



# Sieci przewodowe

**Sieć przewodowa** to rodzaj **sieci komputerowej**, w której **urządzenia są połączone za pomocą fizycznych kabli** — najczęściej **miedzianych (Ethernet)** lub **światłowodowych (fiber optic)**.

 **Najprościej mówiąc:**

Sieć przewodowa = połączenie komputerów, drukarek, routerów itp. **za pomocą kabli**, a nie przez Wi-Fi.



## Główne cechy:

- **Przesył danych:** Dane są transmitowane przez przewodniki, co zapewnia stabilne i szybkie połączenie, zazwyczaj z mniejszą podatnością na zakłócenia niż w sieciach bezprzewodowych.
- **Przykłady zastosowań:** Sieci LAN w domach, biurach, centrach danych czy systemy monitoringu CCTV.



# Przykłady sieci przewodowych

**Ethernet (LAN)** — najczęściej spotykana sieć przewodowa w domach i firmach.

- używa kabli **RJ-45 (skrętka)**,
- prędkości np. **100 Mb/s, 1 Gb/s, 10 Gb/s**,
- łączy komputery, routery, switchy.

**Światłowód (Fiber Optic)** — sieć zbudowana z kabli światłowodowych.

- bardzo duża prędkość transmisji,
- stosowana w łączach między miastami, serwerowniach lub w nowoczesnych domach (FTTH – Fiber To The Home).

**Połączenia szeregowe / przemysłowe** — np. RS-232, RS-485 — używane w automatyce i systemach przemysłowych.



# Jak działa sieć przewodowa

1. Każde urządzenie ma **kartę sieciową (NIC)**.
2. Kable łączą urządzenia poprzez **switch, router** lub **hub**.
3. Dane przesyłane są w postaci **sygnałów elektrycznych lub optycznych** przez przewody.
4. Komunikacja odbywa się według określonych **protokołów sieciowych** (np. TCP/IP, Ethernet).



## Zalety sieci przewodowej

- ✓ **Szybka transmisja danych** – stabilniejsze połączenie niż Wi-Fi.
- ✓ **Niskie opóźnienia (ping)** – ważne np. w grach i serwerach.
- ✓ **Bezpieczeństwo** – trudniej podsłuchać połączenie fizyczne.
- ✓ **Odporność na zakłócenia radiowe** – nie zależy od zasięgu Wi-Fi.



## Wady sieci przewodowej

- ✗ **Trudniejszy montaż** – trzeba prowadzić kable w ścianach lub po podłodze.
- ✗ **Mniejsza mobilność** – urządzenia muszą być fizycznie podłączone.
- ✗ **Koszty instalacji** – więcej sprzętu (kable, gniazda, przełączniki).



# Sieć bezprzewodowa

**Sieć komputerowa**, w której urządzenia **łączą się ze sobą bez użycia kabli**, wykorzystując fale radiowe, podczerwień lub mikrofale do przesyłania danych.

✚ Najprościej mówiąc:

**Sieć bezprzewodowa = połączenie komputerów, telefonów, tabletów i innych urządzeń bez kabli**, np. przez **Wi-Fi** lub **Bluetooth**.



# Przykłady sieci bezprzewodowych

- **Wi-Fi (WLAN – Wireless Local Area Network)**
  - najpopularniejsza sieć domowa lub biurowa,
  - wykorzystuje fale radiowe (częstotliwości 2,4 GHz i 5 GHz),
  - umożliwia łączenie komputerów, smartfonów i innych urządzeń z routerem bez kabli.
- **Bluetooth**
  - służy do połączenia urządzeń na krótką odległość (np. telefon ↔ słuchawki).
- **Sieć komórkowa (LTE, 5G)**
  - zapewnia dostęp do Internetu przez operatorów sieci komórkowych,
  - obejmuje duże obszary (miasta, regiony).
- **Hotspot Wi-Fi**
  - punkt dostępu umożliwiający innym urządzeniom korzystanie z Internetu bezprzewodowo.





# Jak działa sieć bezprzewodowa

1. Urządzenie (np. laptop, telefon) ma **kartę sieciową Wi-Fi**.
2. Wysyła i odbiera dane przez **antenę**, która komunikuje się z **routerem (punktem dostępowym)**.
3. Router przekazuje dane dalej – np. do Internetu przez kabel (światłowód lub Ethernet).
4. Komunikacja odbywa się wg ustalonych **protokołów sieciowych** (np. 802.11).



## Zalety sieci bezprzewodowej

- Mobilność      Można się łączyć z dowolnego miejsca w zasięgu sygnału
- Brak kabli      Nie trzeba prowadzić przewodów przez ściany
- Łatwość instalacji      Wystarczy router i urządzenia z Wi-Fi
- Wiele urządzeń      Może łączyć laptopy, smartfony, drukarki itd.



## Wady sieci bezprzewodowej

- **Mniejsza stabilność** Sygnał może zanikać przez ściany lub zakłócenia
- **Bezpieczeństwo** Łatwiej podsłuchać lub włamać się niż w sieci przewodowej
- **Niższa prędkość** Wolniejsza transmisja niż przez kabel Ethernet
- **Zużycie energii** Urządzenia mobilne szybciej rozładują baterię



# Polityka bezpieczeństwa w sieci

**Polityka bezpieczeństwa sieci (Network Security Policy)** to zbiór zasad, procedur i **wytocznych**, które określają:

- jak chronić dane i zasoby sieciowe,
- kto i w jaki sposób może korzystać z sieci,
- jakie działania są dozwolone, a jakie zabronione,
- jak reagować w razie incydentów bezpieczeństwa.

Innymi słowy: to **plan działania**, który pomaga organizacji **utrzymać poufność, integralność i dostępność danych (zasada CIA: Confidentiality, Integrity, Availability)**.



# Główne cele polityki bezpieczeństwa

1. **Poufność (Confidentiality)** – ochrona danych przed nieautoryzowanym dostępem.  
👉 np. szyfrowanie, uwierzytelnianie użytkowników.
2. **Integralność (Integrity)** – zapobieganie nieautoryzowanym zmianom danych.  
👉 np. sumy kontrolne, kopie zapasowe.
3. **Dostępność (Availability)** – zapewnienie, że dane i usługi są dostępne, gdy są potrzebne.  
👉 np. ochrona przed atakami DDoS, redundancja serwerów.



# Rodzaje polityk bezpieczeństwa w sieci

## 1. **Polityka dostępu do sieci (Access Control Policy)**

Określa, kto może korzystać z sieci i w jakim zakresie.

- ♦ *Przykład:* pracownicy działu finansowego mają dostęp do serwera księgowego, ale nie do serwera HR.

## 2. **Polityka haseł (Password Policy)**

Ustala reguły dotyczące złożoności, długości i zmiany haseł.

- ♦ *Przykład:* hasło musi mieć min. 10 znaków, zawierać litery, cyfry i znaki specjalne, zmiana co 90 dni.

## 3. **Polityka korzystania z Internetu (Acceptable Use Policy)**

Określa, w jaki sposób można korzystać z Internetu w pracy.

- ♦ *Przykład:* zakaz wchodzenia na strony o treści niezgodnej z prawem lub niezwiązanej z pracą.



## Rodzaje polityk bezpieczeństwa w sieci c.d.

### 4. **Polityka zapory sieciowej (Firewall Policy)**

Definiuje, które połączenia są dozwolone, a które blokowane.

- ♦ *Przykład:* ruch przychodzący na port 22 (SSH) dozwolony tylko z sieci firmowej.

### 5. **Polityka aktualizacji i łatania systemów (Patch Management Policy)**

Określa zasady instalowania poprawek bezpieczeństwa.

- ♦ *Przykład:* wszystkie serwery muszą być aktualizowane co najmniej raz w miesiącu.

### 6. **Polityka tworzenia kopii zapasowych (Backup Policy)**

Wskazuje, jak i kiedy wykonywać backupy danych.

- ♦ *Przykład:* codzienna kopia danych na serwer zewnętrzny, przechowywana przez 30 dni.

### 7. **Polityka reagowania na incydenty (Incident Response Policy)**

Opisuje procedury w razie włamania, ataku lub awarii.

- ♦ *Przykład:* po wykryciu ataku DDoS administrator odłącza serwer i powiadamia zespół SOC.



# Praktyczne zastosowania

Pracownik loguje się do sieci VPN z domu

Zastosowana polityka: Polityka dostępu + polityka haseł (Efekt: Bezpieczny, uwierzytelniony dostęp)

Firma blokuje porty 21 (FTP) i 23 (Telnet)

Zastosowana polityka: Polityka zapory sieciowej (Efekt: Ochrona przed niebezpiecznymi protokołami)

Użytkownicy mają ograniczony dostęp do pendrive'ów

Zastosowana polityka: Polityka dostępu do urządzeń (Efekt: Zapobieganie wyciekom danych)

Codzienne kopie zapasowe serwerów

Zastosowana polityka: Polityka backupu (Efekt: Ochrona przed utratą danych)

Pracownicy nie mogą instalować oprogramowania

Zastosowana polityka: Polityka uprawnień użytkownika (Efekt: Redukcja ryzyka złośliwego oprogramowania)





## Polityka haseł

Polityka haseł określa, jak **trudne powinny być hasła** i jak często trzeba je **zmieniać**.

**Ma to chronić** konta użytkowników przed włamaniami.

Hasło **powinno mieć duże i małe litery, cyfry i znaki specjalne**.

**Nie powinno się używać tego samego hasła w kilku miejscach**.

W firmach często **wymaga się zmiany hasła co kilka miesięcy**.

To **pomaga zwiększyć bezpieczeństwo** danych i systemów.



# Polityka korzystania z Internetu

Ta polityka mówi, jak **pracownicy mogą używać Internetu w pracy.**

Na przykład **zabrania wchodzenia na strony niezwiązane z obowiązkami służbowymi.**

**Chroni to firmę przed wirusami** i stratą czasu w pracy.

Dzięki niej **administratorzy mogą łatwiej monitorować ruch sieciowy.**

W szkole taka **zasada też może obowiązywać** – np. nie wolno grać online podczas lekcji.

To **pomaga utrzymać porządek i bezpieczeństwo w sieci.**



## Polityka zapory sieciowej (Firewall)

Zapora sieciowa **to program lub urządzenie, które blokuje niebezpieczne połączenia z Internetu.**

Polityka określa, które **połączenia są dozwolone, a które zablokowane.**

Na przykład może **pozwalać tylko na strony firmowe, a blokować porty używane przez hakerów.**

Dzięki temu **wirusy i nieautoryzowani użytkownicy nie dostają się do sieci.**

Firewall działa jak **strażnik** pilnujący bramy do firmy.

To jedna z podstawowych metod ochrony sieci komputerowych.



## Polityka tworzenia kopii zapasowych (Backup)

Ta polityka mówi, **jak często trzeba robić kopie danych**, żeby ich nie stracić.

**Backupy** mogą być zapisywane na dysku zewnętrznym lub w **chmurze**.

W razie awarii, **ataku lub błędu można łatwo odzyskać dane**.

Firmy często robią kopie codziennie lub co tydzień.

W domu też warto robić kopie zdjęć czy **dokumentów**.

Dzięki temu nawet po awarii komputera dane nie przepadają.



## Polityka reagowania na incydenty

Ta polityka określa, co robić, gdy **dojdzie do ataku lub problemu z bezpieczeństwem**.

Na przykład: kto ma zostać powiadomiony, jak zabezpieczyć dane i jak naprawić system.

Pomaga **szybko zareagować, zanim szkody będą duże**.

W firmach często **tworzy się specjalne zespoły reagowania (tzw. CERT lub SOC)**.

Dzięki tej **polityce każdy wie, jakie są jego obowiązki**.

To pozwala **utrzymać spokój i porządek w sytuacjach kryzysowych**.



# Programowanie rozproszone

to sposób tworzenia oprogramowania, w którym **program (system)** nie działa na jednym komputerze, tylko na **wielu połączonych ze sobą komputerach (w sieci)**.

Każda z tych maszyn wykonuje część zadań, a razem tworzą jeden wspólny system.



## Rodzaje (modele) progr. rozproszonego c.d.

### Rodzaj:

**Klient-serwer (Client-Server)** - Klient wysyła żądania, serwer odpowiada.

Przykład: Strona WWW (przeglądarka ↔ serwer)

**Wielowarstwowe (n-warstwowe)** - System podzielony na warstwy: prezentacji, logiki i danych.

Przykład: Aplikacja webowa: frontend – backend – baza danych

**Peer-to-Peer (P2P)** - Każdy komputer (węzeł) może być klientem i serwerem jednocześnie.

Przykład: Torrent, komunikatory (np. Skype, BitTorrent)



# Rodzaje (modele) programow. rozproszonego

**Zdalne wywołania procedur (RPC)** - Program na jednym komputerze wywołuje funkcję na innym.

Przykład: Java RMI, gRPC

**Systemy oparte o komunikaty (Message-Oriented)** - Komputery komunikują się przez kolejki wiadomości.

Przykład: RabbitMQ, Apache Kafka

**Mikroserwisowe (Microservices)** - System składa się z wielu małych usług, które komunikują się przez sieć.

Przykład: Aplikacje w chmurze (np. Netflix, Amazon)

**Grid Computing / Cloud Computing** - Wiele serwerów współdzieli zasoby obliczeniowe lub pamięć.

Przykład: AWS, Azure, Google Cloud





# Technologie używane w progr. rozproszonym

## Obszar:

## Przykłady

Backend

Java (Spring Cloud), .NET, Node.js

Komunikacja

REST API, gRPC, WebSocket, message queues (RabbitMQ, Kafka)

Bazy danych

MongoDB, Cassandra, PostgreSQL Cluster

Architektura

mikroserwisy, chmura (AWS, Azure, GCP)



# Cechy systemów rozproszonych

## Cecha

## Znaczenie

Współbieżność

wiele komputerów działa jednocześnie

Komunikacja przez sieć

wymiana danych np. przez HTTP, TCP/IP

Niezależność sprzętowa

różne komputery i systemy operacyjne mogą współpracować

Odporność na awarie

gdy jeden element padnie, reszta może działać dalej

Skalowalność

łatwo dodać więcej maszyn, żeby system działał szybciej



## Przykład z życia

### Wyobraź sobie aplikację bankową:

- Serwer A obsługuje logowanie,
- Serwer B zapisuje dane klientów w bazie,
- Serwer C przetwarza płatności.

Współpracują razem w sieci — **to właśnie programowanie rozproszone.**



# GIT

Git to **system kontroli wersji** (ang. *Version Control System*), używany do **śledzenia zmian w plikach**, głównie w projektach programistycznych. Dzięki niemu możesz współpracować z innymi, wracać do wcześniejszych wersji plików i zarządzać historią projektu.



# Najważniejsze cechy Gita

## Śledzenie zmian

- Git zapisuje historię zmian plików.
- Możesz zobaczyć, kto i kiedy zmienił dany fragment kodu.

## Gałęzie (*branches*)

- Pozwalają na tworzenie równoległych wersji projektu.
- Możesz testować nowe funkcje bez psucia głównej wersji (*main* lub *master*).

## Scalanie zmian (*merge*)

- Gałęzie można łączyć.
- Git automatycznie łączy zmiany w jednym projekcie.



# Najważniejsze cechy Gita

## Rozproszony system

- Każdy użytkownik ma pełną kopię repozytorium.
- Możesz pracować lokalnie bez połączenia z internetem.

## Współpraca

- Git ułatwia pracę zespołową.
- Popularne platformy: **GitHub**, **GitLab**, **Bitbucket**.



# Podstawowe komendy Git

**git init**

- Tworzy nowe repozytorium w folderze

**git clone <url>**

- Pobiera repozytorium z internetu

**git add <plik>**

- Dodaje plik do "staging area" (przygotowanie do commit)

**git commit -m "komentarz"**

- Zapisuje zmiany z opisem

**git status**

- Pokazuje stan repozytorium i zmienione pliki



## Podstawowe komendy Git

- git push** - Wysyła zmiany do zdalnego repozytorium
- git pull** - Pobiera zmiany z repozytorium zdalnego
- git branch** - Wyświetla dostępne gałęzie
- git checkout <branch>** - Przełącza się na inną gałąź





# git commit

**git commit** - służy do zapisania zmian, które zostały wcześniej dodane do obszaru **index** (za pomocą **git add**)

Pomocne polecenia do wykonania commit

- **a** - przejście w tryb –INSERT , modyfikowania
- **ESCAPE :wq** - ESC (przełącza w tryb poleceń), w -zapisz, q-quit, wyjście

## Ważne opcje:

**-m "tytuł commit"**: Dodaje komunikat bezpośrednio w poleceniu.

**--amend**: Pozwala edytować ostatni commit (np. poprawić wiadomość lub dodać zapomniane pliki).

**-a**: Automatycznie dodaje wszystkie śledzone pliki do staging przed commitem (skrót od git add dla zmodyfikowanych plików).



# git log

**git log** - służy do wyświetlania historii commitów w repozytorium. W kolejności chronologicznej – od najnowszego do najstarszego.

**Wyświetlane informacje:** Dla każdego commita pokazuje:

- ❑ Hash commita (unikalny identyfikator).
- ❑ Autora i datę.
- ❑ Wiadomość commita.

## Ważne opcje:

**--oneline:** Wyświetla każdy commit w jednej linii (skrótowo: hash + wiadomość).

**--graph:** Pokazuje historię w formie grafu (drzewa branchy).

**-n 5:** Ogranicza do ostatnich n commitów (np. 5).

**--author="Imię":** Filtruje po autorze.

**--since="2023-01-01":** Commit'y od danej daty.

**-p lub --patch:** Pokazuje pełne zmiany (diff) w każdym commicie.



## index i git checkout

**Index w Git** (nazywany też **staging area** albo **strefą przejściową**) to **miejsce, w którym Git przechowuje informacje o zmianach przygotowanych do zatwierdzenia (commit).**

### Przykład 1:

Tworzymy katalog w nim są plik **\$ mkdir folder-test** i **\$ touch folder-test/plik2.txt**.

**Dodajemy je do index** za pomocą polecenie **\$ git add folder-test**. Następnie usuwamy jeden z plików w katalogu folder-test **\$ rm folder-test/plik2.txt**. Aby go odzyskać mimo tego, że nie ma go w katalogu roboczym należy wykonać polecenie **\$ git checkout folder-test/plik2.txt**

### Przykład 2:

Teraz jak dodamy tekst do pliku **folder-test/plik2.txt** to plik będzie wyświetlany 2 razy w index (gdzie będzie pusty) i w katalogu roboczym z nową zawartością tekstu.



## index i git reset HEAD , touch .gitignore

**git reset HEAD <plik>** usuwa wskazany plik z **indexu**, ale **nie usuwa go z katalogu roboczego**.

Plik **.gitignore** służy do tego, aby powiedzieć Gitowi, **których plików i katalogów NIE ma śledzić** (czyli nie dodawać ich do indexu, commitów i repozytorium).

Git ignoruje pliki wymienione w **.gitignore**, o ile nie były wcześniej dodane do repozytorium.  
Zwykle ignorujemy katalogi node\_module, dist, .env oraz pliki \*.log, \*.temp, DS\_Store

Plik **.gitignore** jest niewidoczny dopiero polecenie **\$ ls -all** go wyświetli



## git revert SHA-1

**git revert SHA-1** tworzy **NOWY commit**, który wykonuje **odwrotność zmian z tamtego commita**.

### Przykład 1:

W commit **git show SHA-1** dodaliśmy w 2 linii tekst, a usunęliśmy pierwszą linię tekstu, to możemy to odwrócić. Czyli po wykonaniu polecenia **git revert SHA-1** odwrócimy to co było w pliku. **Druga linia zostanie usunięta, a pierwsza linia zostanie dodana z zawartością tekstu przed usunięciem jej**. Pamiętajmy, że ta **zmiana dodaje nowy commit**



## Działanie git reset z historią commit

**git reset** - resetuje index do danego commit lub zresetowanie indexu do wybranych plików danego commit. Czyli **git reset HEAD nazwa.txt**, pobierze z ostatniego commit plik nazwa.txt przywróci do index. Natomiast **to co jest obecnie w index** umieści w katalogu roboczym

**Sprawdzenie** wykonuje się poleceniem **git restore nazwa-pliku**



## git rebase -interactive HEAD~

**git rebase** - służy do przeniesienia commit z jednego miejsca do drugiego. W szczególności do poprawy commit, łączenia ich, układania w odpowiedniej kolejności, usunięcia zbędnych commit

Commands:

```
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
```



## git branch -verbose, git branch nowy-brache

**git branch -v** - wyświetla listę wszystkich gałęzi razem z ostatnim commitem na każdej z nich (skrót commit hash + pierwsza linia wiadomości, gdzie \* oznacza branche na którym jesteśmy)

**git branch -list** - zwraca listę branch (tylko nazwę)

**git branch -m nowy-branch nowa-nazwa-branch** - zmienia nazwę branch

**git branch -d nazwa-branch** - usuwa branch





## git checkout nazwa-branch

**git checkout nazwa-branch** - przełączenie się na brancha (gałąź) o podanej nazwie

**git checkout -b nazwa-branch** - tworzy nowy branch i od razu przełącza się na niego



# git stash

**git stash** - **tyczasowo odkłada Twoje zmiany na bok**, tak żeby repo wyglądało, jakby nie było żadnych zmian. Zmiany muszą znajdować się w index aby można było je odłożyć na bok za pomocą polecenia git stash

**git stash list** - **wyświetla listę stash**

**git stash pop** - **zdejmuje z listy stash ostatnią zmianę oraz wyświetla szczegółowe informacje o statusie**



## git merge nazwa\_branch

**git merge nazwa\_branch** (lub nazwe commit, SHA-1) - łączy (scala) historię wskazanej gałęzi z aktualną gałęzią na której jesteś.

**Mogą się pojawić konflikty. Czyli jeśli te same linie w tych samych plikach zostały zmienione inaczej w obu branchach – powstaje konflikt.**

Rozwiązanie konfliktów wykonuje się przez otworenie pliku w którym są konflikty. Usunięciu linii <<<<<<, =====, >>>>>> oraz **wybraniem poprawnej wersji**. Po czym należy dodać pliki **\$ git add nazwa\_pliku\_z\_konfliktem** i **wykonać commit**



# git clone

**git clone** - **kopiuje całe zdalne repozytorium** (np. z GitHuba) na Twój komputer.

Oznacza to:

- ❑ **pobiera wszystkie pliki**
- ❑ **pobiera całą historię commitów**
- ❑ **ustawia domyślne połączenie z repozytorium (origin)**
- ❑ **tworzy katalog z projektem**

Przykład:

```
git clone https://github.com/user/projekt.git
```



# git push

**git push** - wysyła Twoje **lokalne commity** do **zdalnego repozytorium** (np. GitHub, GitLab, Bitbucket).

**git push origin nazwa\_brancha**

gdzie:

**origin** – nazwa zdalnego repozytorium

**nazwa\_brancha** – gałąź, którą chcesz wysłać (np. main, feature/login)

**Nie wysyła** stash, plików roboczych, nic co jest nie commit



# git pull

**git pull** - pobierania i integracji zmian z zdalnego repozytorium do lokalnej kopii projektu. Łączy ono operacje **git fetch** (pobieranie zmian) i **git merge** (scalanie ich z bieżącą gałęzią), co pozwala na aktualizację lokalnego kodu o najnowsze wersje od współpracowników.

## **git pull wykonuje wewnętrznie:**

- ❑ **Fetch:** Git pobiera wszystkie nowe commity i branchy z zdalnego repozytorium, ale nie zmienia lokalnych plików.
- ❑ **Merge:** Scalane są zmiany z pobranej gałęzi do Twojej lokalnej. Jeśli są konflikty (np. edytowałeś te same linie), Git poprosi o ich ręczne rozwiązanie.



## git fetch

**git fetch** - służy do **pobierania zmian z zdalnego repozytorium** (np. z GitHuba) **do lokalnego repozytorium, bez automatycznego scalania ich z Twoimi lokalnymi gałęziami**. Aktualizuje ono informacje o zdalnych branchach (np. "origin/main"), co pozwala sprawdzić, co się zmieniło, zanim zdecydujesz się na integrację (np. via merge lub rebase)



## git fetch vs git pull

### Co robi?

**git fetch** pobiera zmiany z serwera

**git pull** pobiera i łączy zmiany

### Czy zmienia pliki lokalne?

✗ NIE

✓ TAK

### Czy łączy zmiany?

✗ NIE

✓ TAK





# Szyfrowanie sieci internetowych HTTPS

**HTTPS (Hypertext Transfer Protocol Secure)** to bezpieczna wersja protokołu HTTP używana w internecie do przesyłania danych między przeglądarką a serwerem. Dzięki HTTPS komunikacja jest

**szyfrowana,**

**uwierzytelniana**

**integralna,**

co zapewnia ochronę danych użytkownika.



# Co daje HTTPS?

## Szyfrowanie (encryption)

Dane przesyłane między przeglądarką a serwerem są zaszyfrowane przy pomocy protokołu **TLS (Transport Layer Security)**. Nawet jeśli ktoś przechwyci ruch sieciowy – zobaczy tylko zaszyfrowane, bezużyteczne dane.

## Uwierzytelnienie (authentication)

**Certyfikat SSL/TLS** potwierdza, że strona, z którą się łączysz, jest prawdziwa i należy do właściwej firmy lub osoby

## Integralność danych

**Zapewnia, że dane nie zostały zmienione po drodze** – nikt nie może ich potajemnie edytować, podmienić ani wstrzyknąć złośliwy kod.



# Certyfikaty internetowe

**Certyfikaty internetowe** (zwane także **certyfikatami cyfrowymi, SSL/TLS certyfikatami**) to specjalne elektroniczne dokumenty, które służą do **potwierdzania tożsamości stron w internecie** oraz **umożliwiają bezpieczne, szyfrowane połączenia** między użytkownikiem a serwerem.

Jest oparty na standardzie X.509 i zawiera zestaw kluczowych informacji, które służą do weryfikacji tożsamości serwera oraz umożliwienia szyfrowania danych.



# certyfikaty internetowe

## 1 Potwierdzają tożsamość strony

Certyfikat zapewnia, że strona, którą odwiedzasz, jest rzeczywiście tym, za co się podaje.

Przykład:

Jeśli wpisujesz <https://bank.pl>, certyfikat gwarantuje, że łączysz się z prawdziwym bankiem, a nie z fałszywą stroną.

## 2 Umożliwiają szyfrowanie danych

Certyfikat zawiera **klucz publiczny**, używany do bezpiecznego przesyłania danych.

Dzięki temu:

- hasła
- numery kart płatniczych
- dane osobowe

są zaszyfrowane i niemożliwe do odczytania przez osoby trzecie.

## 3 Są wystawiane przez zaufane instytucje

Nazywa się je **CA – Certificate Authority**, np.:

- Let's Encrypt
- DigiCert
- GlobalSign
- Sectigo

Przeglądarki ufają tym instytucjom.



## Certyfikat internetowy zawiera m.in.:

- **Wersja (Version)**: Określa wersję standardu certyfikatu (np. wersja 3 dla X.509 v3).
- **Numer seryjny (Serial Number)**: Unikalny identyfikator nadany przez urząd certyfikacji (CA).
- **Algorytm podpisu (Signature Algorithm)**: Określa algorytm użyty do podpisania certyfikatu (np. SHA-256 with RSA).
- **Wydawca (Issuer)**: Dane urzędu certyfikacji, który wystawił certyfikat, w tym nazwa, kraj, organizacja itp.
- **Okres ważności (Validity)**: Daty "od" (Not Before) i "do" (Not After), definiujące, kiedy certyfikat jest aktywny.



## Certyfikat internetowy zawiera m.in.:

- **Podmiot (Subject)**: Dane właściciela certyfikatu, np. nazwa domeny (Common Name), organizacja, kraj, miasto.
- **Informacje o kluczu publicznym podmiotu (Subject Public Key Info)**: Klucz publiczny serwera oraz algorytm klucza (np. RSA lub ECDSA).
- **Rozszerzenia (Extensions)**: Dodatkowe pola, takie jak Subject Alternative Names (SAN) dla alternatywnych domen, użycie klucza (Key Usage), autorytet certyfikacji itp.
- **Podpis cyfrowy (Signature)**: Podpis urzędu CA, weryfikujący autentyczność certyfikatu.



# Rodzaje certyfikatów

## → DV – Domain Validation

Najprostsze i najtańsze (np. Let's Encrypt).  
Sprawdzana jest tylko domena.

## → OV – Organization Validation

Weryfikowana jest firma, która posiada domenę.

## → EV – Extended Validation

Najwyższy poziom weryfikacji – pełna weryfikacja organizacji.  
Kiedyś prezentowane jako zielony pasek w przeglądarce.



# SSL czy TLS

**SSL (Secure Sockets Layer):** To starszy protokół szyfrowania, opracowany w latach 90. przez Netscape. Jego ostatnie wersje (SSL 2.0 i 3.0) miały poważne luki bezpieczeństwa, dlatego od dawna nie są zalecane i nie są używane w praktyce.

**TLS (Transport Layer Security):** To ulepszona i bezpieczniejsza wersja SSL, rozwijana od 1999 roku. Aktualne wersje to TLS 1.2 i TLS 1.3 (TLS 1.0 i 1.1 też są przestarzałe i wycofywane). TLS jest standardem de facto w dzisiejszym internecie.