.

top — monitoruje procesy w czasie rzeczywistym. Na macOS wbudowany. W PowerShell: `Get-Process | Sort-Object CPU -Descending` (jednorazowo).

date — wyświetla bieżącą datę i czas. `date /t` w CMD, `Get-Date` w PowerShell, `date` na macOS.

uptime — pokazuje, jak długo system działa od ostatniego uruchomienia. Działa natywnie na macOS. W PowerShell odpowiednikiem jest **Get-Uptime** (PS 6+). W CMD nie istnieje.

```
uptime
```

whoami — wyświetla nazwę bieżącego użytkownika. Działa identycznie na wszystkich trzech powłokach.

```
whoami
```

uname — informacje o systemie operacyjnym. **uname -a** pokazuje wszystko — jądro, architekturę, wersję. Działa na macOS (zwraca Darwin). Na Windows dostępny przez WSL lub Git Bash. Nie działa w CMD.

Sieć — łączność i transfer

ping — wysyła pakiety ICMP i mierzy czas odpowiedzi. Działa na wszystkich platformach. Na Windows domyślnie wysyła 4 pakiety, na macOS pinguje bez końca.

curl — uniwersalny klient URL. Pobiera strony, wysyła żądania API, testuje punkty końcowe. Wbudowany w Windows (od 2018) i macOS.

wget — pobiera pliki z sieci. Ma prostszą składnię niż curl do zwykłego ściągania. Na macOS wymaga instalacji przez **Homebrew**. Na Windows dostępny przez **winget** lub WSL.

```
wget https://example.com/plik.zip
```

ssh — loguje się zdalnie do innego komputera. `ssh user@adres`. Wbudowany na macOS i w Windows (OpenSSH Client od 2018).

scp — kopiuje pliki przez SSH. Szyfrowane, bezpieczne. `scp plik user@adres:sciezka`. Wymaga uruchomionego serwera SSH po stronie zdalnej.

ipconfig / `ifconfig` — wyświetla adresy IP. Na Windows: `ipconfig`. Na macOS: `ifconfig`. Odpowiednik w PowerShell: `Get-NetIPAddress`.

nslookup — odpytuje serwery DNS. `nslookup nazwa-domeny.pl` zwraca adres IP i serwer DNS. Działa identycznie na obu systemach.

netstat — wyświetla aktywne połączenia sieciowe i nasłuchujące porty. `netstat -an` na Windows. Na macOS działa to samo, ale z większą liczbą opcji.

Uprawnienia — bezpieczeństwo

chmod — zmienia uprawnienia do plików (odczyt, zapis, wykonanie). Działa tylko na macOS. Na Windows uprawnienia steruje się komendą `icacls`.

sudo — wykonuje polecenie jako administrator (root). Standard na macOS. Na Windows odpowiednikiem jest UAC — terminal otwierasz "jako administrator".

chown — zmienia właściciela pliku. Tylko macOS. `chown nowy-user:nowa-grupa plik`. Wymaga `sudo` do cudzych plików.

umask — ustawia domyślne uprawnienia dla nowo tworzonych plików. Działa na macOS. `umask 022` to standard — nowe pliki mają 644, katalogi 755.

id — wyświetla identyfikatory użytkownika (UID) i grup (GID), do których należy. Działa na macOS. W PowerShell: `whoami /groups`.

```
id
```

Pakiety — instalowanie oprogramowania

winget — menedżer pakietów Windows. `winget install nazwa` pobiera i instaluje program z repozytorium. `winget search nazwa` szuka pakietów.

brew — Homebrew. Menadżer pakietów dla macOS. `brew install nazwa` instaluje narzędzia z linii poleceń. `brew install --cask nazwa` instaluje aplikacje z GUI.

apt-get — menedżer pakietów Debiana. Działa na Linux i WSL. `sudo apt-get install nazwa` wymaga uprawnień administratora.

scoop — menedżer pakietów dla Windows bez UAC. Instaluje w katalogu użytkownika, nie potrzebuje praw administratora. `scoop install nazwa`.

npm — menedżer pakietów dla Node.js. Działa na wszystkich platformach. `npm install -g nazwa` instaluje narzędzie globalnie (w systemowym PATH).

40 zaawansowanych komend

Różnice między platformami i szczegóły, które decydują o tym, czy komenda zadziała, czy zaskoczysz błędem.

Podstawy

ls na macOS pochodzi z BSD, na Linux z GNU. Różnice widać przy flagach: `ls --color` działa na GNU, na BSD wymaga `ls -G`. PowerShell aliasuje `ls` do `Get-ChildItem` — zwraca obiekty, nie tekst. Flaga `-Force` to odpowiednik `ls -a`.

rmdir na macOS usuwa tylko pusty katalog. Jeśli spróbujesz usunąć niepusty, dostajesz `Directory not empty`. W CMD `rmdir /s` usuwa z podkatalogami. Odpowiednik na macOS: `rm -r`.

cp i **mv** na macOS mają flagi inne niż w CMD. `cp -R` kopiuje rekurencyjnie, `cp -p` zachowuje atrybuty, `cp -i` pyta przed nadpisaniem. W CMD `copy` ma prostsze opcje — do zaawansowanego kopiowania jest `robocopy`.

Ścieżki: PowerShell akceptuje zarówno `\` jak `/`. CMD tylko `\`. macOS tylko `/`. Nawyk używania `/` we wszystkich powłokach uchroni cię przed błędami przenośności.

Eksploracja

cat na macOS potrafi numerować linie (`cat -n plik`) i wyciszyć puste (`cat -s`). W PowerShell `Get-Content plik` zwraca obiekty z właściwościami — możesz filtrować po zawartości, długości linii, dacie.

less na macOS wspiera wyszukiwanie (`/fraza`, `?fraza` wstecz) i nawigację vi (`j` w dół, `k` w górę). Na Windows nie ma natywnie. Alternatywa w CMD: `more` — ale scrolluje tylko w dół i nie pozwala szukać.

find i **grep** na macOS pochodzą z BSD. `find` nie obsługuje `-printf` (GNU). Zamiast tego łączysz `-print` z potokiem do `awk` lub `sed`. **grep** na macOS wspiera `-E` (rozszerzone regex) i `-i` (wielkość liter). W PowerShell odpowiednik `Select-String` wspiera regex, ale składnia różni się od grep.

wc na macOS rozróżnia znaki (`-m`) i bajty (`-c`). Przy UTF-8 z polskimi znakami to dwie różne wartości. `echo "źdźbło" | wc -m` zwraca 7 (znaków). `wc -c` zwraca 9 (bajtów w UTF-8).

System

ps na macOS domyślnie pokazuje tylko procesy bieżącego użytkownika. `ps aux` pokazuje wszystkie z przydziałem CPU i pamięci. W PowerShell **Get-Process** zwraca obiekty — sortujesz po CPU (`Sort-Object CPU`) lub pamięci (`Sort-Object WorkingSet64`).

kill na macOS używa sygnałów Unix. `kill -15` (SIGTERM) — grzeczne zakończenie. `kill -9` (SIGKILL) — natychmiastowe, bez zapisywania danych. Na Windows **taskkill** bez `/F` odpowiada SIGTERM, z `/F` — SIGKILL.

top na macOS odświeża się co sekundę. Naciśnij `o` i wpisz `cpu` — sortuje procesy według użycia procesora. Naciśnij `q` — wychodzisz. W PowerShell odpowiednik jednorazowy: `Get-Process | Sort-Object CPU -Descending | Select -First 10`.

uname -a na macOS zwraca `Darwin Kernel Version 23.x.x`. To wciąż Unix — Apple nazywa jądro Darwin. Na Windows w PowerShell odpowiednik:

`[System.Environment]::OSVersion.VersionString` — zwraca wersję .NET, nie jądra Windows.

Sieć

curl na Windows jest wbudowany od build 17063 (2018). Na starszych systemach potrzebujesz osobnej instalacji. `curl URL` domyślnie wyświetla treść na standardowe wyjście. `curl -o plik URL` zapisuje do pliku. `curl -I URL` pobiera tylko nagłówki HTTP.

ping na Windows domyślnie wysyła 4 pakiety i kończy. Na macOS pinguje bez końca — musisz przerwać Ctrl+C. `ping -t` na Windows = nieskończona pętla jak na macOS. `ping -c 4` na macOS = limit 4 pakietów jak na Windows.

netstat na macOS pokazuje PID procesu, który otworzył port — przydatne do debugowania "która aplikacja blokuje port 3000". Na Windows `netstat -b` pokazuje nazwę programu, ale wymaga uprawnień administratora.

nslookup działa identycznie na obu systemach. `nslookup -type=mx domena.pl` — serwery poczty. `nslookup -type=ns domena.pl` — serwery DNS dla domeny.

Uprawnienia

chmod używa notacji ósemkowej (777, 755, 644) lub symbolicznej (u+x, g-w, o+r). `chmod 755 plik` = właściciel: rwx, grupa: r-x, inni: r-x. Na macOS brak flagi `--reference` (GNU). Na Windows odpowiednikiem jest `icacls plik /grant uzytkownik:R`.

sudo na macOS zapamiętuje hasło na 5 minut (domyślny `timestamp_timeout` w `/etc/sudoers`). Po tym czasie pyta ponownie. `sudo -k` unieważnia zapamiętane hasło natychmiast. Na Windows od Windows 11 istnieje `sudo` jako alias do podniesienia uprawnień — ale działa inaczej niż na Unix. Otwarcie terminala "jako administrator" to bezpieczniejsza droga.

Pakiety

winget na Windows: `winget search nazwa` — szuka. `winget install nazwa --exact` — instaluje konkretny pakiet. `winget list` — pokazuje zainstalowane. `winget upgrade --all` — aktualizuje wszystko. Źródła: Microsoft Store, GitHub releases i własne repozytoria.

brew na macOS wymaga Xcode Command Line Tools. Instalacja: `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`.

`brew update` — odświeża listę pakietów. `brew upgrade` — aktualizuje wszystkie. `brew install --cask nazwa` — aplikacje z GUI (Google Chrome, VS Code).

apt-get działa na Debian, Ubuntu i WSL. `apt-cache search nazwa` — szuka. `apt-get install nazwa` — instaluje. `apt` to nowsze, ładniejsze opakowanie z paskiem postępu i kolorami. `sudo apt update && sudo apt upgrade` — standardowa rutyna aktualizacji.

scoop dla Windows instaluje w `C:\Users\nazwa\scoop\` — nie wymaga praw administratora. `scoop bucket add extras` dodaje rozszerzone repozytorium. `scoop update` odświeża wszystko. Idealne do narzędzi deweloperskich: git, python, ffmpeg, nodejs.

npm nie jest menedżerem systemowym, ale przez niego instalujesz kluczowe narzędzia deweloperskie. `npm install -g nazwa` — globalnie, trafia do PATH. `npx nazwa` uruchamia bez instalacji. `npm ls -g --depth=0` — lista globalnie zainstalowanych pakietów.

Sieć

dig — zaawansowane zapytanie DNS. Zwraca więcej informacji niż `nslookup` — czas odpowiedzi, autorytatywny serwer, sekcje odpowiedzi. `dig google.com` zwraca rekord A (IPv4). `dig -t MX domena.pl` — serwery poczty. `dig +short google.com` — tylko adres IP, bez szczegółów.

```
dig google.com
```

whois — sprawdza właściciela domeny, datę rejestracji, serwery DNS. `whois domena.pl`. Na macOS wbudowany. Na Windows wymaga instalacji (przez `winget`). Wynik zależy od operatora domeny — różne formaty dla .pl, .com, .de.

```
whois domena.pl
```

Procesy i system

htop — podgląd procesów na żywo. Kolorowy, interaktywny — przewijasz strzałkami, sortujesz klawiszem F6, zabijasz F9. Na macOS wymaga instalacji (`brew install htop`). Na Linux dostępny przez menedżer pakietów. Na Windows nie działa natywnie — alternatywa: `Process Explorer`.

```
htop
```

df — wyświetla miejsce na dyskach. `df -h` pokazuje w czytelnym formacie (GB, MB). Na macOS pokazuje też typ systemu plików. W PowerShell: `Get-PSDrive` — lista wszystkich dysków (w tym sieciowych).

```
df -h
```

uptime — czas pracy systemu od ostatniego uruchomienia. Na macOS pokazuje też średnie obciążenie (load average) — trzy liczby: 1, 5 i 15 minut. W PowerShell 6+ odpowiednik: `Get-Uptime`. Na Windows Server przez WSL.

```
uptime
```

Sieć i dyski

du — sumuje rozmiar plików i katalogów. `du -sh *` pokazuje rozmiar każdego pliku i podkatalogu w bieżącym katalogu. `-s` = suma, `-h` = czytelny format (K, M, G). `du -sh * | sort -h` sortuje od najmniejszego do największego.

```
du -sh *
```

mount — wyświetla zamontowane systemy plików. Na macOS pokazuje dyski wewnętrzne, zewnętrzne i sieciowe. `mount` bez argumentów — lista. `diskutil list` na macOS pokazuje partycje. W PowerShell: `Get-Volume`.

```
mount
```

DevOps i skrypty

crontab — planuje zadania cykliczne. `crontab -e` otwiera edytor z listą zadań. Każda linia: minuta, godzina, dzień miesiąca, miesiąc, dzień tygodnia, komenda. `0 9 * * 1-5 /sciezka/skrypt.sh` — od poniedziałku do piątku o 9:00. Na Windows odpowiednik: `schtasks`.

```
crontab -e
```

wget — zaawansowane pobieranie plików. `wget -O wyjscie.zip https://example.com/plik` zapisuje pod własną nazwą. `wget -c https://example.com/duzy.zip` wznawia przerwane pobieranie. Na macOS wymaga instalacji (`brew install wget`).

```
wget -O output.zip https://example.com/file
```

tar — archiwizacja plików. `tar -czf archiwum.tar.gz folder` pakuje folder do skompresowanego archiwum. `tar -xzf archiwum.tar.gz` rozpakowuje. `-c` = create, `-x` =

extract, `-z` = gzip, `-f` = nazwa pliku. Działa identycznie na macOS i Linux. Na Windows przez WSL lub Git Bash.

```
tar -czf archiwum.tar.gz folder
```

diff — porównuje dwa pliki linia po linii. `diff plik1.txt plik2.txt` pokazuje różnice. Linie z < — z pierwszego pliku, z > — z drugiego. `diff -u plik1 plik2` — format unified, czytelniejszy. Na Windows przez Git Bash lub WSL.

```
diff plik1.txt plik2.txt
```

Dodatki

tee — zapisuje output do pliku i jednocześnie wyświetla na ekranie. `komenda | tee plik.txt`. `tee -a plik.txt` dopisuje do istniejącego pliku. Działa na macOS. W PowerShell odpowiednik: `Tee-Object`.

```
ls -la | tee listing.txt
```

xargs — wykonuje komendę dla każdej linii z wejścia. `find . -name "*.tmp" | xargs rm` — znajduje i usuwa pliki .tmp. `xargs -n 1` — po jednym argumencie na wywołanie. `xargs -P 4` — równolegle 4 procesy. Działa na macOS. Na Windows przez WSL.

```
echo "a b c" | xargs -n 1 echo
```

watch — powtarza komendę cyklicznie. `watch -n 5 komenda` — wykonuje co 5 sekund. `watch -d komenda` — podświetla różnice między kolejnymi wyjściami. Działa na macOS (wymaga `brew install watch`) i Linux. Na Windows brak natywnego odpowiednika.

```
watch -n 1 df -h
```

flock — blokada pliku dla skryptów. Zapobiega równoczesnemu uruchomieniu. `flock /tmp/blokada.lock komenda` — jeśli blokada istnieje, czeka. `flock -n /tmp/blokada.lock komenda` — jeśli blokada istnieje, kończy z błędem. Działa na macOS. Na Windows przez WSL.

```
flock -n /tmp/build.lock npm run build
```

Koniec książki

Pod skórą systemu