




**Zarządzanie konfiguracją  
oprogramowania (SCM)**

**Git**

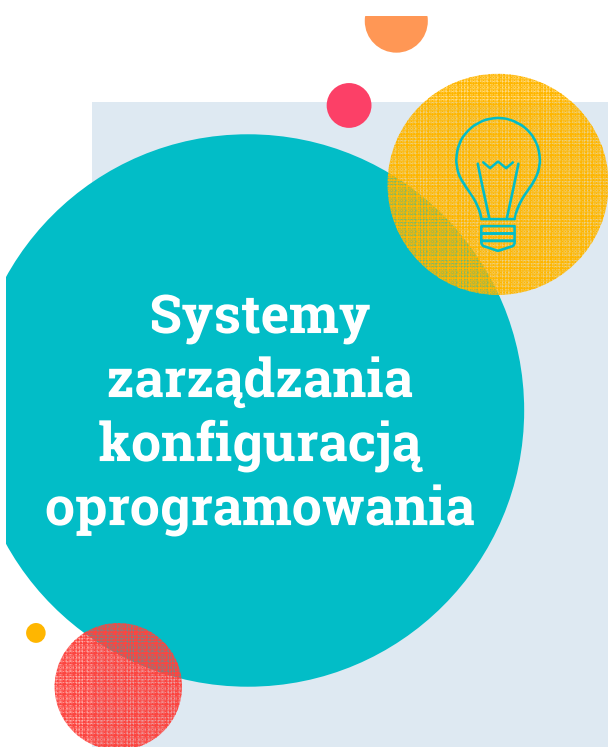
dr inż. Marcin Szelaąg  
Instytut Informatyki, PP



## Problemy zarządzania konfiguracją (SCM)

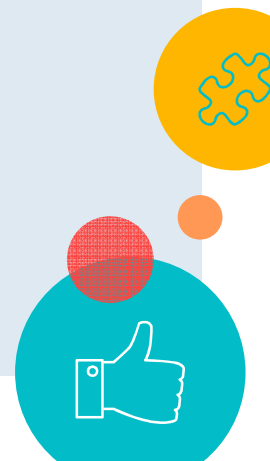
- **Mnogość artefaktów**, nad którymi trzeba zapanować
- **Równoczesna praca** (zmiany) wielu osób
- **Wiele wersji** poszczególnych artefaktów (kwestia identyfikacji wersji, wyznaczania różnic pomiędzy wersjami)
- Na daną wersję (konfigurację) produktu mogą składać się **różne wersje artefaktów**
- **Równoległe** poprawianie wcześniejszej wersji oprogramowania i praca nad nową wersją





## Systemy zarządzania konfiguracją oprogramowania

- W celu rozwiązania ww. problemów stosuje się **systemy zarządzania konfiguracją**
- Oprócz zastosowania systemu zarządzania konfiguracją konieczne jest również ustalenie **procedur** wprowadzania zmian (workflow), wydawania nowej wersji, poprawiania defektów, łączenia zmian z różnych wersji, itd.





# Git

- <http://git-scm.com>
- **Rozproszony SCM**
- Nastawiony na: **szybkość działania**, **integralność danych (SHA-1)**, **nieliniowy rozwój** oprogramowania („tanie” tworzenie gałęzi!), efektywne scalanie zmian
- 2005, **Linus Torvalds**
- **Pełne repozytorium** w każdym węźle sieci
- **#everything-is-local**
- **Śledzenie zawartości** (snapshot), a nie pojedynczych plików!

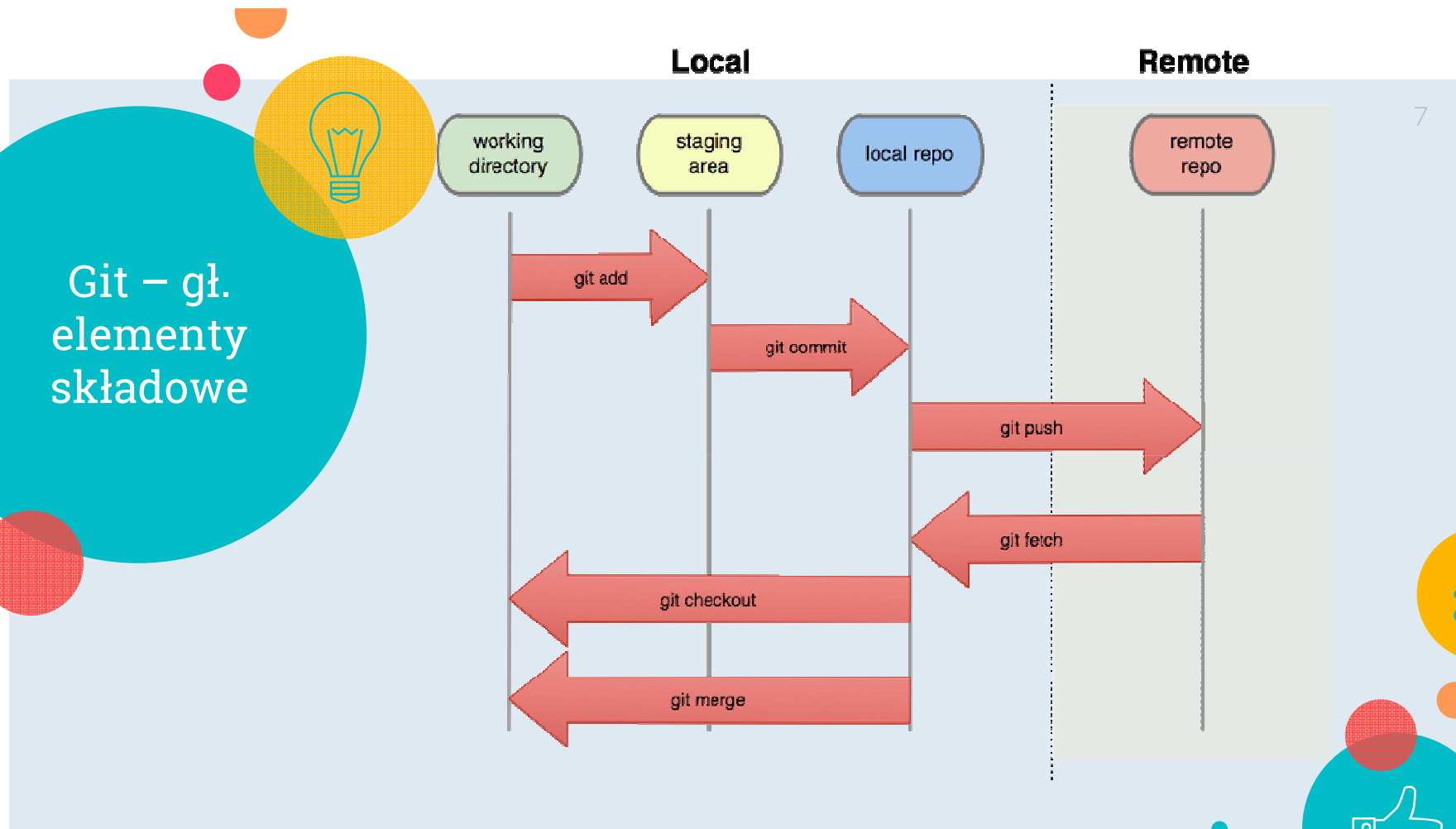
## Referencje

- **ProGit book:** <https://git-scm.com/book/en/v2>
- **About Git:** <https://git-scm.com/about>
- **Git reference:** <https://git-scm.com/docs/>
- **Git user manual:** <https://git-scm.com/docs/user-manual.html>
- **Resources to learn Git:** <https://try.github.io/>
- **Visualizing Git:** <http://git-school.github.io/visualizing-git/>
- **Git cheatsheet:** <http://ndpsoftware.com/git-cheatsheet.html>

## Referencje

- **Visual Git reference:**  
<http://marklodato.github.io/visual-git-guide/index-en.html>
- **Visualizing Git Concepts:**  
<http://onlywei.github.io/explain-git-with-d3/>
- **GitHub Git Cheat Sheet:**  
<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>
- **Escape a git mess, step-by-step:**  
<http://justinhileman.info/article/git-pretty/git-pretty.png>
- **Understanding Git – Index:**  
<https://medium.com/hackernoon/understanding-git-index-4821a0765cf>

# Git – gł. elementy składowe



Źródło: <https://greenido.files.wordpress.com/2013/07/git-local-remote.png?w=696&h=570>

## Git – gł. elementy składowe

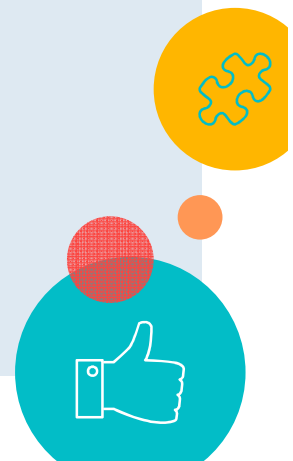
- **Przestrzeń robocza** (workspace) – zawiera pliki i katalogi projektu
- **Repozytorium lokalne** – baza obiektów (.git/objects) + referencje (.git/refs)
- **Indeks** (staging area) (.git/index) – „podgląd” następnego commita w repozytorium lokalnym ([wizualizacja](#))
- **Repozytorium zdalne** (remote/upstream repository) – repozytorium, z którym synchronizowane są zmiany w repozytorium lokalnym (np. na serwerze w sieci, na dysku sieciowym, w chmurze); **może być więcej niż jedno!**





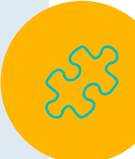

## git init

- Repozytorium typu **bare** i **non-bare**
- **git init --bare my-project.git**
- Repozytorium typu bare („czyste”, czyli bez przestrzeni roboczej) :
  - może być używane jako zdalne (współdzielone) repozytorium
  - umożliwia wgrywanie danych poleceniem `git push`
  - nie umożliwia dokonywania zmian poleceniem `git commit`





## git clone

- Dostępne **protokoły transmisji danych** (<https://git-scm.com/book/pl/v2/Git-on-the-Server-The-Protocols>):
    - **Local protocol** (`git clone <path>/proj.git`)
    - **Smart HTTP** / Dumb HTTP (`git clone https://example.com/project.git`)
    - **SSH protocol** (`git clone [ssh://]user@server/project.git`)
    - **Git protocol** (`git clone git://example.com/project.git`)
- 
- 

## Struktura katalogu .git

- Struktura katalogu **.git**:
  - hooks
  - info
  - logs
  - **objects** – baz obiektów
  - **refs** (podkatalogi: heads, remotes, tags)
  - config – aktualna konfiguracja projektu
  - **HEAD** – referencja symboliczna do aktualnej gałęzi lub nazwa commita (w stanie detached HEAD)
  - **index** – indeks

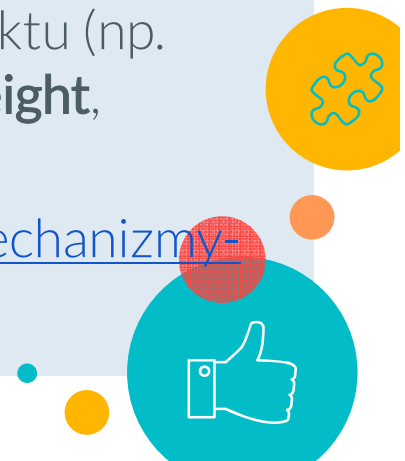
## Baza obiektów Git (.git/objects)

- Każdy obiekt identyfikowany jest poprzez nazwę – **40-znakowy ciąg cyfr heksadecymalnych** (160 bitów), np.  
72bd545a5ad58c71ac5da4ae0cba26142cec1dab
- Nazwa ta jest generowana przez **funkcję skrótu SHA-1**, na podstawie **zawartości obiektu** => nazwa obiektu jest w 100% zdeterminowana jego zawartością
- **Założenie:** jeżeli 2 obiekty mają tę samą nazwę, to mają identyczną zawartość



## Rodzaje obiektów Git

13

- Git definiuje **4 rodzaje obiektów**:
    - **blob** - odpowiada plikowi na dysku
    - **tree** – odpowiada katalogowi
    - **commit** – stan (snapshot) całego projektu
    - **tag** – referencja do innego obiektu (np. commita); dwa rodzaje: **lightweight**, **annotated**
  - <https://git-scm.com/book/pl/v1/Mechanizmy-wewnetrzne-w-Git-Obiekty-Gita>
- 



## Rodzaje obiektów Git - blob

- **Blob** – zawartość wersji pliku: 14
    - `$ echo 1234 > log.txt`
    - `$ git hash-object -w log.txt`  
**0bfea84767eb50e91f8658dab75471a0f78b84bf**
    - `$ git cat-file -p`  
0bfea84767eb50e91f8658dab75471a0f78b84bf  
**1234**
    - powstał plik:  
`.git/objects/0b/fea84767eb50e91f8658dab75471a0f78b84bf`
- 
- 

## Rodzaje obiektów Git - blob



**Schemat tworzenia obiektu blob** dla pliku o zawartości "what is up, doc?" (długość 16 znaków):

- `content = "what is up, doc?"`
- `header = "blob #{content.length}\0"`
- `store = header + content`
- `digest = sha1(store)`
- `compressed_store=zlib(store)`
- `path= ".git/objects/" + digest[0,2] + '/' + digest[2,38]`
- `file.open(path, 'w')`
- `file.write(compressed_store)`



## Rodzaje obiektów Git - tree

◦ **Tree** – odpowiednik katalogu:

- grupa plików i podkatalogów
- lista obiektów typu blob i tree, np.:

```
$ git cat-file -p  
8d36a8a827a1d5d09f661aca627ed64d38c4d41f  
100644 blob 877b08...b8fe43 README  
100644 blob cccdd3...2fa3e6 gradlew  
040000 tree 7787a0...88f861 src
```

(wielokropek został użyty wyłącznie na potrzeby prezentacji na slajdzie)



## Rodzaje obiektów Git - commit

**Commit** – snapshot zawartości gł. katalogu projektu:17

```
$ git cat-file -p develop
tree 9e00b0...399cc
parent 74ae9e...cb9112
parent 134381...67eebe (gdymerge commit)
author Marcin Szelag
<mszelag@cs.put.poznan.pl> 1551448514
+0100
committer Marcin Szelag
<mszelag@cs.put.poznan.pl> 1551448514
+0100
Added new Junit tests.
```

## Rodzaje obiektów Git - tag

**Lightweight tag** – stały wskaźnik na commit (w przeciwieństwie do brancha). Po wykonaniu polecenia

18

- `git tag v1.0 [commit]`
- powstanie plik `.git/refs/tags/v1.0` zawierający nazwę (SHA-1) wskazywanego commita

## Rodzaje obiektów Git - tag

**Annotated tag** – obiekt wskazujący stale na inny obiekt (najczęściej commit); posiada następujące właściwości:

- object (SHA-1 wskazywanego obiektu)
- type (typ wskazywanego obiektu)
- tag
- tagger, date (kto i kiedy stworzył taga)
- message (opis taga)
- Po wykonaniu polecenia
  - `git tag -a v1.0 [commit] -m "My tag"`
- powstanie plik `.git\refs\tags\v1.0` zawierający nazwę (SHA-1) stworzonego obiektu typu tag

## Git – śledzenie plików

- **Pliki śledzone** (tracked) – wersja pliku o tej nazwie istnieje już w bazie obiektów Gita
- **Pliki nieśledzone** (untracked)
- **git add** – aktualizacja indeksu (możliwa wielokrotnie przed commitem) – dla nowego lub zmodyfikowanego pliku
- **git status** – wyświetlenia statusu (aktualna gałąź, śledzone nowe/zmienione pliki, nieśledzone pliki)

# Git – śledzenie plików

Administrator: C:\Windows\System32\cmd.exe

```
c:\Temp\ruleLearn>git status
On branch develop
Your branch is up to date with 'origin/develop'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README
        deleted:    gradlew

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   gradlew.bat
        deleted:    settings.gradle

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

        log.txt
        test2.txt


c:\Temp\ruleLearn>
```

21





## git add

- **git add** <pathspec> - dodanie pliku(ów) do indeksu
  - gł. przełączniki:
    - **-u** (--update) – aktualizacja indeksu dla **zmienionych** i **usuniętych** (śledzonych) plików pasujących do pathspec (np. \*.c)
    - **-A** (--all) – **j.w.** + **dodanie nieśledzonych** plików pasujących do pathspec (pod warunkiem, że nie są ignorowane)
- 



## git commit

23

- **git commit** – stworzenie commita na podstawie stanu indeksu
- gł. przełączniki:
  - **-a** (`--all`) – automatyczne dodanie do indeksu plików **zmienionych i usuniętych** (śledzonych)
  - **-m "Message"** – dodanie opisu commita
- `.git/COMMIT_EDITMSG` – pamięta message ostatniego commita

## Git - gałęzie

- gałąź (branch) – automat. przesuwany **wskaźnik na commit**
- **git/refs/heads/master** – zawiera SHA-1 najnowszego commita na branchu master (head, tip)
- **.git/HEAD** – zawiera ref. symboliczną, np.:  
ref: refs/heads/master
- **git branch** -> lista lokalnych gałęzi
- **git branch -r** -> lista zdalnych gałęzi
- **git branch branch-name** -> stworzenie gałęzi
- **git checkout branch-name** -> przełączenie gałęzi
- **git checkout -b new-branch** -> stworzenie nowej gałęzi i przełączenie się na nią
- **git branch -d branch-name** -> usunięcie gałęzi



## Git - gałęzie

### o Rodzaje gałęzi:

25

- o **remote branch** (gałąź lokalna w zdalnym repo)
- o **remote-tracking branch** (np. origin/master) – tylko do odczytu (snapshot gałęzi zdalnej)
- o **local branch** (np. feature/experimental)
- o **local tracking branch** (np. master) – gałąź lokalna, która jest synchronizowana z gałęzią zdalną; synchronizacja ustawiana automatycznie gdy checkout gałęzi zdalnej lub przez polecenie `git branch`, np.

```
git branch --set-upstream-  
to=upstream/foo [foo]
```



## Git – interakcja ze zdalnym repozytorium

- `git push` (gdy ustawiony upstream branch)
- `git push <upstream> <branch>`
- `git push <upstream> <tag>`
- `git push <upstream> --tags`
- `git pull` (gdy ustawiony upstream branch)
- `git pull <upstream> <branch>`
- `git fetch <upstream>`
  
- `git pull = git fetch + git merge`

26

## Git – łączenie zmian z różnych gałęzi

### Łączenie gałęzi:

27

- `git checkout develop`
- **git merge** [--no-ff] feature/gui [-m "Message"] – powstaje tzw. **merge commit** (o dwóch rodzicach); w razie konfliktu – konieczne ręczne scalenie + ręczny commit (indeks pamięta do 3 wersji każdego skonfliktowanego pliku – dla wspólnego przodka, HEAD i MERGE\_HEAD)
- `git merge --abort` – wycofanie się (gdy konflikty)
- **git rebase, git rebase -i** (<https://git-scm.com/docs/git-rebase>) – zastosowanie commit'ów na innej gałęzi

## Git – wycofywanie zmian

- **git reset** <tryb><commit> – zmiana wskazania gałęzi; tryby:
  - `--soft` – nie zmienia katalogu roboczego ani indeksu (tylko ustawia head na commit)
  - `--mixed` (domyślna opcja) – reset indeksu do stanu z commita
  - `--hard` – reset indeksu i katalogu roboczego do stanu z commita
- **git reset -- <file>** - usunięcie nowej wersji pliku z indeksu (przywrócenie tam wersji pliku `file` z lokalnego repozytorium); bez zmiany pliku w katalogu roboczym

28



## Git – wycofywanie zmian

- **git revert** – wycofanie zmian z commita(ów),<sup>29</sup> poprzez stworzenie nowego commita wycofującego zmiany
- **git checkout** <branch> – przełącza gałąź, aktualizuje katalog roboczy i indeks
- **git checkout -b** <branch>
- **git checkout** <commit> – stan DETACHED HEAD
- **git checkout <commit> -- <file>** - pobranie 1 pliku z commita (odrzuca niezapisane zmiany w pliku!)

(<https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting>)

## Git – schowek

- `git stash [push] [-m= "message"]`
  - `git stash pop [--index]`
  - `git stash apply [stash@{rev}] [--index]`
  - Schowek ma postać **stosu niepowiązanych commitów**:  
**stash@{0}** - ostatnio dod. el., **stash@{1}** – wcześ. el.
  - `git stash list, git stash show`
  - `git stash drop [stash@{revision}],  
git stash clear`
  - `git stash branch <new-branchname>`
- ```
?> git stash push -m"Fixing config"
```
- Saved working directory and index state On develop:  
Pushing changes to stash
- Powstał plik **.git/refs/stash**

30

# Git – schowek

Lister - [c:\Temp\ruleLearn\.git\refs\stash]

Plik Edytuj Opcje Kodowanie Pomoc

b5efb72a6e16ead1b4e8b2ea5ae59f71b01d995e

Administrator: C:\Windows\System32\cmd.exe

```
c:\Temp\ruleLearn>git status
On branch develop
Your branch is up to date with 'origin/develop'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README
        deleted:    gradlew
        new file:   log.txt
        new file:   test2.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   gradlew.bat
        deleted:    settings.gradle

c:\Temp\ruleLearn>git stash push -m "Stash example"
Saved working directory and index state On develop: Stash example

c:\Temp\ruleLearn>git status
On branch develop
Your branch is up to date with 'origin/develop'.

nothing to commit, working tree clean

c:\Temp\ruleLearn>git stash list
stash@{0}: On develop: Stash example

c:\Temp\ruleLearn>git stash show
README          | 2 +-
gradlew         | 172 -----
gradlew.bat     | 2 +-
log.txt         | 1 +
settings.gradle | 26 -----
test2.txt       | 1 +
6 files changed, 4 insertions(+), 200 deletions(-)
```

# Git – schowek

Lister - [c:\Temp\ruleLearn\.git\refs\stash]

Plik [Edytuj](#) [Opcje](#) [Kodowanie](#) [Pomoc](#)  
**b5efb72a6e16ead1b4e8b2ea5ae59f71b01d995e**

```
Administrator: C:\Windows\System32\cmd.exe
c:\Temp\ruleLearn>git cat-file -t b5efb7
commit
c:\Temp\ruleLearn>git cat-file -p b5efb7
tree 97f441050600dc6e6c54bd36352a377fd8ee6932
parent 72bd545a5ad58c71ac5da4ae0cba26142cec1dab
parent 215db916740011b9764eaf6fe189f03660807806
author Marcin Szelag <marcin.szelag@cs.put.poznan.pl> 1552852687 +0100
committer Marcin Szelag <marcin.szelag@cs.put.poznan.pl> 1552852687 +0100

On develop: Stash example

c:\Temp\ruleLearn>git cat-file -p 97f4410
100644 blob 0bdb83f54a4115a9a731ba1dc1552bea95977c84 .gitattributes
100644 blob 2cfbe8e4b5a2868c43427d172bd49eeFa4365ead .gitignore
100644 blob b67376020173274ded7cc91126ac9f06f94290cb README
100644 blob 22457dfa3cfe8d62c7cc6794e763ec78c9c00344 build.gradle
100644 blob 0ead9c7dae4e5de4eeb1ae51de80414b6ff115f gradle.properties
040000 tree 41b00e4652f3d317e2c90f121e770916deaecd55 gradle
100644 blob f50f5172acc409ad764593bd827d1625203ce4dc gradlew.bat
100644 blob e56e15bb7ddb6bd0b6d924b18fcee53d8713d7ea log.txt
040000 tree 05044ddc57376b1b3ede409f0d84d7d580feeb70 src
100644 blob 0bfea84767eb50e91f8658dab75471a0f78b84bf test2.txt

c:\Temp\ruleLearn>git log -n 1
commit 72bd545a5ad58c71ac5da4ae0cba26142cec1dab (HEAD -> develop, tag: v2.0, tag: v1.0, origin/develop)
Author: Marcin Szelag <marcin.szelag@cs.put.poznan.pl>
Date: Fri Mar 1 14:55:14 2019 +0100

    Added tests for generating "at most" certain DRSA rules for windsor data set.

c:\Temp\ruleLearn>git cat-file -t 215db9
commit
c:\Temp\ruleLearn>git cat-file -p 215db9
tree 9f83b56885049bf2a847a272d953d53d442a988d
parent 72bd545a5ad58c71ac5da4ae0cba26142cec1dab
author Marcin Szelag <marcin.szelag@cs.put.poznan.pl> 1552852686 +0100
committer Marcin Szelag <marcin.szelag@cs.put.poznan.pl> 1552852686 +0100

index on develop: 72bd545 Added tests for generating "at most" certain DRSA rules for windsor data set

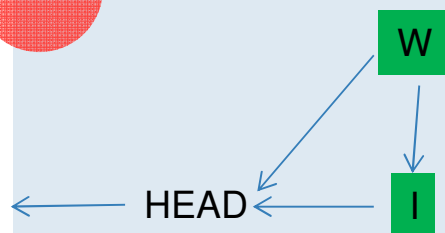
c:\Temp\ruleLearn>git cat-file -p 9f83b5
100644 blob 0bdb83f54a4115a9a731ba1dc1552bea95977c84 .gitattributes
100644 blob 2cfbe8e4b5a2868c43427d172bd49eeFa4365ead .gitignore
100644 blob b67376020173274ded7cc91126ac9f06f94290cb README
100644 blob 22457dfa3cfe8d62c7cc6794e763ec78c9c00344 build.gradle
100644 blob 0ead9c7dae4e5de4eeb1ae51de80414b6ff115f gradle.properties
040000 tree 41b00e4652f3d317e2c90f121e770916deaecd55 gradle
100644 blob f9553162f122c71b34635112e717c3e733b5b212 gradlew.bat
100644 blob e56e15bb7ddb6bd0b6d924b18fcee53d8713d7ea log.txt
100644 blob 6bf24221b1db1f7d5131f7b0d486bfd3a54e0b4 settings.gradle
040000 tree 05044ddc57376b1b3ede409f0d84d7d580feeb70 src
100644 blob 0bfea84767eb50e91f8658dab75471a0f78b84bf test2.txt
```

workspace  
w momencie  
git stash push  
(usunięte pliki  
gradlew i  
settings.gradle)

indeks  
w momencie  
git stash push  
(usunięty tylko  
plik gradlew)



## Git – schowek: wnioski



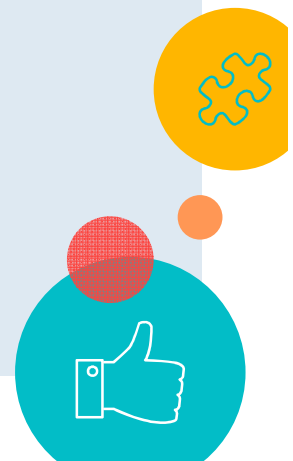
- Pozycja schowka odpowiada **parze commit'ów**
- Tree **commita W** zawiera wersje plików z katalogu roboczego
- Jego rodzic nr 1 to HEAD
- Jego rodzic nr 2 to commit I
- **Commit I:**
  - jego rodzic to HEAD
  - jego tree zawiera wersje plików z indeksu

<https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning>



## Git – schowek: wnioski

- Pytanie – jaki będzie efekt?  
**git checkout stash**



## Git – schowek: wnioski

- Pytanie – jaki będzie efekt?  
**git checkout stash**

```
c:\Temp\rulelearn>git checkout stash
Note: checking out 'stash'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
```

```
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
```

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 59aca2a WIP on develop: 72bd545 Added tests for generating "at most" certain DRSA rules for windsor data set.
```

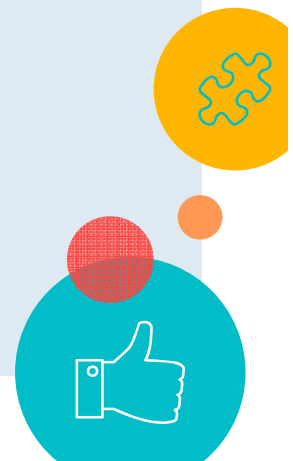
## Git – schowek: wnioski

- Pytanie – gdzie Git trzyma nazwy (SHA-1) commitów typu W?
  - `.git/refs/stash` – nazwa commita na górze stosu
  - `.git/logs/refs/stash` – lista wszystkich commitów na stosie (w schowku), w kolejności odwrotnej



## Git – konfiguracja

- **ignorowanie plików** obecnych w katalogu roboczym zarządzanym przez Git-a
- **.gitattributes**



## Git – ignorowanie plików

\* działa pod warunkiem, że  
nie jest ignorowany cały  
katalog zawierający plik  
pasujący do wzorca z  
negacją

- **.gitignore** - pliki umożliwiające wskazanie Gitowi ignorowanych plików (najwyższy priorytet)
- Wgrywane do repozytorium, współdzielone
- Ignorowanie nie działa dla plików już śledzonych
- **.git/info/exclude** (średni priorytet)
- `git config --global core.excludesFile ~/.gitignore` (najniższy priorytet)
- **!<wzorzec>** - nie ignorowanie pasujących plików\*

<https://git-scm.com/docs/gitignore>

<https://pl.atlassian.com/git/tutorials/saving-changes/gitignore> => **Git ignore patterns**

## Git – ignorowanie plików

- **git check-ignore -v <file>** - sprawdzenie jaki wzorzec pasuje do pliku file
- **git rm --cached filename** - Git przestaje śledzić plik, ale zostawia go w katalogu roboczym

```
?> git check-ignore -v tested-  
file.class  
.gitignore:8:/tested-file.class  
tested-file.class
```

## Git – .gitattributes (przykład)

# Set the default behavior, if `core.autocrlf` not set. 40

\* `text=auto`

# Explicitly declare text files you want to always be normalized and converted to native line endings on checkout.


\* `.java text`

# Declare files that will always have LF line endings on checkout.

\* `.sh text eol=lf`

<https://git-scm.com/docs/gitattributes>





## Git – inne ważne polecenia

- git config
- git log
- git diff
- git worktree
- git mergetool
- git cherry-pick  
(<https://stackoverflow.com/questions/9339429/what-does-cherry-picking-a-commit-with-git-mean>)
- git reflog  
(<https://pl.atlassian.com/git/tutorials/rewriting-history/git-reflog>)



## Git - gui

- **gitk** – wbudowana przeglądarka repozytoriów
- **git-gui** – wbudowany graficzny interfejs Git-a
- **TortoiseGit** – GUI dla Windows Shell
- **SourceTree** – GUI dla Windows i Mac
- **github, gitlab, bitbucket, ...** – serwisy internetowe

## Git – command-line

- **Czy znajomość poleceń jest przydatna?**
- Git plumbing vs porcelain (<https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain>)

Dziękuję!  
Pytania?

