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, poprawiana 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!

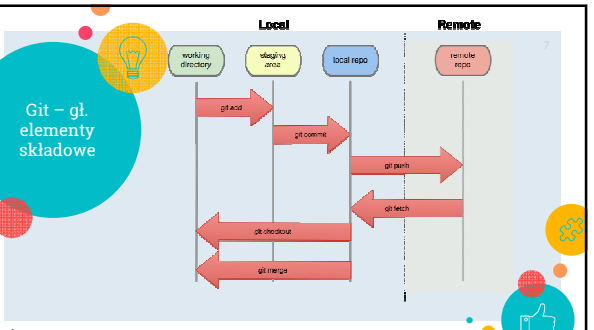
Referencje

- **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-wewnętrzne-w-Git-Obiekty-Gita

Rodzaje obiektów Git - blob

14

- **Blob** – zawartość wersji pliku:
  - `$ 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

15

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

16

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

17

Commit – snapshot zawartości gł. katalogu projektu:

```
$ git cat-file -p develop
tree 9e00b0…399cc
parent 74ae9e…cb9112
parent 134381…67eebe (gdy merge 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

18

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

- `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

20

- **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

21

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>
```

git add

22

- **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

25

- Rodzaje gałęzi:
  - **remote branch** (gałąź lokalna w zdalnym repo)
  - **remote-tracking branch** (np. origin/master) – tylko do odczytu (snapshot gałęzi zdalnej)
  - **local branch** (np. feature/experimental)
  - **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

26

- **`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`

Git – łączenie zmian z różnych gałęzi

27

- `Łączenie gałęzi:`
  - `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

28

- **`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

Git – wycofywanie zmian

29

- **`git revert`** – wycofanie zmian z commita(ów), 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

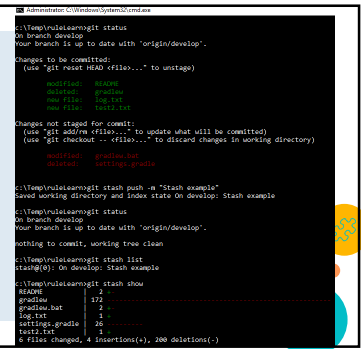
30

- **`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}`** – wcześn. el.
- **`git stash list, git stash show`**
- **`git stash drop`** `[stash@{revision}]`, **`git stash clear`**
- **`git stash branch <new-branchname>`**

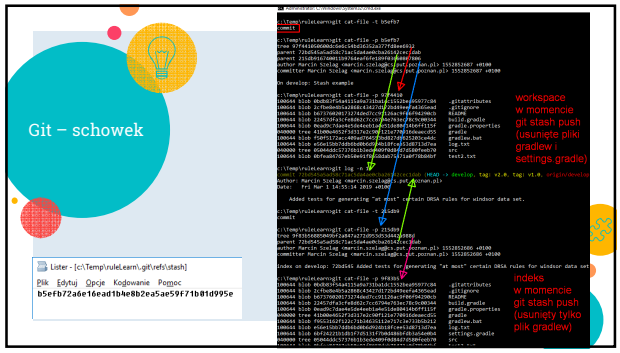
```
?>git stash push -m"Fixing config"
Saved working directory and index state On develop:
Pushing changes to stash
```

Powstał plik **`.git/refs/stash`**

Git – schowek



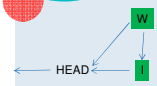
Git – schowek



Git – schowek: wnioski

33

- 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

34

- Pytanie – jaki będzie efekt?

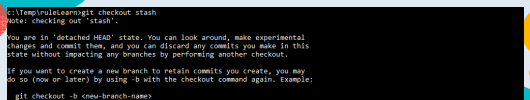
```
git checkout stash
```

Git – schowek: wnioski

35

- Pytanie – jaki będzie efekt?

```
git checkout stash
```



Git – schowek: wnioski

36

- 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

37

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

Git – ignorowanie plików

- `.gitignore` - pliki umożliwiające wskazanie Git-owi 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*

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

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

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

Git – ignorowanie plików

39

- `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)

40

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

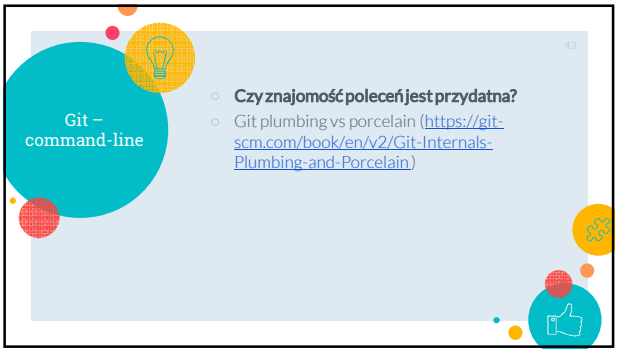
41

- 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

42

- **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
43
Czy znajomość poleceń jest przydatna?
Git plumbing vs porcelain (https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain)


44
Dziękuję!
Pytania?