

Podstawy pracy z Git

AM

29 września 2024



1 Wstępna konfiguracja Git

Po instalacji systemu Git należy go skonfigurować, aby można było ustalić naszą tożsamość podczas wprowadzania zmian w repozytorium. Aby ustawić naszą nazwę użytkownika oraz e-mail podczas pracy z Git'em wykonujemy następujące dwa polecenia:

```
git config --global user.name [nazwa użytkownika]
git config --global user.email [adres e-mail]
```

Przykładowo:

```
git config --global user.name "Adam Nowak"
git config --global user.email "email@wp.pl"
```

Gdy podczas pracy Git wymaga wprowadzenia jakiejś informacji otwiera systemowy edytor tekstowy. W przypadku systemu Windows, edytor tekstowy wybierany jest podczas instalacji Git'a. Jeżeli chcemy zmienić edytor, lub instalujemy Git na innym systemie używamy komendy:

```
git config --global core.editor [nazwa edytora]
```

Przykładowo - polecenie, które ustawi Visual Studio Code jako nasz domyślny edytor przy pracy z Git będzie wyglądać następująco:

```
git config --global core.editor "code --wait"
```

Jeżeli chcemy ustawić Notepad++ jako nasz domyślny edytor przy pracy z Git, musimy podać ścieżkę do jego pliku wykonywalnego. Ta komenda może wyglądać następująco (zależnie od folderu, w którym zainstalowany jest Notepad++):

```
git config --global core.editor "C:/Program Files (x86)/Notepad++/notepad++.exe"
```

Aby wyświetlić listę wszystkich ustawień wprowadzamy polecenie:

```
git config --list
```

```

Admin@DESKTOP-030061C MINGW64 ~
$ git config --global user.name "Adam Nowak"

Admin@DESKTOP-030061C MINGW64 ~
$ git config --global user.email "email@wp.pl"

Admin@DESKTOP-030061C MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor=notepad
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Adam Nowak
user.email=email@wp.pl

```

Rysunek 1: Wygląd przykładowej konfiguracji w konsoli, gdzie notatnik jest edytorem tekstowym Git'a.

2 Inicjalizacja repozytorium

2.1 Nowe repozytorium z istniejącego katalogu

Aby utworzyć nowe repozytorium w wybranym katalogu najpierw musimy do niego przejść. Możemy zrobić to poprzez otwarcie szukanego katalogu w środowisku graficznym, kliknięcie prawego przycisku myszy i wybranie "Git Bash Here" w menu kontekstowym, lub nawigując w systemie plików przy pomocy komend w terminalu. Po przejściu do wybranego katalogu wykonujemy polecenie:

```
git init
```

Poskutkuje to pojawieniem się podkatalogu ".git", który zawiera wszystkie niezbędne metadane repozytorium.

```

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona
$ git init
Initialized empty Git repository in C:/Users/Admin/Documents/strona/.git/

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls
galeria.htm index.htm obsada.htm style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls -a
./ ../ .git/ galeria.htm index.htm obsada.htm style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ |

```

Rysunek 2: Przykład inicjalizacji repozytorium w katalogu "strona", wgląd w jego zawartość za pomocą komendy "ls", oraz wyświetlenie ukrytych plików, w tym podkatalogu ".git".

Następnie dodać należy pliki, których zmiany chcemy śledzić. Dokonujemy tego za pomocą komendy:

```
git add [plik]
```

Możemy dodać konkretny plik podając jego nazwę jako argument polecenia. Np.:

```
git add LICENSE
```

Możemy również śledzić zmiany we wszystkich plikach z konkretnym rozszerzeniem, np.:

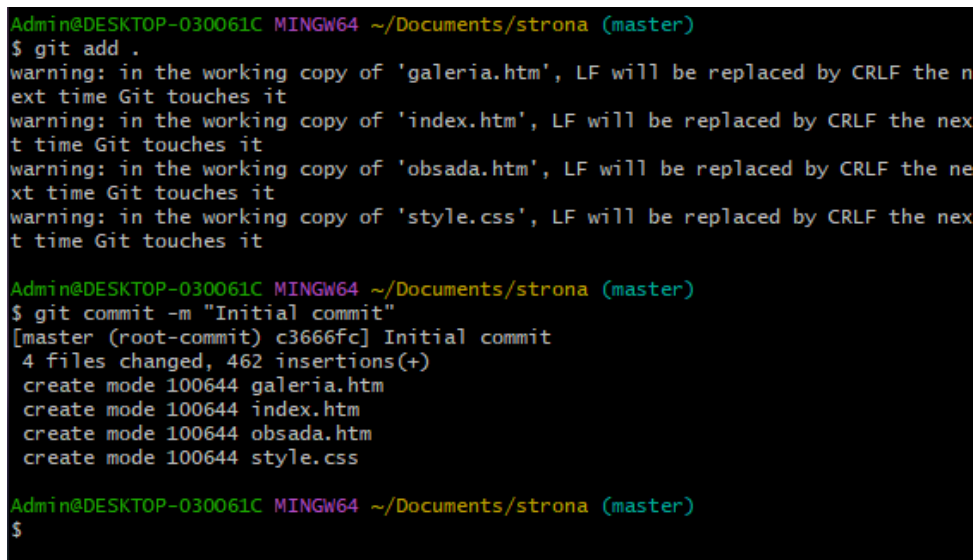
```
git add *.cpp
```

Jeżeli dodać chcemy wszystkie pliki znajdujące się aktualnie w katalogu musimy użyć polecenia:

```
git add .
```

Na koniec zatwierdzamy zmiany przy pomocy komendy:

```
git commit -m "Initial commit"
```



```
Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git add .
warning: in the working copy of 'galeria.htm', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'index.htm', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'obsada.htm', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'style.css', LF will be replaced by CRLF the next time Git touches it

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git commit -m "Initial commit"
[master (root-commit) c3666fc] Initial commit
4 files changed, 462 insertions(+)
create mode 100644 galeria.htm
create mode 100644 index.htm
create mode 100644 obsada.htm
create mode 100644 style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$
```

Rysunek 3: Wygląd inicjacji nowego repozytorium w folderze "strona".

2.2 Klonowanie repozytorium z serwera

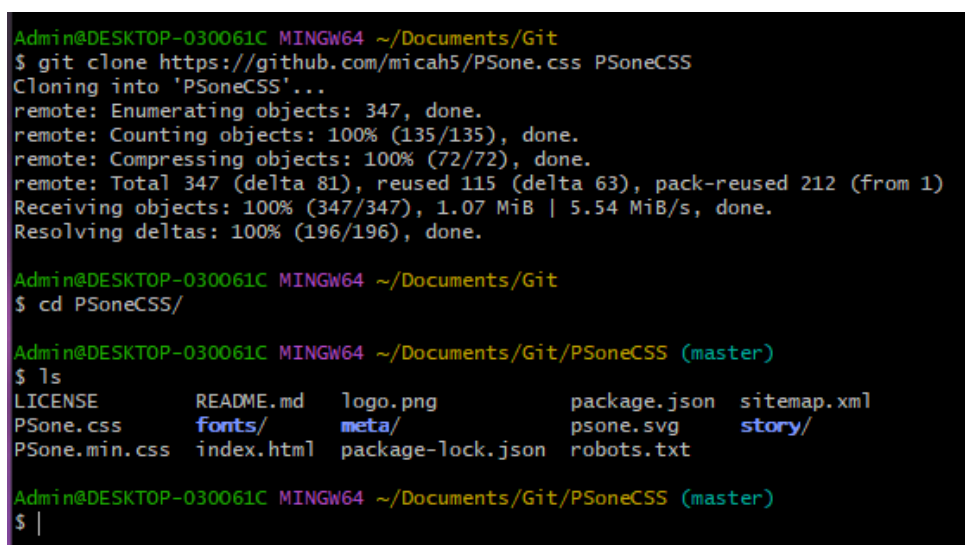
Jeżeli chcemy pracować lokalnie na repozytorium znajdującym się na innym serwerze musimy sklonować dane repozytorium wykonując polecenie:

```
git clone [URL] [katalog]
```

Gdzie [URL] zastępujemy linkiem do repozytorium, a [katalog] nazwą katalogu z zawartością repozytorium, który zostanie utworzony lokalnie na naszym dysku. Przykładowo:

```
git clone https://github.com/username/repo-name.git repozytorium
```

...sklonuje zawartość repozytorium z podanego adresu do katalogu "./repozytorium".



```
Admin@DESKTOP-030061C MINGW64 ~/Documents/Git
$ git clone https://github.com/micah5/PSone.css PSoneCSS
Cloning into 'PSoneCSS'...
remote: Enumerating objects: 347, done.
remote: Counting objects: 100% (135/135), done.
remote: Compressing objects: 100% (72/72), done.
remote: Total 347 (delta 81), reused 115 (delta 63), pack-reused 212 (from 1)
Receiving objects: 100% (347/347), 1.07 MiB | 5.54 MiB/s, done.
Resolving deltas: 100% (196/196), done.

Admin@DESKTOP-030061C MINGW64 ~/Documents/Git
$ cd PSoneCSS/

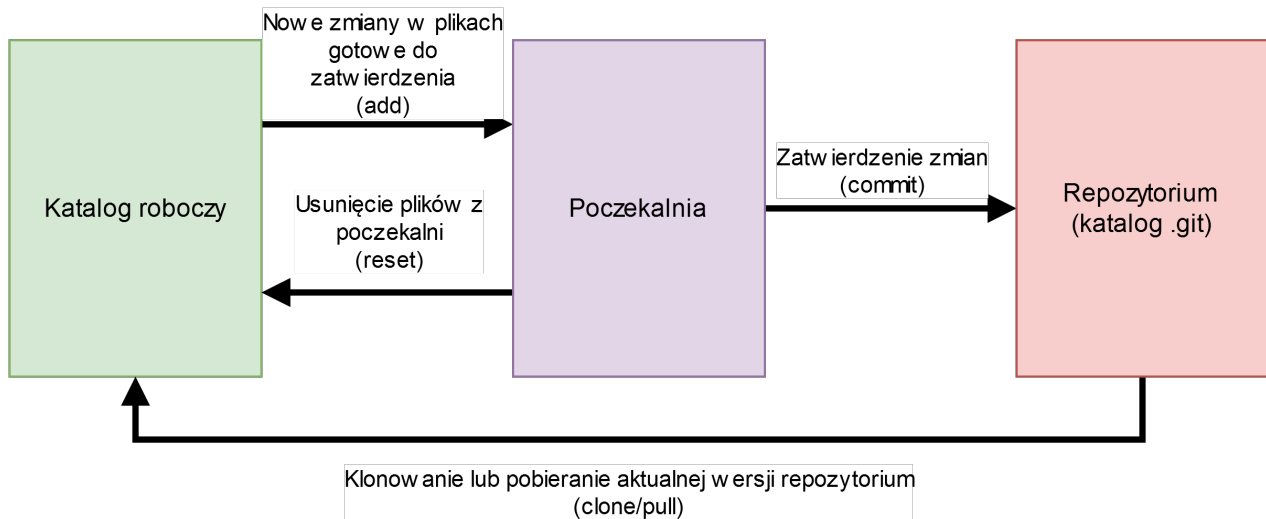
Admin@DESKTOP-030061C MINGW64 ~/Documents/Git/PSoneCSS (master)
$ ls
LICENSE      README.md    logo.png     package.json  sitemap.xml
PSone.css    fonts/       meta/        psone.svg     story/
PSone.min.css index.html   package-lock.json robots.txt
```

Rysunek 4: Przykład klonowania repozytorium "PSone.css" do katalogu "PSoneCSS" oraz wgląd w jego zawartość za pomocą komendy "ls".

3 Praca na plikach

Podczas pracy w systemie Git pliki mogą być w 3 stanach:

1. Zmodyfikowany - Plik w katalogu roboczym został zmodyfikowany,
2. W poczekalni - Zmodyfikowany plik znajduje się w poczekalni i oczekuje na zatwierdzenie,
3. Zatwierdzony - Modyfikacje zostały zatwierdzone i nowa wersja pliku została zapisana w repozytorium.



Rysunek 5: Uproszczona wizualizacja pracy w Git, opracowanie własne: [?]

3.1 Działania na plikach

Aby sprawdzić w jakim stanie znajdują się pliki w naszym katalogu roboczym używamy komendy:

```
git status
```

Dodawanie zmodyfikowanych plików, lub nowych plików, których zmiany chcemy zacząć śledzić do poczekalni odbywa się przy pomocy polecenia:

```
git add [plik]
```

```

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls
galeria.htm index.htm obsada.htm style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git add *.htm
warning: in the working copy of 'galeria.htm', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'index.htm', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'obsada.htm', LF will be replaced by CRLF the next time Git touches it

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   galeria.htm
        new file:   index.htm
        new file:   obsada.htm

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        style.css

```

Rysunek 6: Zawartość konsoli po dodaniu do poczekalni wszystkich plików z katalogu "strona" z rozszerzeniem "htm", oraz wgląd w stan plików przy pomocy polecenia "git status".

Jeżeli jednak w poczekalni znajdzie się plik, którego zmian nie chcemy śledzić (np. dodany został plik tekstowy z danymi testowymi po dodaniu "hurtowo" wszystkich plików w katalogu), możemy go z niej usunąć przy pomocy komendy:

```
git reset [plik]
```

```

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls
api.key.txt galeria.htm index.htm obsada.htm style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git add .

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git status
On branch master

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   api.key.txt

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git reset api.key.txt

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git status
On branch master

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        api.key.txt

nothing added to commit but untracked files present (use "git add" to track)

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$

```

Rysunek 7: Przykład usunięcia pliku "api.key.txt" z poczekalni, oraz sprawdzenie statusu plików.

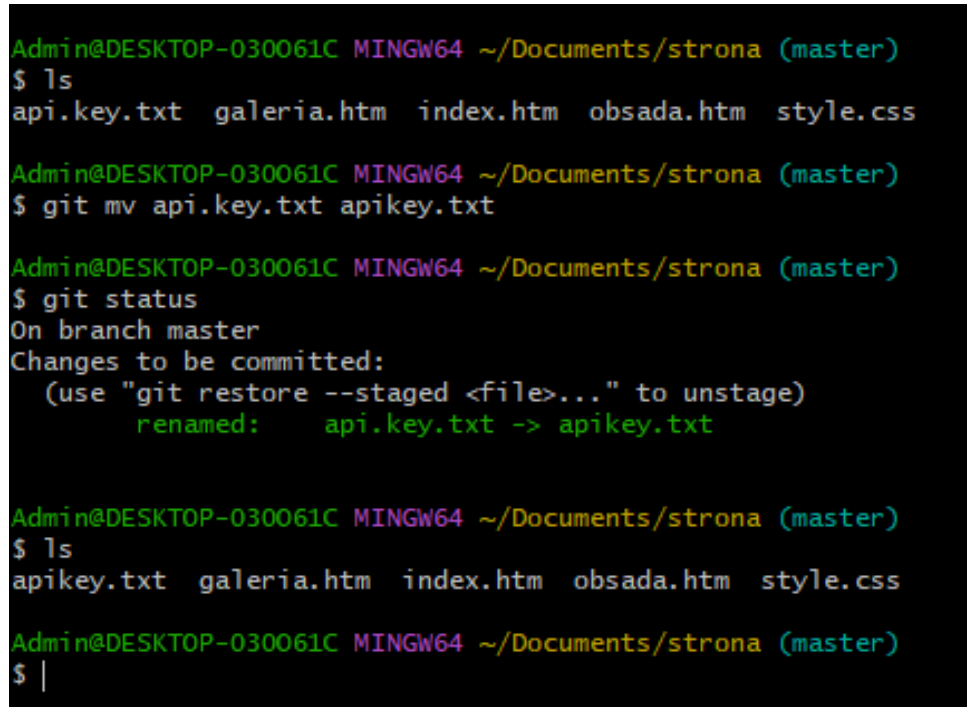
Jeżeli chcemy kompletnie usunąć plik zarówno z repozytorium jak i naszego katalogu roboczego używamy komendy:

```
git rm [plik]
```

Jeżeli zmienimy nazwę pliku w naszym katalogu roboczym, Git zarejestruje to jako usunięcie starego pliku znajdującego się w repozytorium i dodanie kompletnie nowego. Możemy więc zamiast tego użyć polecenia:

```
git mv [plik ze starą nazwą] [plik z nową nazwą]
```

..które zmieni nazwę pliku w katalogu roboczym i poprawnie zarejestruje zmianę w repozytorium.



```
Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls
api.key.txt  galeria.htm  index.htm  obsada.htm  style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git mv api.key.txt apikey.txt

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:   api.key.txt -> apikey.txt

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls
apikey.txt  galeria.htm  index.htm  obsada.htm  style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ |
```

Rysunek 8: Przykład zmiany nazwy pliku "api.key.txt" na "apikey.txt" przy pomocy komendy "git mv" z poprawnie zarejestrowaną zmianą w repozytorium i w katalogu roboczym.

3.2 Zatwierdzanie zmian

Kiedy chcemy zatwierdzić wszystkie zmiany w plikach w poczekalni i zapisać je w repozytorium używamy polecenia:

```
git commit
```

...co spowoduje otwarcie edytora tekstowego określonego w konfiguracji Git z prośbą o wpisanie tam wiadomości do migawki. Wiadomość ta powinna zawierać informacje o zmianach, które wprowadziliśmy w plikach. Aby zatwierdzić zmiany i podać wiadomość do migawki w tym samym poleceniu dodajemy parametr "m" i wpisujemy wiadomość w następujący sposób:

```
git commit -m [wiadomość]
```

Jeżeli chcemy cofnąć zmiany wprowadzone w ostatnim commicie możemy użyć komendy:

```
git commit --amend
```

..która pozwoli nam pracować na plikach z poprzedniej migawki repozytorium. Kiedy chcemy zobaczyć historię zmian w repozytorium możemy użyć komendy:

```
git log
```

..która wyświetli wszystkie zarejestrowane migawki, wraz z ich dołączonymi wiadomościami, autorami i datami.

```

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git rm obsada.htm
rm 'obsada.htm'

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ ls
apikey.txt  galeria.htm  index.htm  style.css

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:   api.key.txt -> apikey.txt
        deleted:   obsada.htm

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git commit -m "Usunięcie podstrony z obsadą (obsada.htm) i poprawa nazwy pliku
z kluczem do API (apikey.txt)"
[master 6d476dc] Usunięcie podstrony z obsadą (obsada.htm) i poprawa nazwy pliku
z kluczem do API (apikey.txt)
2 files changed, 69 deletions(-)
rename api.key.txt => apikey.txt (100%)
delete mode 100644 obsada.htm

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ git log
commit 6d476dc2e4d0256aa5820fedcd9a4803fd040a3c (HEAD -> master)
Author: Adam Nowak <email@wp.pl>
Date: Mon Sep 9 00:26:51 2024 +0200

    Usunięcie podstrony z obsadą (obsada.htm) i poprawa nazwy pliku z kluczem do
    API (apikey.txt)

commit c3666fc5974094160942995c08e0fa85585b7e52
Author: Adam Nowak <email@wp.pl>
Date: Thu Sep 5 06:12:15 2024 +0200

    Initial commit

Admin@DESKTOP-030061C MINGW64 ~/Documents/strona (master)
$ |

```

Rysunek 9: Przykład usunięcia pliku za pomocą "git rm", sprawdzenia stanu plików, zatwierdzenia zmian z dodaną wiadomością oraz sprawdzenia historii migawek.

4 Pomijanie śledzenia zmian konkretnych plików

Chcąc uniknąć śledzenia zmian w konkretnych plikach, lub plikach o konkretnych rozszerzeniach musimy zajrzeć do pliku ".gitignore". Jest to plik, który zawiera listę plików, których Git nie powinien pod żadnym pozorem śledzić i zapisywać w repozytorium. Jest to szczególnie przydatne w przypadku posiadania plików ekskluzywnych do naszego lokalnego środowiska (np. plików tymczasowych), lub tajnych informacji (np. kluczy API).

Jeżeli plik .gitignore jeszcze nie istnieje w katalogu roboczym to powinniśmy go utworzyć. Następnie zapisujemy w nim reguły unikania plików linia po linii. Przykładowo - jeżeli nie chcemy zapisać pliku API.txt do repozytorium, zapisujemy następującą linię w pliku .gitignore:

```
API.txt
```

Możemy również dodać komentarz do reguły, aby łatwo ją później odnaleźć na liście. Komentarze poprzedzamy znakiem #, w następujący sposób:

```
# Klucz API projektu
API.txt
```

Istnieje możliwość wykluczenia ze śledzenia wszystkich plików o konkretnych rozszerzeniach, poprzedzając rozszerzenie znakiem *. Przykładowo - jeżeli chcemy ignorować pliki o rozszerzeniu .log, zapisujemy następującą regułę w .gitignore:

```
*.log
```

Możemy także wykluczyć wszystkie pliki w konkretnym podkatalogu zapisując jego nazwę oraz znak /. Przykładowo - chcąc uniknąć śledzenia wszystkich plików w podkatalogu "test", zapisujemy:

```
test/
```

Jeżeli chcemy jednak śledzić tylko jeden konkretny plik znajdujący się w podkatalogu "test", możemy dodać wyjątek, poprzez zapisanie relatywnej ścieżki do pliku poprzedzonej znakiem !. Przykładowo - chcąc uniknąć śledzenia wszystkich plików w podkatalogu "test" z wyjątkiem pliku "dane.txt", powinniśmy zapisać:

```
test/  
!/test/dane.txt
```

Przykładowy plik .gitignore może więc wyglądać następująco:

```
# Klucz API  
API.txt  
  
# Logi  
*.log  
logs/  
  
# Katalog test (z wyjątkiem dane.txt)  
test/  
!/test/dane.txt
```