

# Systemy kontroli wersji

## Git Część I – Podstawy



Aleksander Lamża  
ZKSB · Instytut Informatyki  
Uniwersytet Śląski w Katowicach

[aleksander.lamza@us.edu.pl](mailto:aleksander.lamza@us.edu.pl)

- Czym jest Git?
- Dokumentacja i zasoby
- Ogólne informacje
- Instalowanie i konfigurowanie
- Git lokalnie. Podstawowe operacje

---

## **Wstępne wymagania**

- Wprowadzenie do systemów kontroli wersji

# Czym jest Git?

## Rozproszony system kontroli wersji

<http://git-scm.com/>

**Rozproszony** – nie jest wymagany ciągły dostęp do centralnego repozytorium.

Nadaje się do **małych i dużych projektów**.

Świetnie sprawdza się też **lokalnie**.



The screenshot shows the Git website homepage. At the top, the Git logo is followed by the tagline "--local-branching-on-the-cheap". A search bar is on the right. The main content area describes Git as a free and open source distributed version control system. It highlights features like being easy to learn, having a tiny footprint, and lightning fast performance. A diagram on the right illustrates the distributed nature of Git with multiple local repositories. Below this, there are four sections: 'About' (advantages of Git), 'Documentation' (command reference, Pro Git book, etc.), 'Downloads' (GUI clients, binary releases), and 'Community' (mailing list, chat, etc.). A 'Pro Git' section mentions it's available on Amazon. On the right, a monitor displays the latest stable release (1.7.12.1) and a 'Download for Linux' button. Below the monitor are links for Linux GUIs, Tarballs, Mac Build, and Source Code. The footer features a 'Companies & Projects Using Git' section with logos for Google, Facebook, Microsoft, Twitter, LinkedIn, Netflix, and others. It also includes logos for various operating systems and frameworks like Android, Ubuntu, Rails, Qt, GNOME, Eclipse, and Xcode. At the very bottom, it states 'This open sourced site is hosted on GitHub' and 'Git is a member of the Software Freedom Conservancy'.



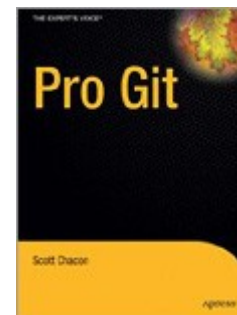
## Reference Manual

<http://git-scm.com/docs>

## Książka „Pro Git”

W wersji elektronicznej za darmo  
(online, PDF, mobi, ePub)

<http://git-scm.com/book>



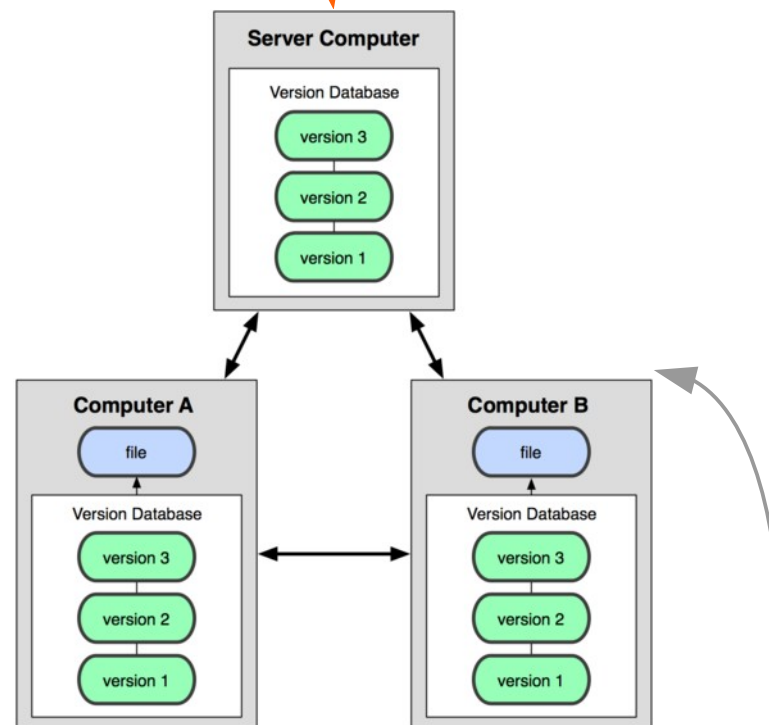
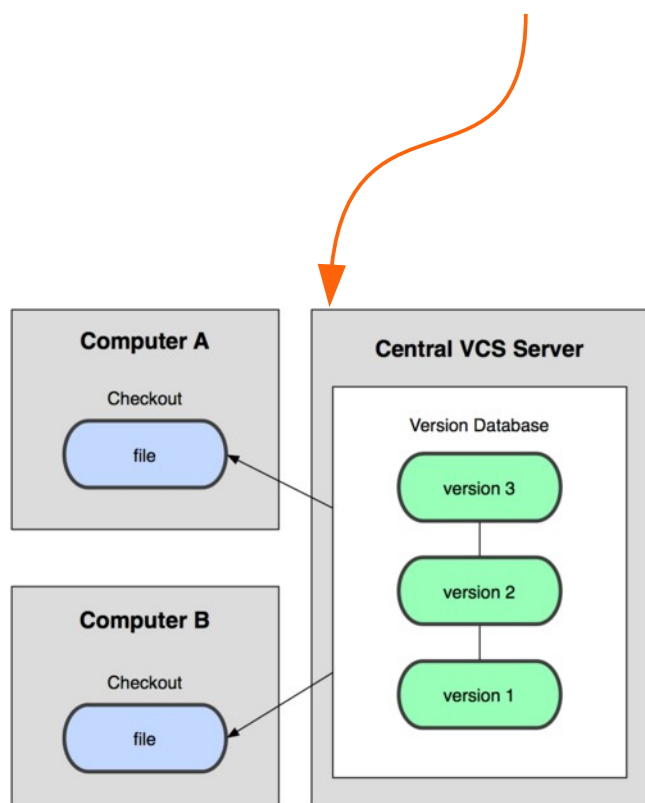
## Filmy, tutoriale itp.

<http://git-scm.com/doc/ext>

<http://git-scm.com/videos>

# Ogólne informacje

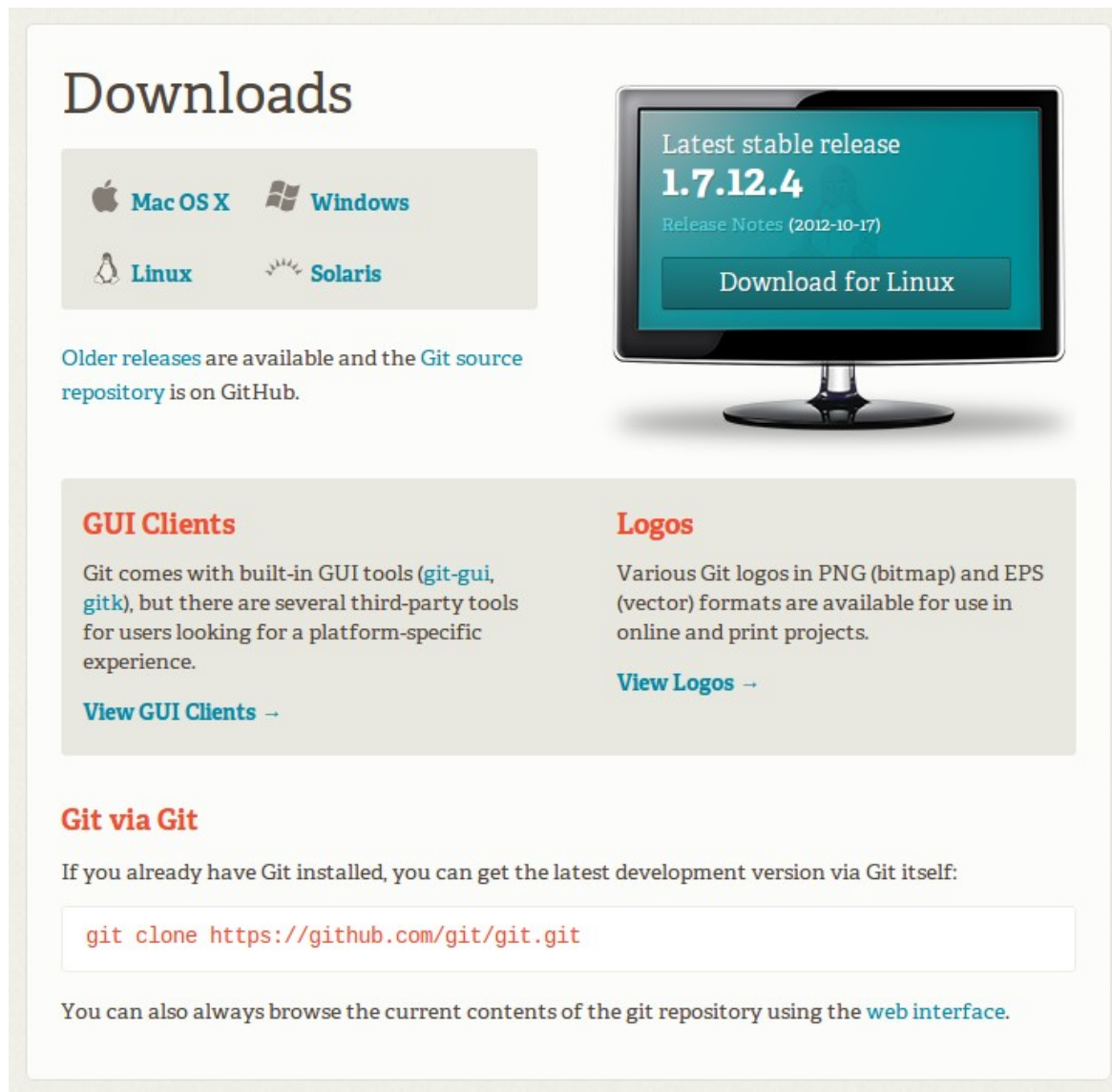
## Porównanie **scentralizowanych** i **rozproszonych** systemów kontroli wersji



Dzięki temu, że na każdym komputerze jest pełne repozytorium, wykonywanie czynności operujących na historii jest o wiele szybsze. Można też pracować offline.

# Instalowanie i konfigurowanie

## Jak zainstalować?

A screenshot of the Git Downloads page. The page has a light beige background. At the top left, the word "Downloads" is in a large, dark font. Below it, there's a section with four operating system icons: Apple logo for "Mac OS X", Windows logo for "Windows", Linux penguin logo for "Linux", and Solaris sun logo for "Solaris". To the right of this is a computer monitor displaying "Latest stable release 1.7.12.4" in white text on a teal background, with a link for "Release Notes (2012-10-17)" and a button that says "Download for Linux". Below the OS icons, a line of text says "Older releases are available and the Git source repository is on GitHub." Further down, there are two columns. The left column is titled "GUI Clients" in red and contains text about built-in GUI tools (git-gui, gitk) and third-party tools, with a link "View GUI Clients →". The right column is titled "Logos" in red and contains text about various Git logos in PNG and EPS formats, with a link "View Logos →". At the bottom, a section titled "Git via Git" in red explains how to get the latest development version via Git itself, showing a terminal command in a light box: "git clone https://github.com/git/git.git". Below this, it says "You can also always browse the current contents of the git repository using the web interface." An arrow from the URL "git-scm.com/downloads" at the bottom left points to the screenshot.

git-scm.com/downloads

# Instalowanie i konfigurowanie

Po zainstalowaniu, trzeba skonfigurować

Przede wszystkim dane użytkownika

```
$ git config --global user.name "Alojzy Git"
$ git config --global user.email "alozzy.git@nail.com"
```

Opcji konfiguracji jest jeszcze sporo,  
ale większości z nich nie trzeba „ruszać”.

```
$ git config --global core.editor "notatnik :)"
```

automatycznie ustawia się domyślny edytor systemowy

Dostępne są trzy poziomy konfiguracji:

- systemowa (`--system`),
- użytkownika (`--global`)
- lokalna (na poziomie repozytorium)

# Git lokalnie. Podstawowe operacje

- Tworzenie repozytorium
- Dodawanie plików
- Zatwierdzanie zmian
- Modyfikowanie plików i kontrola zmian
- Klonowanie repozytorium



# Tworzenie repozytorium

## Tworzymy repozytorium

Zaczynamy od utworzenia katalogu, w którym będziemy przechowywać pliki projektu.

katalog roboczy  
(working directory)

```
$ mkdir projekt  
$ cd projekt
```

**git init**

```
$ git init  
Initialized empty Git repository in ../../projekt/.git/
```

```
.git  
├── branches  
├── config  
├── description  
├── HEAD  
├── hooks  
│   ├── applypatch-msg.sample  
│   ├── commit-msg.sample  
│   ├── post-update.sample  
│   ├── pre-applypatch.sample  
│   ├── pre-commit.sample  
│   ├── prepare-commit-msg.sample  
│   ├── pre-rebase.sample  
│   └── update.sample  
├── info  
│   └── exclude  
├── objects  
│   ├── info  
│   └── pack  
└── refs  
    ├── heads  
    └── tags
```

Przydałby nam się jakikolwiek plik...

# Tworzenie repozytorium

## Tworzymy plik

W katalogu projektu tworzymy jakiś plik...



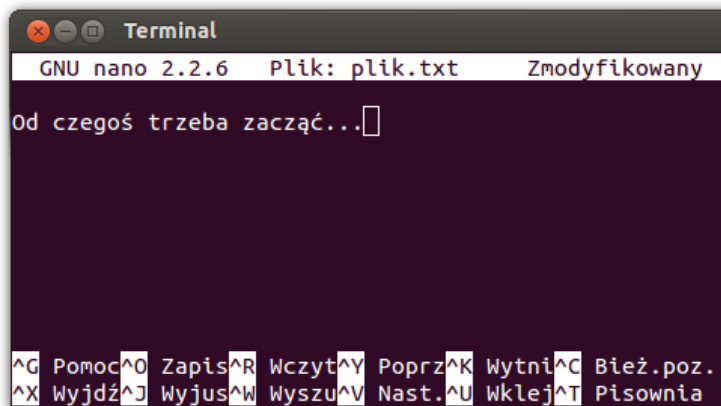

```
$ touch plik.txt
```

...sprawdzamy, czy powstał...



```
$ ls  
plik.txt
```

...i coś w nim wpisujemy,  
a następnie zapisujemy zmiany.



```
Terminal  
GNU nano 2.2.6 Plik: plik.txt Zmodyfikowany  
Od czegoś trzeba zacząć...  
^G Pomoc ^O Zapis ^R Wczyt ^Y Poprz ^K Wytń ^C Bież.poz.  
^X Wyjdź ^J Wyjś ^W Wysz ^V Nast. ^U Wklej ^T Pisownia
```

Teraz sprawdzimy status naszego repozytorium.


## Sprawdzamy status

```
git status
```

Każdy plik w katalogu roboczym znajduje się w jednym z dwóch stanów:

**tracked** (śledzony) lub **untracked** (nieśledzony).

Utworzony przed chwilą plik jest nieśledzony.



```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   plik
nothing added to commit but untracked files present (use "git add" to track)
```

Zgodnie z sugestią (use "git add <file>...") dodamy ten plik poleceniem **add**.



O tym, gdzie trafi ten plik, już za moment.

## Dodajemy pliki

```
git add plik
```

Można też użyć symboli wieloznacznych, np.:  
`git add *.java`

A w ten sposób można dodać cały bieżący katalog:  
`git add .`

I ponownie sprawdzamy status:

```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#    new file:   plik.txt
#
```

To oznacza, że *plik.txt* jest w stanie „śledzony” i znajdzie się w najbliższym commicie.

Przyszła czas na zatwierdzenie zmian!

# Zatwierdzanie zmian


## Zatwierdzamy zmiany

```
git commit -m "komentarz do rewizji"
```

```
$ git commit -m "Pierwszy commit"  
[master (root-commit) c998d05] Pierwszy commit  
1 file changed, 1 insertion(+)  
create mode 100644 plik.txt
```

Rzut oka na status:

```
$ git status  
# On branch master  
nothing to commit (working directory clean)
```

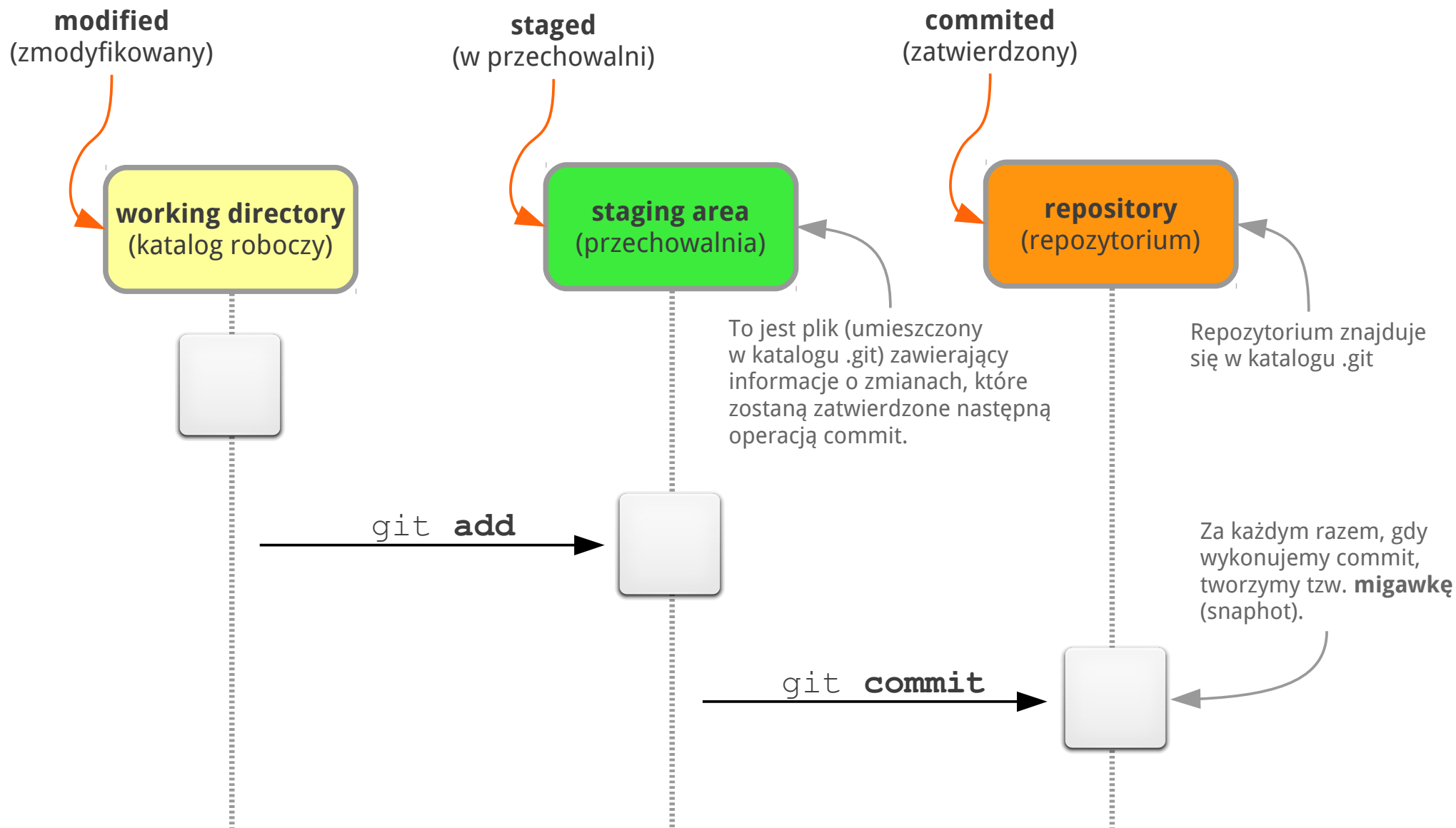


Katalog roboczy jest czysty, tzn. nie zawiera żadnych ślędzonych i zmodyfikowanych plików.

# Typowy „workflow”

## Małe podsumowanie

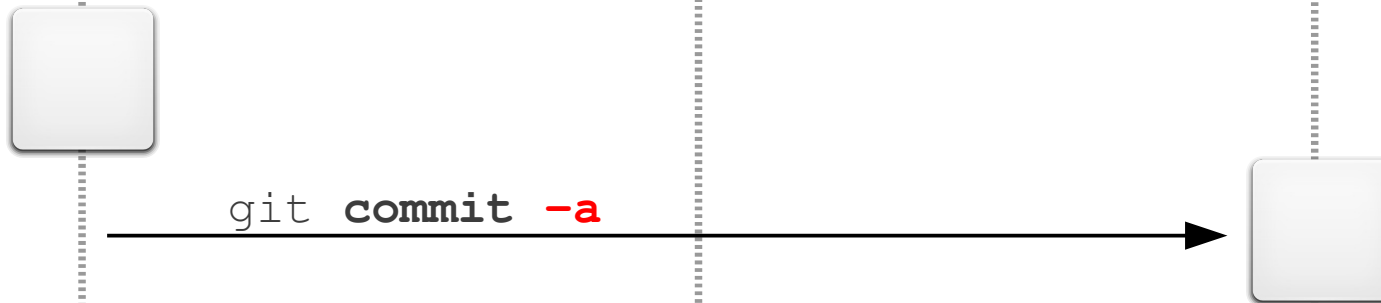
Trzy stany plików i odpowiadające im sekcje projektu:



Typowy sposób pracy wygląda następująco:

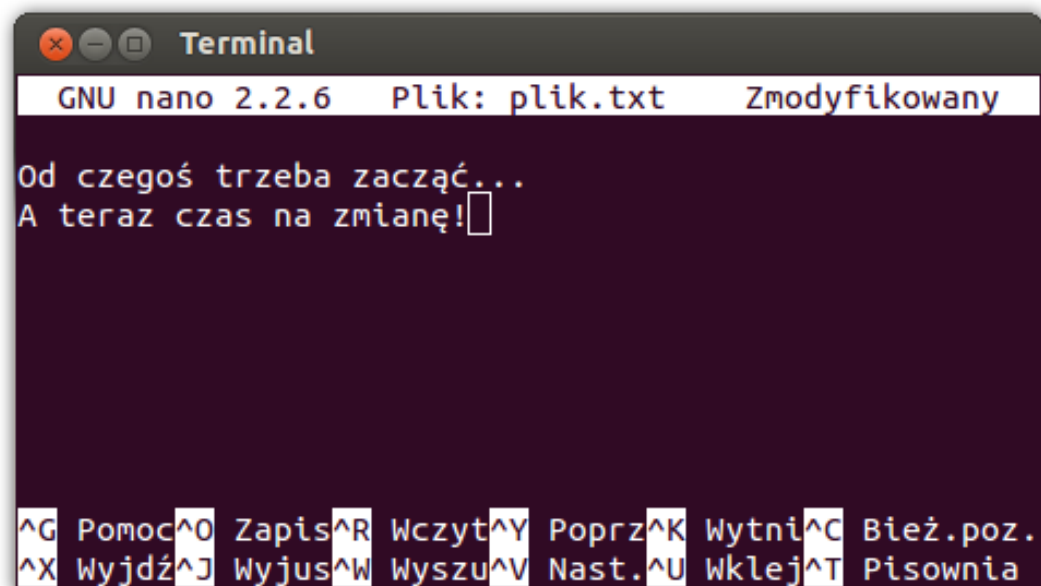
- ❶ Pliki tworzymy i modyfikujemy w **katalogu roboczym**.
- ❷ Jeśli chcemy, by zmiany zostały uwzględnione, pliki przenosimy do **przechowalni**.
- ❸ Gdy chcemy zachować bieżący stan projektu w **repozytorium**, zatwierdzamy zmiany.

Jeśli chcemy, możemy pominąć etap dodawania plików do przechowalni.  
Służy do tego opcja **-a** operacji **commit**:



# Zmiany w repozytorium

Zmiany, zmiany...



```
Terminal
GNU nano 2.2.6  Plik: plik.txt  Zmodyfikowany

Od czegoś trzeba zacząć...
A teraz czas na zmianę!

^G Pomoc^O Zapis^R Wczyt^Y Poprz^K Wytni^C Bież.poz.
^X Wyjdź^J Wyjusz^W Wyszusz^V Nast.^U Wklej^T Pisownia
```

Sprawdzamy status:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   plik.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```



# Zmiany w repozytorium

Hm... Co się zmieniło?

`git diff`

```
$ git diff
diff --git a/plik.txt b/plik.txt
index ae30363..8774b8b 100644
--- a/plik.txt
+++ b/plik.txt
@@ -1,2 @@
   Od czegoś trzeba zacząć...
+A teraz czas na zmianę!
```

Jeszcze jedna mała modyfikacja i ponownie `diff`...

```
$ git diff
diff --git a/plik.txt b/plik.txt
index ae30363..fe7b508 100644
--- a/plik.txt
+++ b/plik.txt
@@ -1,2 @@
-  Od czegoś trzeba zacząć...
+Trzeba zacząć...
+A teraz czas na zmianę!
```

Aby zatwierdzić zmiany, wykonujemy polecenie `git commit`.

# Zmiany w repozytorium

## O różnicach trochę dokładniej

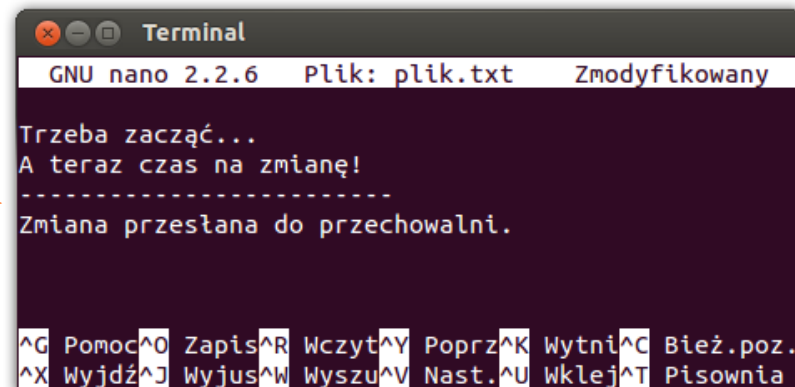
Wprowadzamy zmianę w pliku.

Dodajemy do przechowalni:

```
$ git add plik.txt
```

Znów modyfikujemy plik.

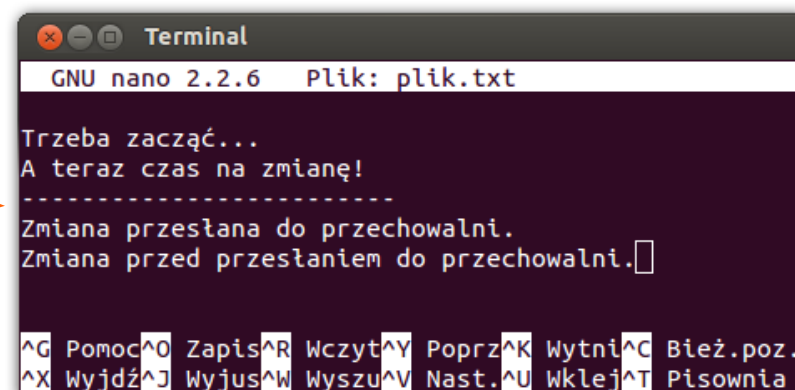
(ale tym razem nie dodajemy pliku do przechowalni)



```
Terminal
GNU nano 2.2.6  Plik: plik.txt  Zmodyfikowany

Trzeba zacząć...
A teraz czas na zmianę!
-----
Zmiana przesłana do przechowalni.

^G Pomoc^O Zapis^R Wczyt^Y Poprz^K Wytni^C Bież.poz.
^X Wyjdź^J Wyjusz^W Wyszusz^V Nast.^U Wklej^T Pisownia
```



```
Terminal
GNU nano 2.2.6  Plik: plik.txt

Trzeba zacząć...
A teraz czas na zmianę!
-----
Zmiana przesłana do przechowalni.
Zmiana przed przesłaniem do przechowalni.

^G Pomoc^O Zapis^R Wczyt^Y Poprz^K Wytni^C Bież.poz.
^X Wyjdź^J Wyjusz^W Wyszusz^V Nast.^U Wklej^T Pisownia
```

Jaki będzie wynik `diff-a`?

# Zmiany w repozytorium

`diff`

Wyświetla zmiany, które nie zostały jeszcze dodane do przechowalni.

```
$ git diff
diff --git a/plik.txt b/plik.txt
index 143dd9f..06e7339 100644
--- a/plik.txt
+++ b/plik.txt
@@ -2,3 +2,4 @@
Trzeba zacząć...
A teraz czas na zmianę!
-----
Zmiana przesłana do przechowalni.
+Zmiana przed przesłaniem do przechowalni.
```

Wyświetla zmiany między przechowalnią a ostatnim commitem.  
(To zmieniliby się w repozytorium, gdyby wykonać `git commit`).

`diff --staged`

od wersji 1.6.1;  
wcześniej `--cached`

```
$ git diff --staged
diff --git a/plik.txt b/plik.txt
index bf0e5f7..143dd9f 100644
--- a/plik.txt
+++ b/plik.txt
@@ -1,3 +1,4 @@
Trzeba zacząć...
A teraz czas na zmianę!
-----
+Zmiana przesłana do przechowalni.
```

`diff HEAD`

```
$ git diff HEAD
diff --git a/plik.txt b/plik.txt
index bf0e5f7..06e7339 100644
--- a/plik.txt
+++ b/plik.txt
@@ -1,3 +1,5 @@
Trzeba zacząć...
A teraz czas na zmianę!
-----
+Zmiana przesłana do przechowalni.
+Zmiana przed przesłaniem do przechowalni.
```

Wyświetla zmiany między katalogiem roboczym a ostatnim commitem.  
(To zmieniliby się w repozytorium, gdyby wykonać `git commit -a`).

## Anuluj, anuluj! – cofanie zmian

bo popsuliśmy :(

A co jeśli chcemy **wycofać** wprowadzone przez nas zmiany (ale jeszcze nie zatwierdzone) ?

Jedna zmiana została już  
dodana do przechowalni.

Rzut oka na status:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   plik.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   plik.txt
#
```

A druga jeszcze nie.

# Cofanie zmian

Jeśli chcemy wycofać **zmianę z katalogu roboczego** (przed dodaniem do przechowalni):

```
git checkout -- <file>
```

Wyciąłem zbędne fragmenty.

```
$ git diff HEAD
[...]
+Zmiana przesłana do przechowalni.
+Zmiana przed przesłaniem do przechowalni.

$ git checkout -- plik.txt

$ git diff HEAD
[...]
+Zmiana przesłana do przechowalni.
```

To jest bardzo niebezpieczna operacja!  
Przywrócenie poprzedniej wersji jest  
nieodwracalne, więc lepiej się dwa razy  
zastanowić...

Lepszym rozwiązaniem jest intensywne  
wykorzystanie **gałęzi** (branches)  
i ewentualnie użycie polecenia  
`git stash`.

(O tym trochę później...)

# Cofanie zmian

Jeśli chcemy wycofać **zmianę z przechowalni**, musimy ją najpierw „wyciągnąć”:

```
git reset HEAD <file>
```

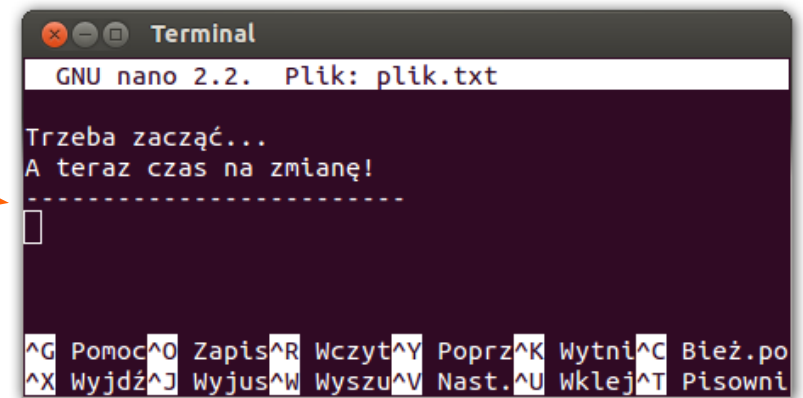
```
$ git reset HEAD plik.txt
Unstaged changes after reset:
M   plik.txt

$ git checkout -- plik.txt

$ git diff HEAD
$
```

Pusto, czyli zmiany zostały wycofane...

...co widać też tu.



```
GNU nano 2.2. Plik: plik.txt

Trzeba zacząć...
A teraz czas na zmianę!
-----
^G Pomoc^O Zapis^R Wczyt^Y Poprz^K Wytni^C Bież.po
^X Wyjdź^J Wyjusz^W Wyszuz^V Nast.^U Wklej^T Pisowni
```

# Zatwierdzanie zmian

## Zatwierdzanie zatwierdzonego

Przypuśćmy, że wykonaliśmy commit, ale nie uwzględniliśmy wszystkich pożądaných zmian.


Utworzyliśmy dodatkowy plik (drugi.txt),  
ale przed commitem nie wykonaliśmy dla niego `git add`:

```
$ touch drugi.txt

$ ls
plik.txt  drugi.txt

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   plik.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       drugi.txt
nothing added to commit but untracked files present (use "git add" to track)

$ git commit -m "Kolejny commit"
[master 8dcf6e8] Kolejny commit
1 file changed, 1 insertion(+)
```



Na przykład  
nie dodaliśmy  
utworzonych lub  
zmodyfikowanych  
plików.

# Zatwierdzanie zmian

Zorientowaliśmy się, że w commicie nie ma pliku:

```
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   drugi.txt
nothing added to commit but untracked files present (use "git add" to track)
```

Więc szybko naprawiamy błąd:

```
git commit --amend
```

Nie możemy zapomnieć o dodaniu  
pominiętego wcześniej pliku!

```
$ git add drugi.txt

$ git commit --amend -m "Kolejny commit"
[master 9116596] Kolejny commit
1 file changed, 1 insertion(+)
create mode 100644 drugi.txt

$ git status
# On branch master
nothing to commit (working directory clean)
```

Dzięki temu w repozytorium nie będziemy  
mieć nieprawidłowej migawki.



# Historia zatwierdzania

## Historia zatwierdzania

**git log**

```
$ git log
```

```
commit 9116596a0d108118b5e5805c6d95ec127f20f951
Author: Olek Lamża <olek.lamza@gmail.com>
Date:   Fri Sep 28 13:13:44 2012 +0200
```

Kolejny commit

```
commit e8786d5cf9404c075d6c5d038a48ac2c5af03bc5
Author: Olek Lamża <olek.lamza@gmail.com>
Date:   Fri Sep 28 09:40:52 2012 +0200
```

Drugi commit

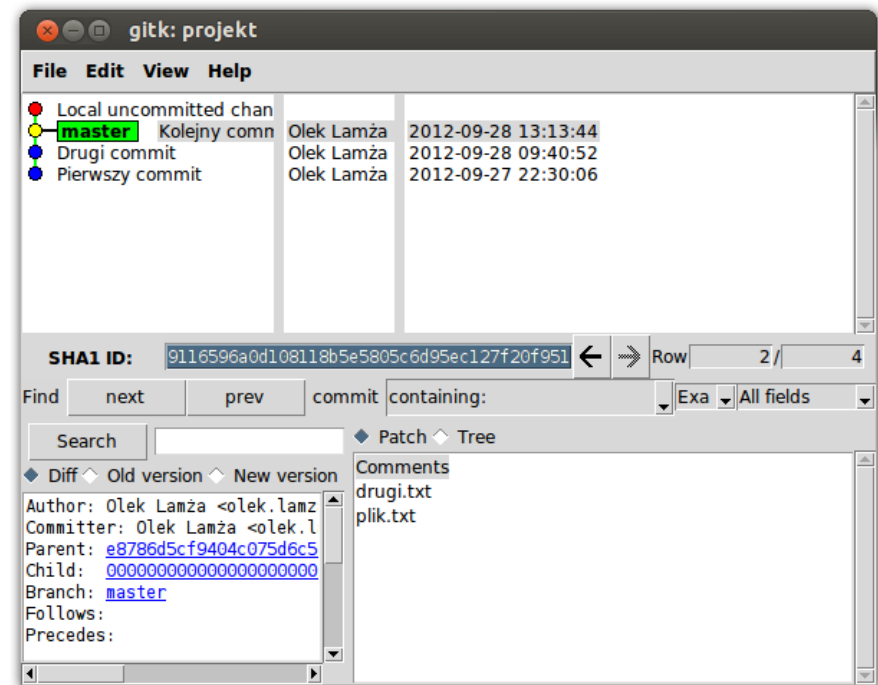
```
commit c998d05aed02349bb2011ac8882b715b8a756b79
Author: Olek Lamża <olek.lamza@gmail.com>
Date:   Thu Sep 27 22:30:06 2012 +0200
```

Pierwszy commit

Po dołączeniu opcji **-p** zostaną wyświetlone wszystkie zmiany wprowadzone w poszczególnych migawkach.

User friendlier? ;)

**gitk**



## Co dalej?

- Gałęzie
- Praca zespołowa
- GitHub