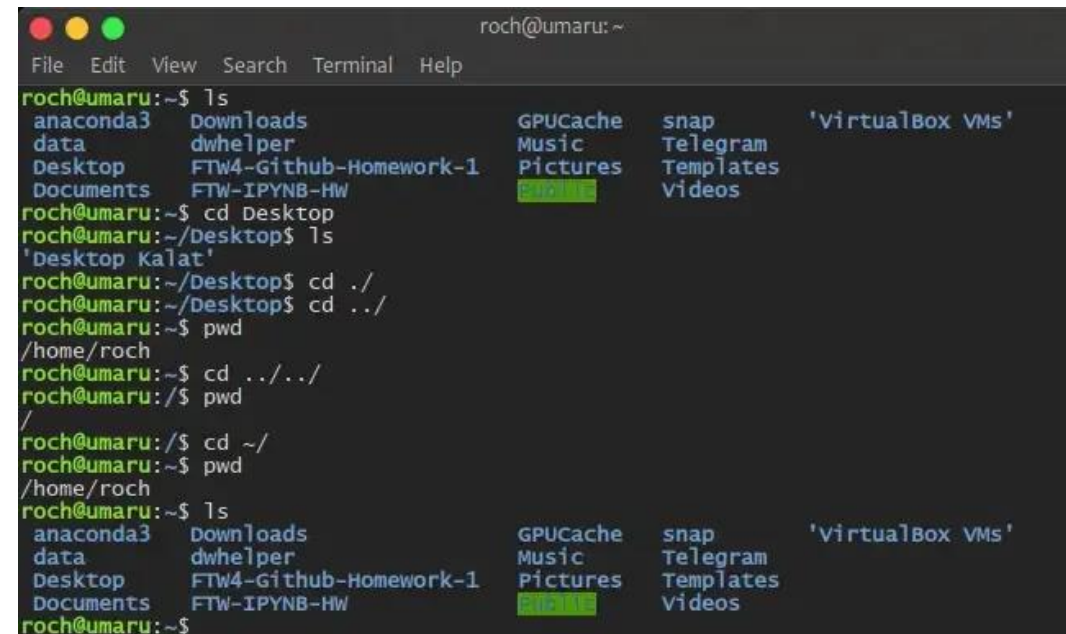


Narzędzia systemowe i system kontroli wersji GIT

Terminal

Jakie funkcjonalności posiada terminal?

- komunikuje się z systemem operacyjnym za pomocą poleceń.
- przyspiesza pracę
- wykorzystuje CMD, PowerShell, WSL lub Git Bash w systemie Windows
- zarządza systemem plików oraz instaluje oprogramowanie



```
roch@umaru: ~  
File Edit View Search Terminal Help  
roch@umaru:~$ ls  
anaconda3 Downloads GPUcache snap 'VirtualBox VMs'  
data dwheper Music Telegram  
Desktop FTW4-Github-Homework-1 Pictures Templates  
Documents FTW-IPYNB-HW STAGE Videos  
roch@umaru:~$ cd Desktop  
roch@umaru:~/Desktop$ ls  
'Desktop Kalat'  
roch@umaru:~/Desktop$ cd ./  
roch@umaru:~/Desktop$ cd ../  
roch@umaru:~$ pwd  
/home/roch  
roch@umaru:~$ cd ../../  
roch@umaru:/$ pwd  
/  
roch@umaru:/$ cd ~/  
roch@umaru:~$ pwd  
/home/roch  
roch@umaru:~$ ls  
anaconda3 Downloads GPUcache snap 'VirtualBox VMs'  
data dwheper Music Telegram  
Desktop FTW4-Github-Homework-1 Pictures Templates  
Documents FTW-IPYNB-HW STAGE Videos  
roch@umaru:~$
```

Źródło: [link](#)

System plików z poziomu konsoli

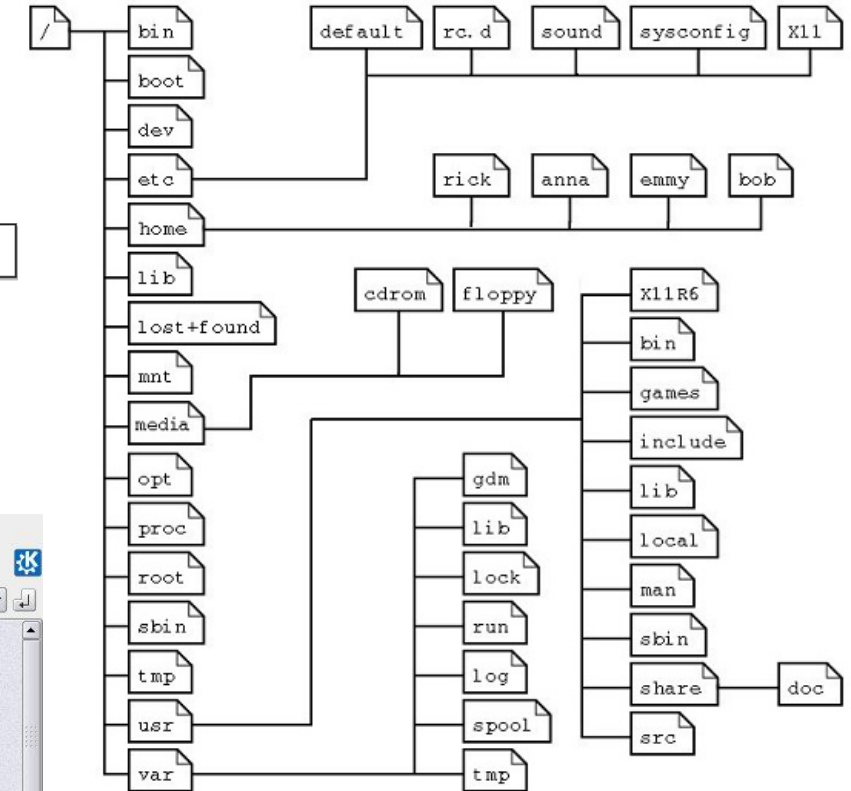
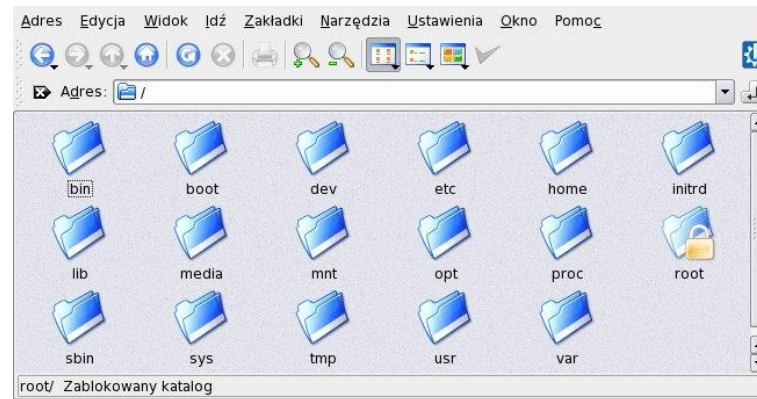
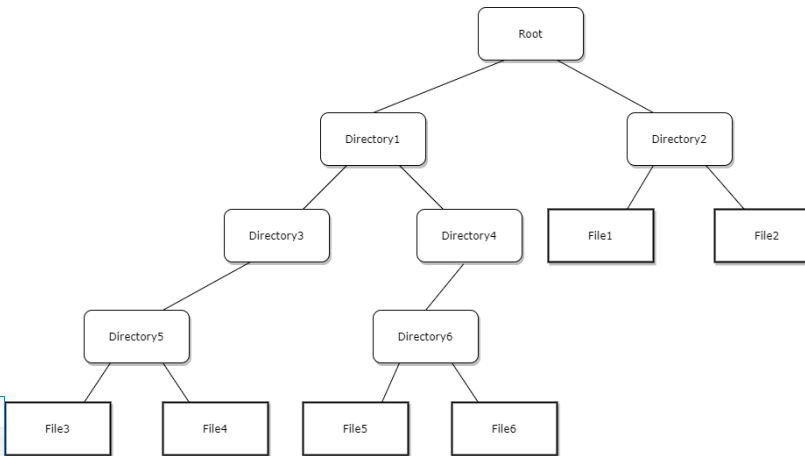
Jak wygląda to na
Twoim systemie?

```

pi@raspberrypi: ~
GNU nano 2.7.4 File: drzewko.txt

bzip2recover
bzless -> bzmore
bzmore
cat
chgrp
chmod
chown
chvt
con2fbmap
cp
cpio
daash
date
dd
df
dir
dmesg
dnsdomainname -> hostname
domainname -> hostname
dumpkeys
echo
ed

```



Podstawowe polecenia

cd – zmienia bieżący katalog na inny (od 'change directory')

- *cd dirname* – zmienia aktualny katalog na 'dirname'
- *cd dir1/dir2/dir3* – wchodzi do katalogu 'dir3', który jest w katalogu 'dir2', który jest w 'dir1'
- *cd* – z dowolnego miejsca, zmienia katalog na domowy
- *cd ..* – przechodzi do katalogu o jeden wyższego w drzewie katalogów niż obecny
- *cd /home/dir* – z dowolnego miejsca, przechodzi do katalogu zaczynając od początku drzewa: /
- *cd -* – przechodzi do poprzedniego katalogu

Podstawowe polecenia

ls – wyświetla zawartość katalogu

- *ls* – listuje katalog . (*ls .*)
- *ls plik1 plik2 plik3* – listuje tylko wymienione pliki
- *ls *.txt* – wypisze wszystkie pliki o nazwie kończącej się na '.txt'
- *ls katalog1 katalog2* – listuje wymienione katalogi
- *ls -l* – szczegółowa lista
- *ls -a* – wypisuje również ukryte pliki (czyli te których nazwa zaczyna się kropką)
- *ls -R* – listuje katalogi rekursywnie (czyli wyświetla również zawartość podkatalogów)
- *ls -d* – wyświetla tylko nazwy katalogów, tak jak zwyczajnych plików, czyli nie listuje ich zawartości

Podstawowe polecenia

pwd – wypisuje ścieżkę obecnego katalogu (od 'print working directory')

cat – wypisuje wszystkie podane mu pliki na standardowe wyjście

- *cat plik* – jeśli nie przekierujemy standardowego wyjścia do innego pliku (>, >>) lub programu (|), to wypisze plik na ekran
- *cat plik1 plik2 plik3* – wypisze po kolei zawartość wszystkich plików

touch – zmienia czas dostępu i modyfikacji pliku, lub jeśli plik nie istnieje - tworzy go.

- *touch plik*

Podstawowe polecenia

cp – kopiuje plik

- *cp plik1 plik2* – stworzy ./plik2 identyczny z plik1
- *cp plik3 ../katalog/* – stworzy plik ../katalog/plik3
- *cp pom.* podkatalog/* – kopiuje wszystkie pliki zaczynające się na 'pom.' do ./podkatalog/
- *cp plik5 ~/katalog/jakis/pliczek* – stworzy plik ~/katalog/projekt/plik

mv – przesuwa plik (tym samym służy również do zmiany nazwy)

- *mv plik1 plik2* – zmieni nazwę pliku z ./plik1 na plik2
- *mv plik3 ../katalog/jakis/* – przesunie plik do ../katalog/jakis/plik3
- *mv plik4 podkatalog/* – przesunie plik ./podkatalog/plik4
- *mv plik5 ~/katalog/* – przesunie i zmieni nazwę ~/katalog/projekt/plik5

Podstawowe polecenia

rm – usuwanie plików/katalogów

- *rm file* - kasuje plik
- *rm -r directory* – kasuje wszystko w katalogu i wszystkie jego podkatalogi (--recursive)
- *rm -f file* – nie pyta się czy skasować (--force)

mkdir – tworzy katalog

- *mkdir new_katalog*
- *mkdir /home/users/name/new_katalog*

rmdir – usuwa pusty katalog

- *rmdir directory*

Edycja plików z poziomu konsoli - vim

Vim (skrót od **ang. vi improved**) – wieloplatformowy klon edytora tekstu *vi*, napisany przez holenderskiego programistę Bramę Molenaara. Vim należy do grupy **wolnego oprogramowania** o **otwartym kodzie źródłowym**. Pierwsza wersja została wydana w 1991 roku.

Vim jest przede wszystkim edytorem modalnym, tzn. mającym więcej niż jeden tryb pracy. Zależnie od sposobu liczenia, Vim ma od trzech trybów „elementarnych” (**NORMAL**, **INSERT** i **COMMAND-LINE/EX**) do jedenastu. Według twórcy Vima można wyróżnić aż 6 trybów podstawowych (*basic modes*) i 5 dodatkowych (*additional modes*). Ich opis znajduje się w bardzo obszernej dokumentacji podręcznej, dostępnej z poziomu Vima. Wprowadzenie w terminologię i specyfikę trybów uzyskuje się po wpisaniu `:help vim-modes` w tzw. trybie *Ex*. Przejście w tryb *Ex* zawsze zaczyna się od dwukropka (poprzedzonego wciśnięciem klawisza `Esc` w przypadku trybu wyjściowego innego niż **NORMAL**).

Do pobrania: <https://i.imgur.com/YLlnLIY.png>

Edycja plików z poziomu konsoli - vim

Zadania z vim:

- tworzenie plików
- otwieranie plików
- zapis do pliku
- nawigacja po pliku
- wyszukiwanie
- zamiana
- pluginy

System kontroli wersji

Jest to oprogramowanie służące do śledzenia zmian głównie w kodzie źródłowym oraz do pomocy programistom w łączeniu zmian dokonanych w plikach przez wiele osób w różnym czasie.

Systemy kontroli wersji umożliwiają (ang. VCS, Version Control System):

- śledzenie wszystkich zmian dokonywanych na pliku/plikach
- przywołanie dowolnej wcześniejszej wersji pliku/plików
- porównywanie wprowadzonych zmian
- łączenie zmian dokonanych na plikach



GIT

Git – rozproszony system kontroli wersji. Stworzył go Linus Torvalds jako narzędzie wspomagające rozwój jądra Linux. Git stanowi wolne oprogramowanie i został opublikowany na licencji GNU GPL w wersji 2.

Linus Torvalds – autor Linuxa.

Linux - rodzina uniksopodobnych systemów operacyjnych opartych na jądrze Linux. Linux jest jednym z przykładów wolnego i otwartego oprogramowania (jego kod źródłowy może być dowolnie wykorzystywany, modyfikowany i rozpowszechniany).



<https://www.kernel.org/>



Rozpoczęcie pracy z GIT

Zalety i pojęcia GITa

Pozwala m.in. na:

- śledzenie zmian w plikach
- łatwy powrót do wcześniejszej wersji pliku
- sprawną zdalną współpracę

Pojęcia:

- **repozytorium** - projekt na lokalnej maszynie (katalog), w którym zostało zainicjowane repozytorium git, repozytorium może być również zdalne np. <https://github.com/pandas-dev/pandas>
- **commit** - opisanie zmian, które trafią do repozytorium; za pomocą commitów możemy cofać się w historii modyfikacji plików
- **pull/push** - pobranie / wysłanie zmian do repozytorium zdalnego
- **branch** - gałęzie, w momencie jeżeli pracujemy w kilka osób każdy może mieć swojego brancha dzięki temu nie pracujemy razem nad jednym plikiem
- **merge** - proces scalania gałęzi

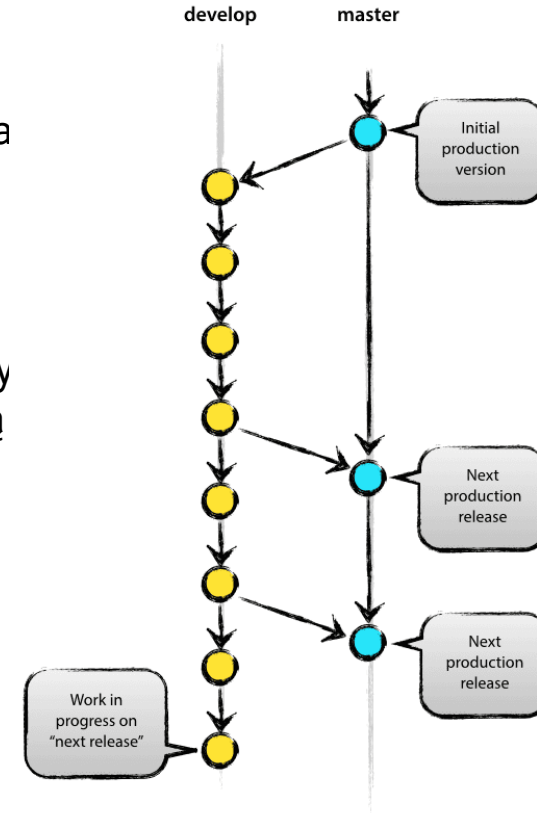
Praca z GitFlow

Proces został opisany przez Vincenta Driessen'a (<https://nvie.com/posts/a-successful-git-branching-model/>), z założenia ma ułatwić pracę z repozytorium (najczęściej w przypadku większych projektów).

Zmian w kodzie nie robimy na głównym branchu **master**, w tym celu tworzymy oddzielną gałąź developerską np. **develop**. W momencie kiedy zmiany na **develop** będą pełne i działające, scalamy je z główną gałęzią **master**.

Rodzaje gałęzi:

- Feature branches
- Release branches
- Hotfix branches



Praca z GitFlow

Feature branches:

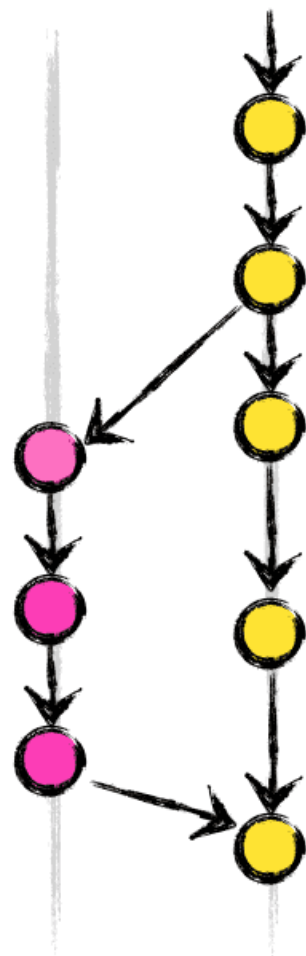
Wykorzystujemy do tworzenia nowych funkcjonalności w projekcie, powinniśmy je tworzyć z brancha developerskiego.

Tworzenie:

`git checkout -b feature\customfeature develop`

feature
branches

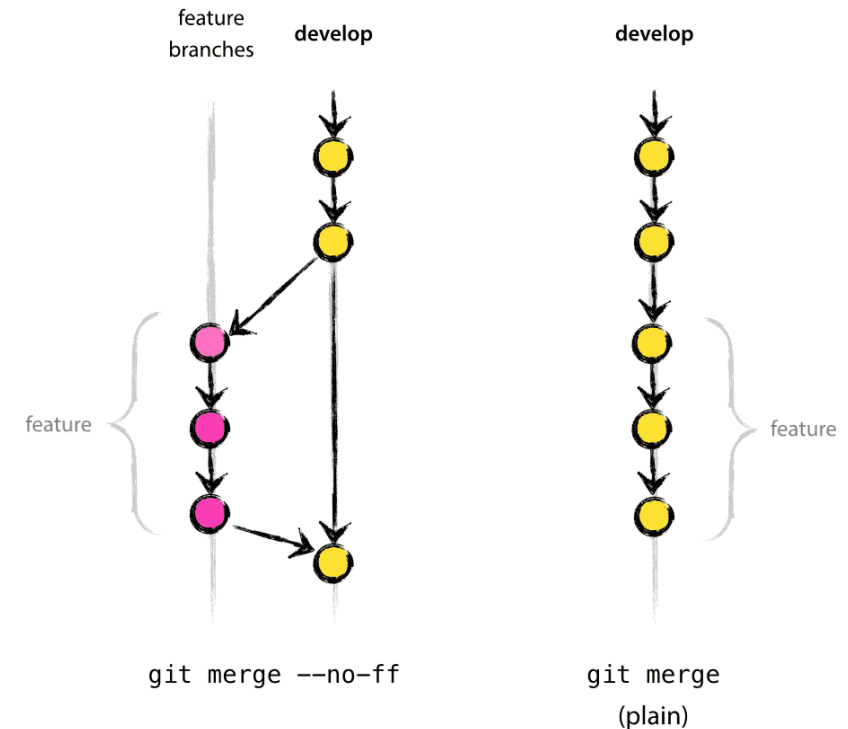
develop



Praca z GitFlow

Feature branches:

```
git checkout develop  
git merge --no-ff feature\customfeature  
git branch -d feature\customfeature  
git push origin develop
```



Praca z GitFlow

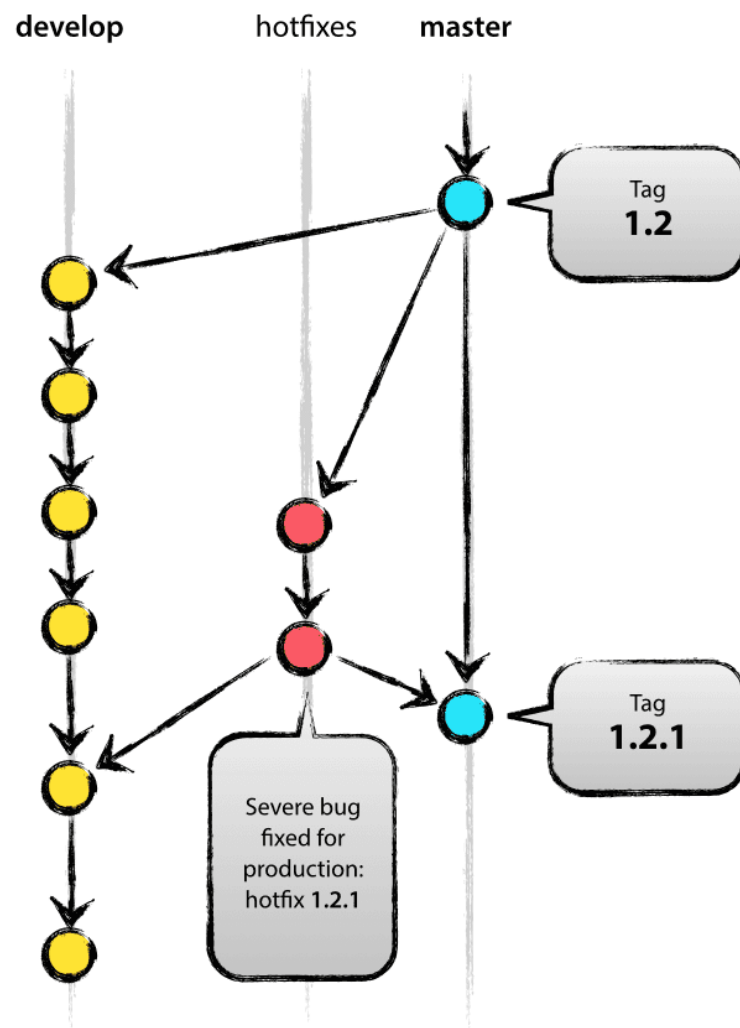
Release branches:

Służą do przygotowywania nowych wydań produkcyjnych. Nie zawsze są wykorzystywane - wszystko zależy od specyfiki firmy i projektu. Tworzenie gałęzi release następuje z brancha developerskiego.

Praca z GitFlow

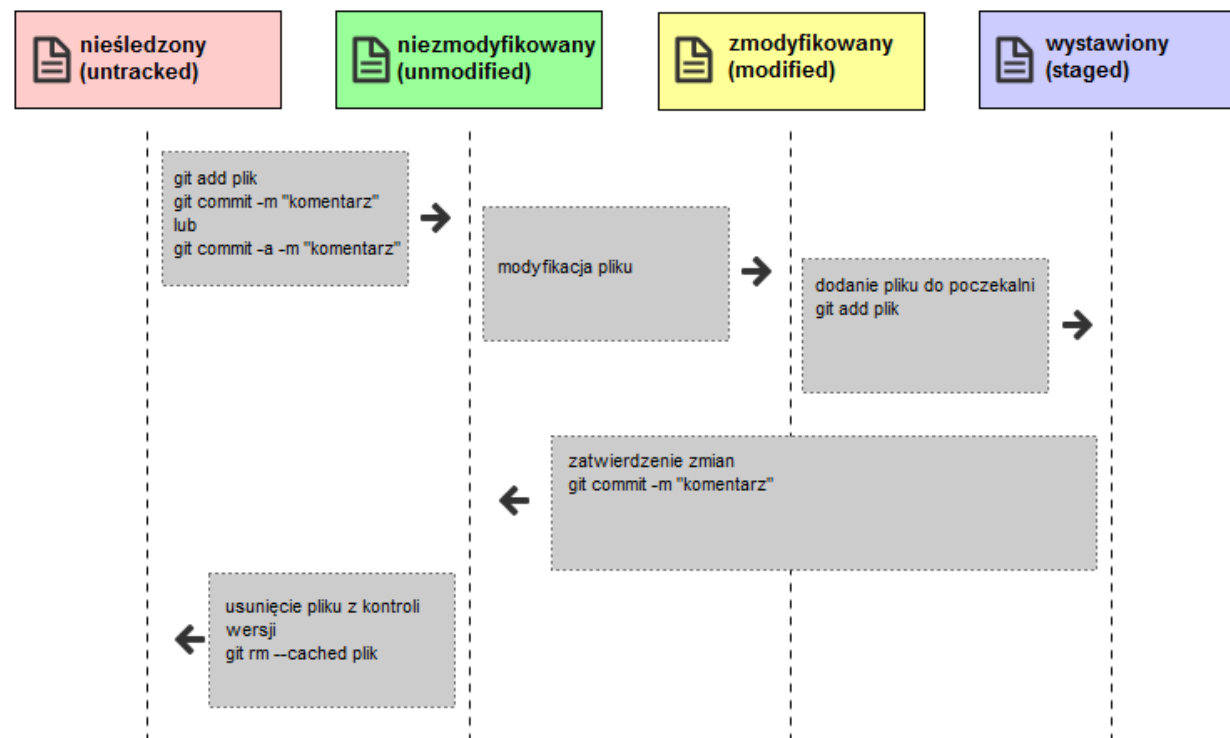
Hotfix branches:

Służą do przygotowywania poprawek.

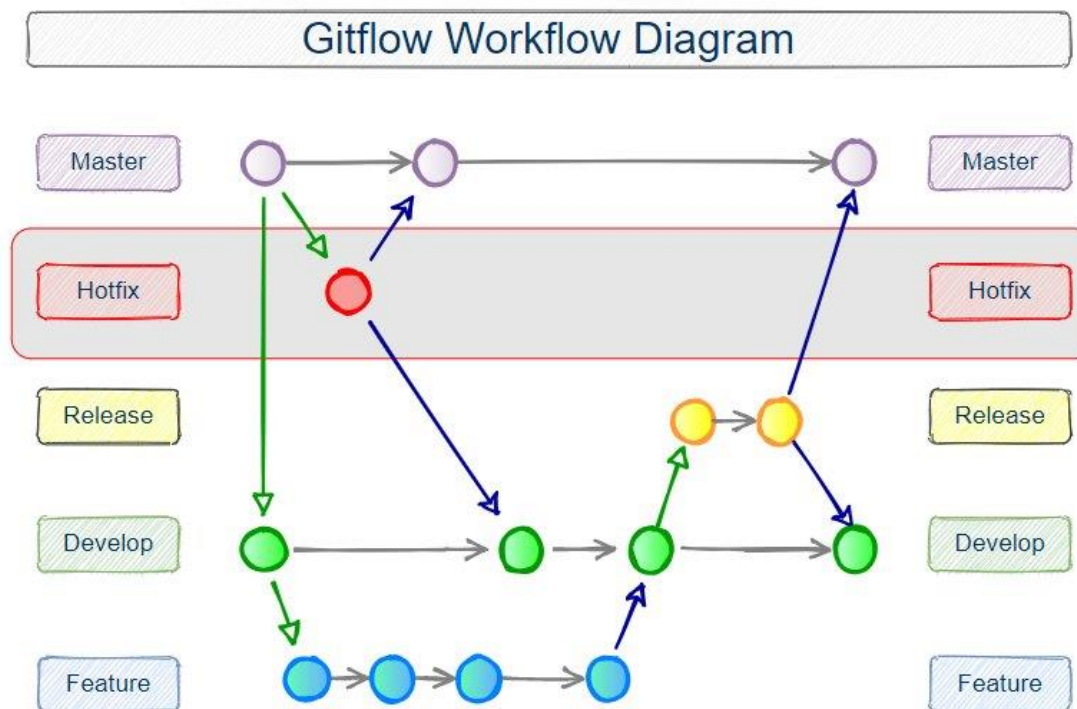


Stany plików

- 1. Nie śledzony
- 1. Niemodyfikowany
- 1. Zmodyfikowany
- 1. Zatwierdzony



GitFlow



Źródło: [link](#)

Git GUI

- Github Desktop.
- SourceTree.
- GitKraken.
- SmartGit.
- Git Cola.
- GitForce.

Git konfiguracja

Do konfiguracji GITa służy komenda git config. Może ona przyjmować jedną z opcji:

- local (konfiguracja dla każdego repozytorium)
- global (konfiguracja dla każdego użytkownika)
- system (konfiguracja dla całego systemu)

Możliwe są również inne opcje, które zmieniają działanie komendy:

- unset name
- list

Git polecenia

Inicjalizacja repozytorium

git init

Dodanie plików do repozytorium

git add .

Commit plików

git commit -m „add final files”

git commit --amend – nadpisuje ostatni commit (poprawa czegoś)

Dodanie adresu repozytorium

git remote add origin <http://github.com/user/myrepo.git>

Push plików

git push -u origin master

GIT Branches

git branch - **listuje wszystkie lokalne gałęzie**

git checkout -b feature/1 - **utworzenie i przejście do gałęzi feature/1**

git checkout develop - **przejście do gałęzi develop**

git branch -d feature/1 - **usunięcie gałęzi feature/1**

GIT polecenia

Klonowanie istniejącego repozytorium

git clone <http://github.com/user/repo.git>

Sprawdzenie stanu plików

git status

Dodanie jednego pliku

git add README

Lista różnic

git diff

Nowy branch

git branch test

Przełączenie się na nowego brancha

git checkout test

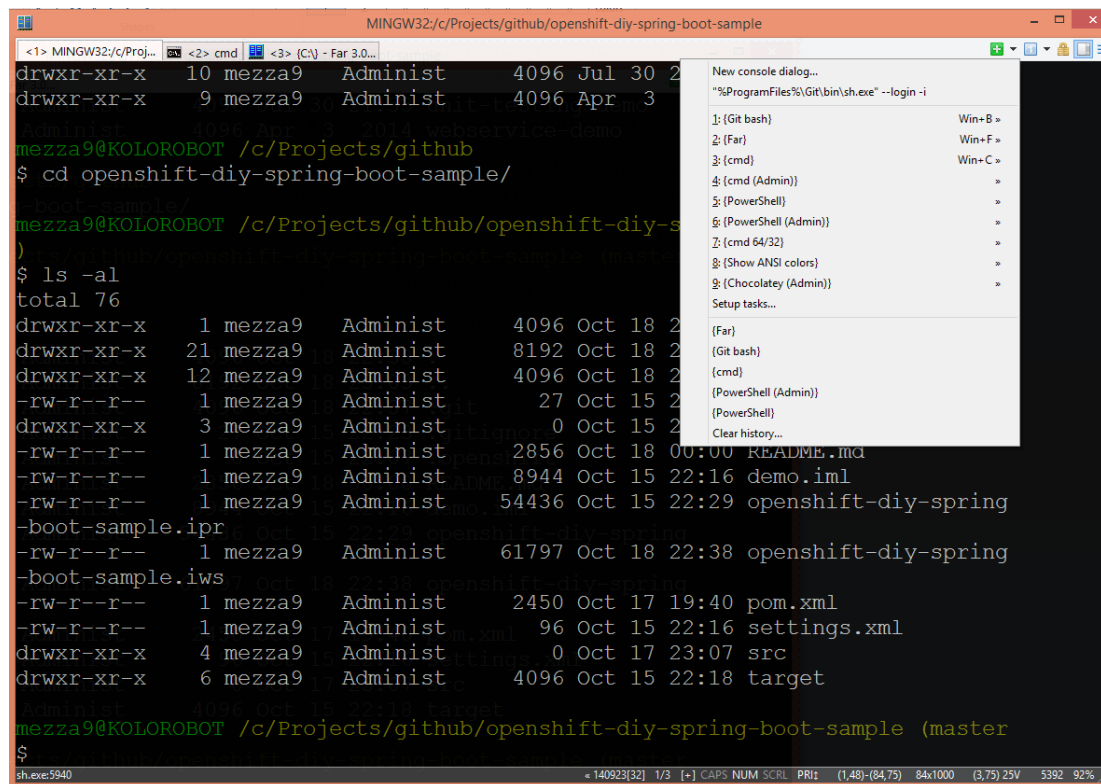
Merge z masterem

git merge test

ConEmu

Alternatywny klient terminala na system Windows.

Pozwala między innymi tworzyć karty. <https://conemu.github.io/>



```
MINGW32/c/Projects/github/openshift-diy-spring-boot-sample
<1> MINGW32/c/Proj... <2> cmd <3> [C:\] - Far 3.0...
drwxr-xr-x 10 mezza9 Administ 4096 Jul 30 2
drwxr-xr-x 9 mezza9 Administ 4096 Apr 3
mezza9@KOLOROBOT /c/Projects/github
$ cd openshift-diy-spring-boot-sample/
mezza9@KOLOROBOT /c/Projects/github/openshift-diy-s
)
$ ls -al
total 76
drwxr-xr-x 1 mezza9 Administ 4096 Oct 18 2
drwxr-xr-x 21 mezza9 Administ 8192 Oct 18 2
drwxr-xr-x 12 mezza9 Administ 4096 Oct 18 2
-rw-r--r-- 1 mezza9 Administ 27 Oct 15 2
drwxr-xr-x 3 mezza9 Administ 0 Oct 15 2
-rw-r--r-- 1 mezza9 Administ 2856 Oct 18 00:00 README.md
-rw-r--r-- 1 mezza9 Administ 8944 Oct 15 22:16 demo.iml
-rw-r--r-- 1 mezza9 Administ 54436 Oct 15 22:29 openshift-diy-spring
-boot-sample.ipr
-rw-r--r-- 1 mezza9 Administ 61797 Oct 18 22:38 openshift-diy-spring
-boot-sample.iws
-rw-r--r-- 1 mezza9 Administ 2450 Oct 17 19:40 pom.xml
-rw-r--r-- 1 mezza9 Administ 96 Oct 15 22:16 settings.xml
drwxr-xr-x 4 mezza9 Administ 0 Oct 17 23:07 src
drwxr-xr-x 6 mezza9 Administ 4096 Oct 15 22:18 target
mezza9@KOLOROBOT /c/Projects/github/openshift-diy-spring-boot-sample (master
$
sh.exe:5940
```


WSL - Windows Subsystem for Linux

- <https://devpebe.com/2019/11/22/dlaczego-programista-powinien-uzywac-wsl-v2-czyli-linux-na-windowsie-10/>
- <https://docs.microsoft.com/pl-pl/windows/wsl/wsl2-index>

Edycja plików z poziomu konsoli - vim

Vim (skrót od ang. vi improved) – wieloplatformowy klon edytora tekstu vi, napisany przez holenderskiego programistę Bramę Moolenaar. Vim należy do grupy wolnego oprogramowania o otwartym kodzie źródłowym. Pierwsza wersja została wydana w 1991 roku.

Vim jest przede wszystkim edytorem modalnym, tzn. mającym więcej niż jeden tryb pracy. Zależnie od sposobu liczenia, Vim ma od trzech trybów „elementarnych” (**NORMAL**, **INSERT** i **COMMAND-LINE/EX**) do jedenastu. Według twórcy Vima można wyróżnić aż 6 trybów podstawowych (*basic modes*) i 5 dodatkowych (*additional modes*). Ich opis znajduje się w bardzo obszernej dokumentacji podręcznej, dostępnej z poziomu Vima. Wprowadzenie w terminologię i specyfikę trybów uzyskuje się po wpisaniu :help vim-modes w tzw. trybie Ex. Przejście w tryb Ex zawsze zaczyna się od dwukropka (poprzedzonego wciśnięciem klawisza Esc w przypadku trybu wyjściowego innego niż NORMAL).

Skróty klawiszowe: <https://i.imgur.com/YLInLIY.png>

Edycja plików z poziomu konsoli - vim

Modes	
ESC	Command mode, toggles modes when in insert or visual mode
i	insert mode
v	Visual mode, start highlighting characters
V	Visual mode, start highlighting lines

Shell Command	
:shell	Opens command prompt
exit	Exits command prompt

Files	
:e	Open a new file filename
:w	Save changes to a file filename
:q	Exit Vim. If there is unsaved files, Vim will not exit
:q!	Exit Vim without saving changes
:x or zz	Exits Vim and saves changes to the file if changes were made

Yank (Copy), Delete (Cut) and Put (Paste)	
yy	Yank the current line
:y	Yank the current line
y	Yank the highlighted text
:d or dd	Delete current line
D	Delete to the end of the line
d	Delete the highlighted text
dw	Delete word
dl	Delete character at cursor position
p	Put text after cursor position, put lines below current line
P	Put text before cursor position, put lines above current line
x	Delete current character

Moving Around	
h or left	Move LEFT one character
j or down	Move DOWN one line
k or up	Move UP one line
l or right	Move RIGHT one line
H	Move to the FIRST line of the screen
M	Move to the MIDDLE line of the screen
L	Move to the LAST line of the screen
b	Move to the BEGINNING of the word
B	Move to the BEGINNING of a blank delimited word
e	Move to the END of the word
E	Move to the END of black delimited word
w	Move to the NEXT word
W	Move to the NEXT blank delimited word
:n	Jump to line n (jump to line 42->:42)
To move up 9 lines type: 9k	

Polecane materiały związane z tematyką bloku

- <https://learngitbranching.js.org/>
- <https://try.github.io/>
- <https://www.katacoda.com/courses/git>
- <https://git-scm.com/book/en/v2>
- <http://index-of.es/Varios-2/How%20Linux%20Works%20What%20Every%20Superuser%20Should%20Know.pdf>

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows

<https://windows.github.com>

GitHub for Mac

<https://mac.github.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms

<http://git-scm.com>

CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```

Creates a new local repository with the specified name

```
$ git clone [url]
```

Downloads a project and its entire version history

MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```

Lists all new or modified files to be committed

```
$ git diff
```

Shows file differences not yet staged

```
$ git add [file]
```

Snapshots the file in preparation for versioning

```
$ git diff --staged
```

Shows file differences between staging and the last file version

```
$ git reset [file]
```

Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```

Records file snapshots permanently in version history

GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```

Lists all local branches in the current repository

```
$ git branch [branch-name]
```

Creates a new branch

```
$ git checkout [branch-name]
```

Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```

Combines the specified branch's history into the current branch

```
$ git branch -d [branch-name]
```

Deletes the specified branch

REFACTOR FILENAMES

Relocate and remove versioned files

```
$ git rm [file]
```

Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```

Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```

Changes the file name and prepares it for commit

SUPPRESS TRACKING

Exclude temporary files and paths

```
*.log  
build/  
temp-*
```

A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```

Lists all ignored files in this project

SAVE FRAGMENTS

Shelve and restore incomplete changes

```
$ git stash
```

Temporarily stores all modified tracked files

```
$ git stash pop
```

Restores the most recently stashed files

```
$ git stash list
```

Lists all stashed changesets

```
$ git stash drop
```

Discards the most recently stashed changeset

REVIEW HISTORY

Browse and inspect the evolution of project files

```
$ git log
```

Lists version history for the current branch

```
$ git log --follow [file]
```

Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```

Shows content differences between two branches

```
$ git show [commit]
```

Outputs metadata and content changes of the specified commit

REDO COMMITS

Erase mistakes and craft replacement history

```
$ git reset [commit]
```

Undoes all commits after [commit], preserving changes locally

```
$ git reset --hard [commit]
```

Discards all history and changes back to the specified commit

SYNCHRONIZE CHANGES

Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```

Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```

Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```

Uploads all local branch commits to GitHub

```
$ git pull
```

Downloads bookmark history and incorporates changes

Dziękujemy!