Wykład 11.05.2016

Co zostanie wypisane na ekranie? (1)

```
class A {
  public:
    void wypisz(int n) {
         cout << "XXX: " << n << endl;
       }
};

class B : public A {
  public:
    void wypisz(double r) {
         cout << "YYY: " << r << endl;
       }
};
```

```
void testuj(A * obiekt_a) {
  obiekt_a->wypisz(5);
}

int main() {
  B obiekt_b;
  testuj(&obiekt_b);
}
```

Co zostanie wypisane na ekranie? (2)

```
class A {
  public:
    void wypisz(int n) {
         cout << "XXX: " << n << endl;
       }
};

class B : public A {
  public:
    void wypisz(int r) {
         cout << "YYY: " << r << endl;
       }
};
```

```
void testuj(A * obiekt_a) {
  obiekt_a->wypisz(5);
}

int main() {
  B obiekt_b;
  testuj(&obiekt_b);
}
```

Co zostanie wypisane na ekranie? (3)

```
class A {
  public:
       virtual void wypisz(int n) {
         cout << "XXX: " << n << endl;
       }
};

class B : public A {
  public:
    void wypisz(int r) {
         cout << "YYY: " << r << endl;
       }
};
```

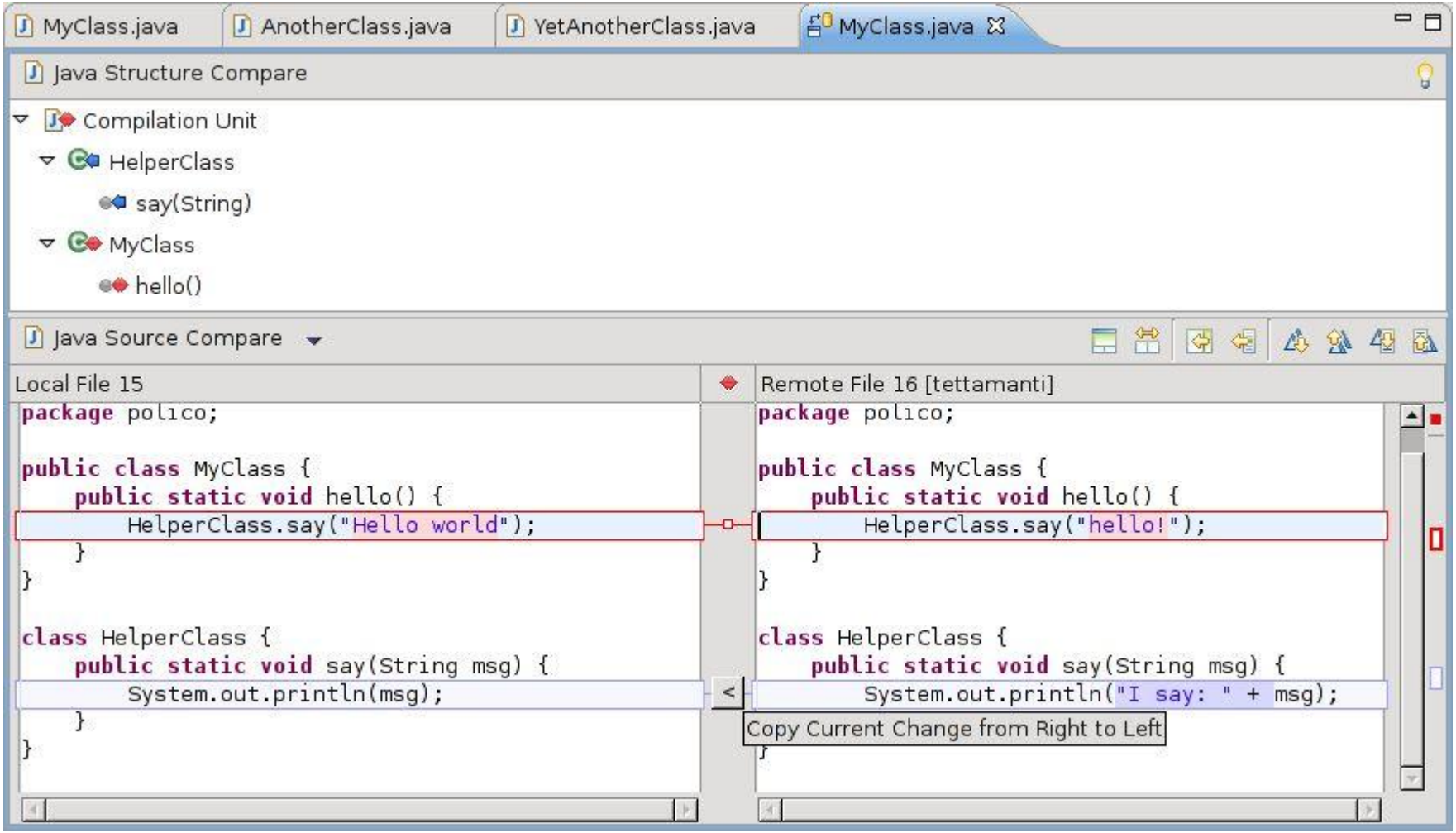
```
void testuj(A * obiekt_a) {
  obiekt_a->wypisz(5);
}

int main() {
  B obiekt_b;
  testuj(&obiekt_b);
}
```

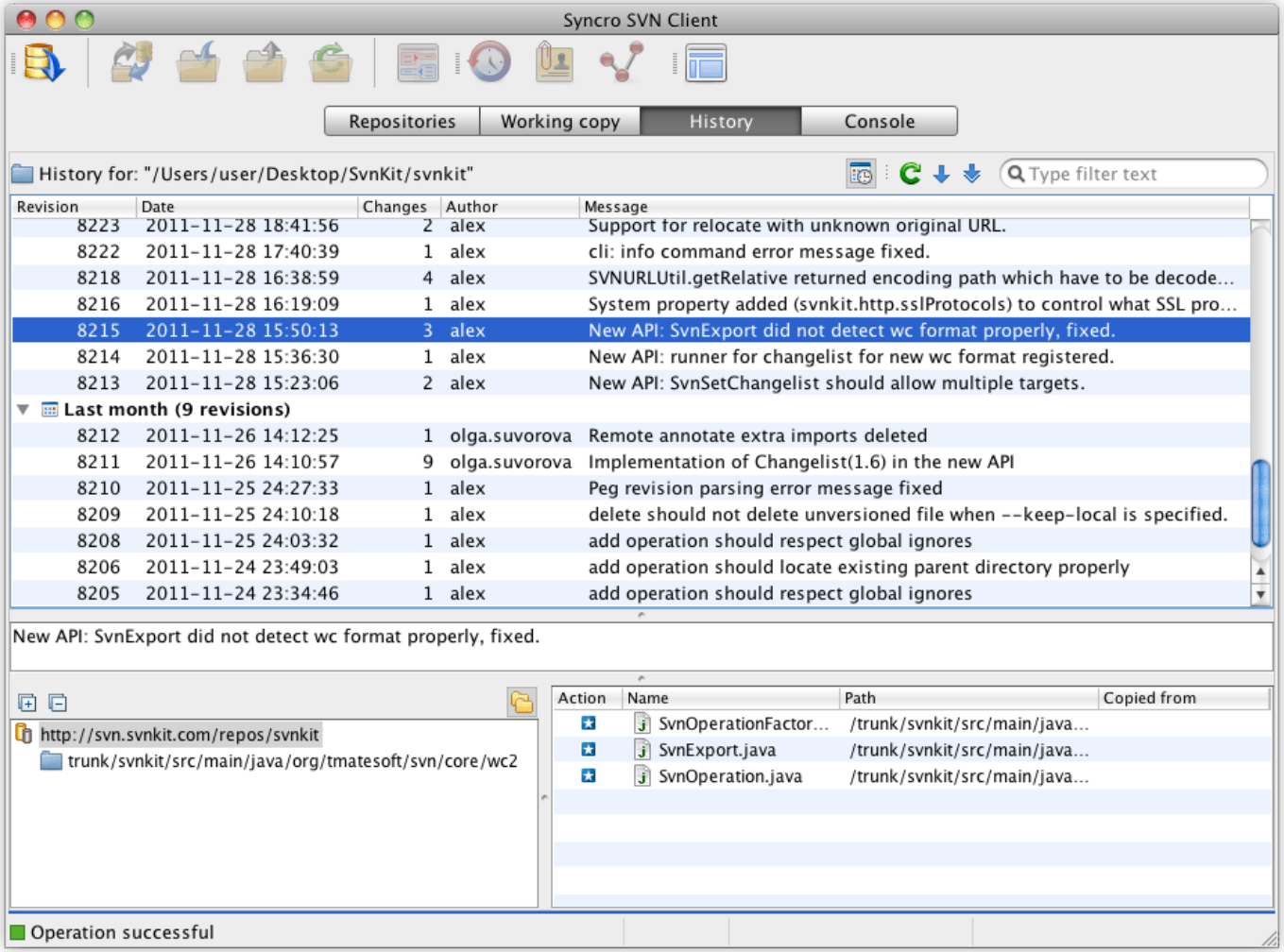
Czym są systemy kontroli wersji

- Praca w zespole
- Możliwość powrotu do dowolnego miejsca w historii (rewizja)
- Możliwość równoległego rozwoju dwóch gałęzi aplikacji
- Łatwość wersjonowania aplikacji

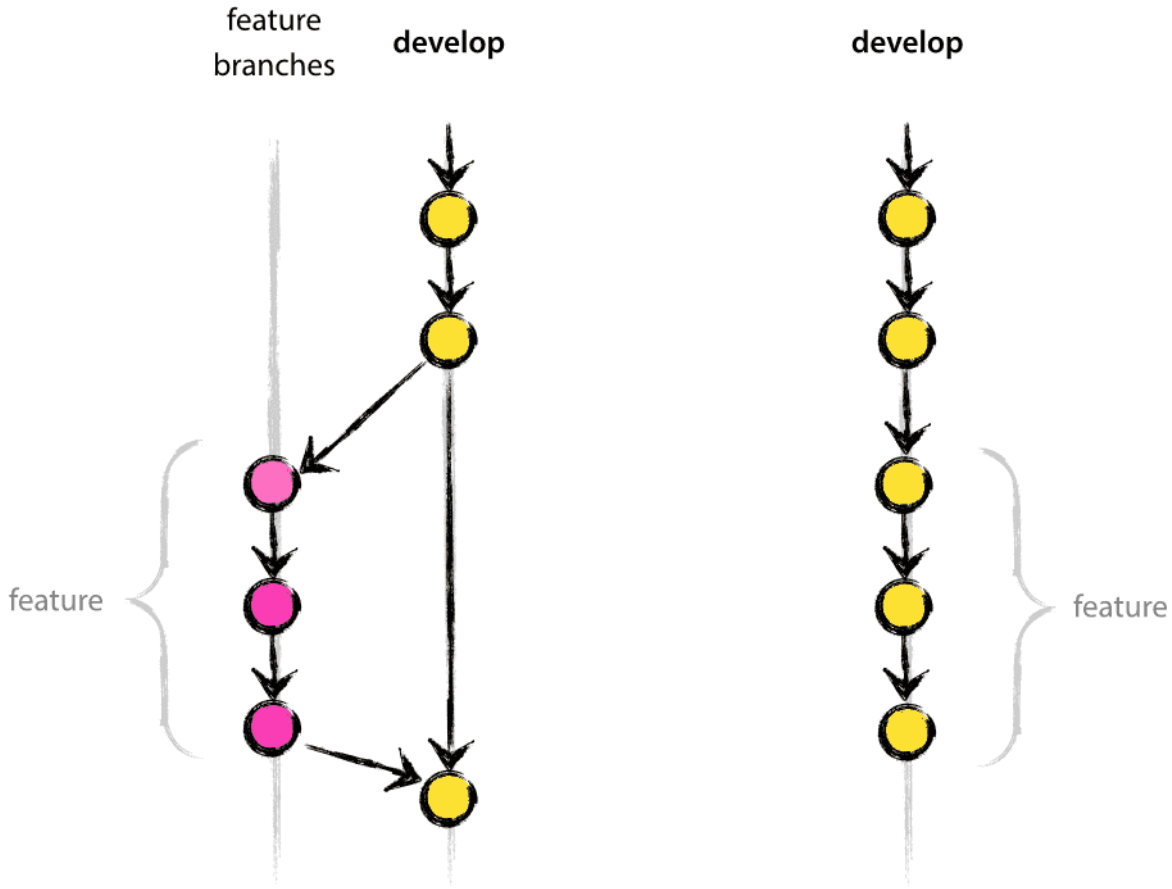
Różnice


MyClass.java
AnotherClass.java
YetAnotherClass.java
MyClass.java
Java Structure Compare
Compilation Unit
HelperClass
say(String)
MyClass
hello()
Java Source Compare
Local File 15
Remote File 16 [tettamanti]
package polico;
public class MyClass {
public static void hello() {
HelperClass.say("Hello world");
}
}
class HelperClass {
public static void say(String msg) {
System.out.println(msg);
}
}
package polico;
public class MyClass {
public static void hello() {
HelperClass.say("hello!");
}
}
class HelperClass {
public static void say(String msg) {
System.out.println("I say: " + msg);
}
}
<
Copy Current Change from Right to Left

Historia

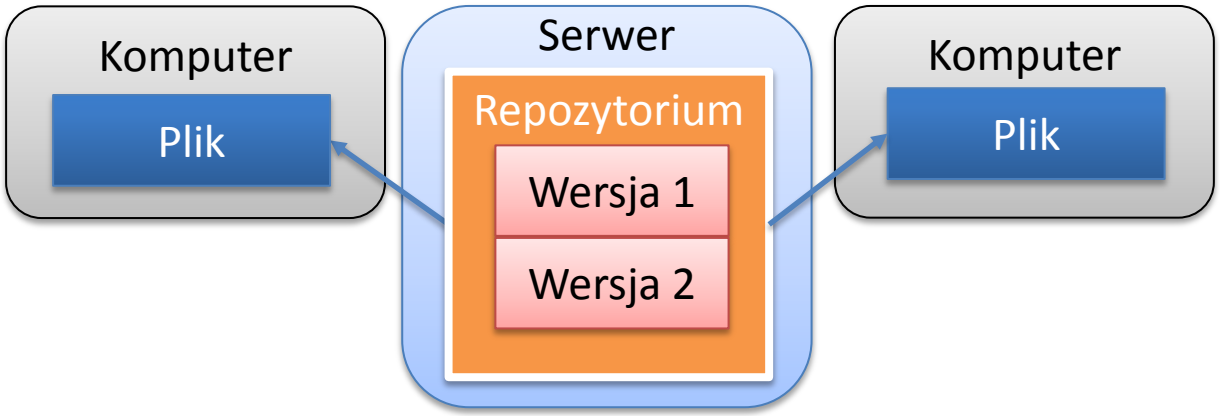

Syncro SVN Client
Repositories
Working copy
History
Console
History for: "/Users/user/Desktop/SvnKit/svnkit"
Type filter text
Revision
Date
Changes
Author
Message
8223 2011-11-28 18:41:56 2 alex Support for relocate with unknown original URL.
8222 2011-11-28 17:40:39 1 alex cli: info command error message fixed.
8218 2011-11-28 16:38:59 4 alex SVNURLUtil.getRelative returned encoding path which have to be decode...
8216 2011-11-28 16:19:09 1 alex System property added (svnkit.http.sslProtocols) to control what SSL pro...
8215 2011-11-28 15:50:13 3 alex New API: SvnExport did not detect wc format properly, fixed.
8214 2011-11-28 15:36:30 1 alex New API: runner for changelist for new wc format registered.
8213 2011-11-28 15:23:06 2 alex New API: SvnSetChangelist should allow multiple targets.
Last month (9 revisions)
8212 2011-11-26 14:12:25 1 olga.suvorova Remote annotate extra imports deleted
8211 2011-11-26 14:10:57 9 olga.suvorova Implementation of Changelist(1.6) in the new API
8210 2011-11-25 24:27:33 1 alex Peg revision parsing error message fixed
8209 2011-11-25 24:10:18 1 alex delete should not delete unversioned file when --keep-local is specified.
8208 2011-11-25 24:03:32 1 alex add operation should respect global ignores
8206 2011-11-24 23:49:03 1 alex add operation should locate existing parent directory properly
8205 2011-11-24 23:34:46 1 alex add operation should respect global ignores
New API: SvnExport did not detect wc format properly, fixed.
http://svn.svnkit.com/repos/svnkit
trunk/svnkit/src/main/java/org/tmatesoft/svn/core/wc2
Action
Name
Path
Copied from
SvnOperationFactor...
/trunk/svnkit/src/main/java...
SvnExport.java
/trunk/svnkit/src/main/java...
SvnOperation.java
/trunk/svnkit/src/main/java...
Operation successful

Równoległy rozwój



Modele pracy

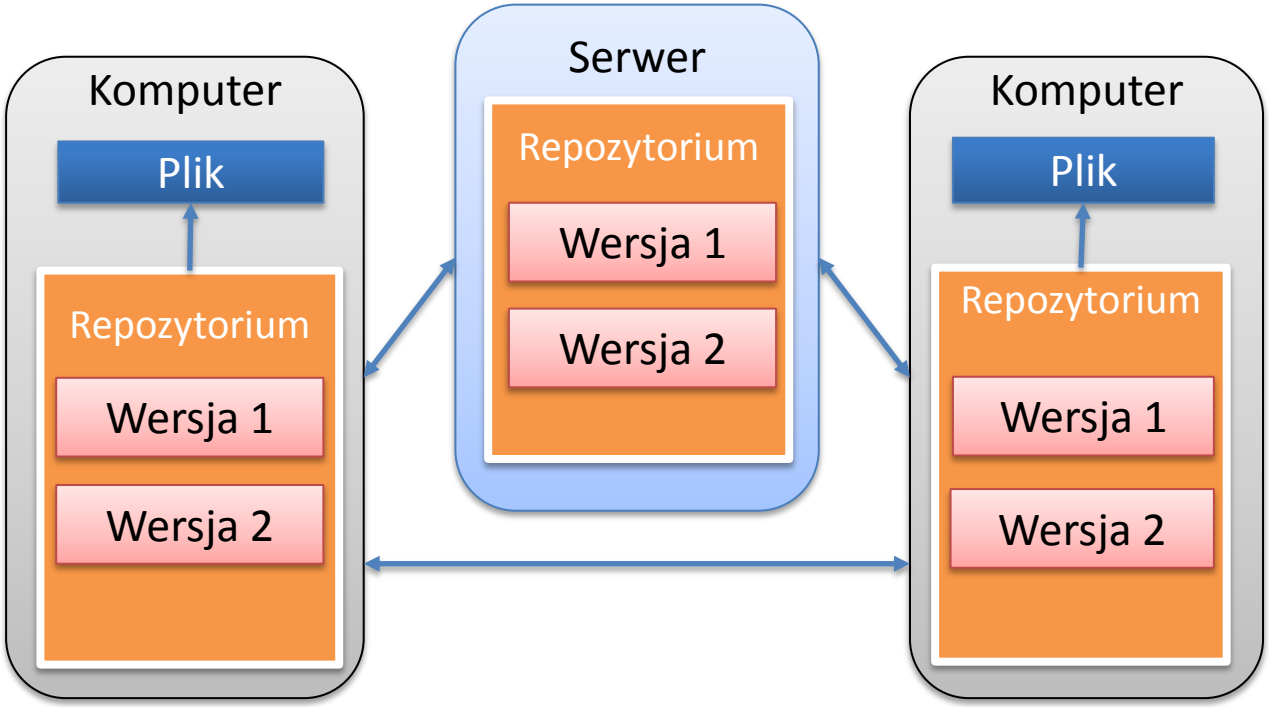
- Model Scentralizowany
  - SVN
  - CVS
  - Perforce



- Używana jako miejsce przechowywania kopii zapasowej (Backupu)
- Brak prywatnych branch'y
- Do wykonania większości standardowych operacji wymaga połączenia do serwera.

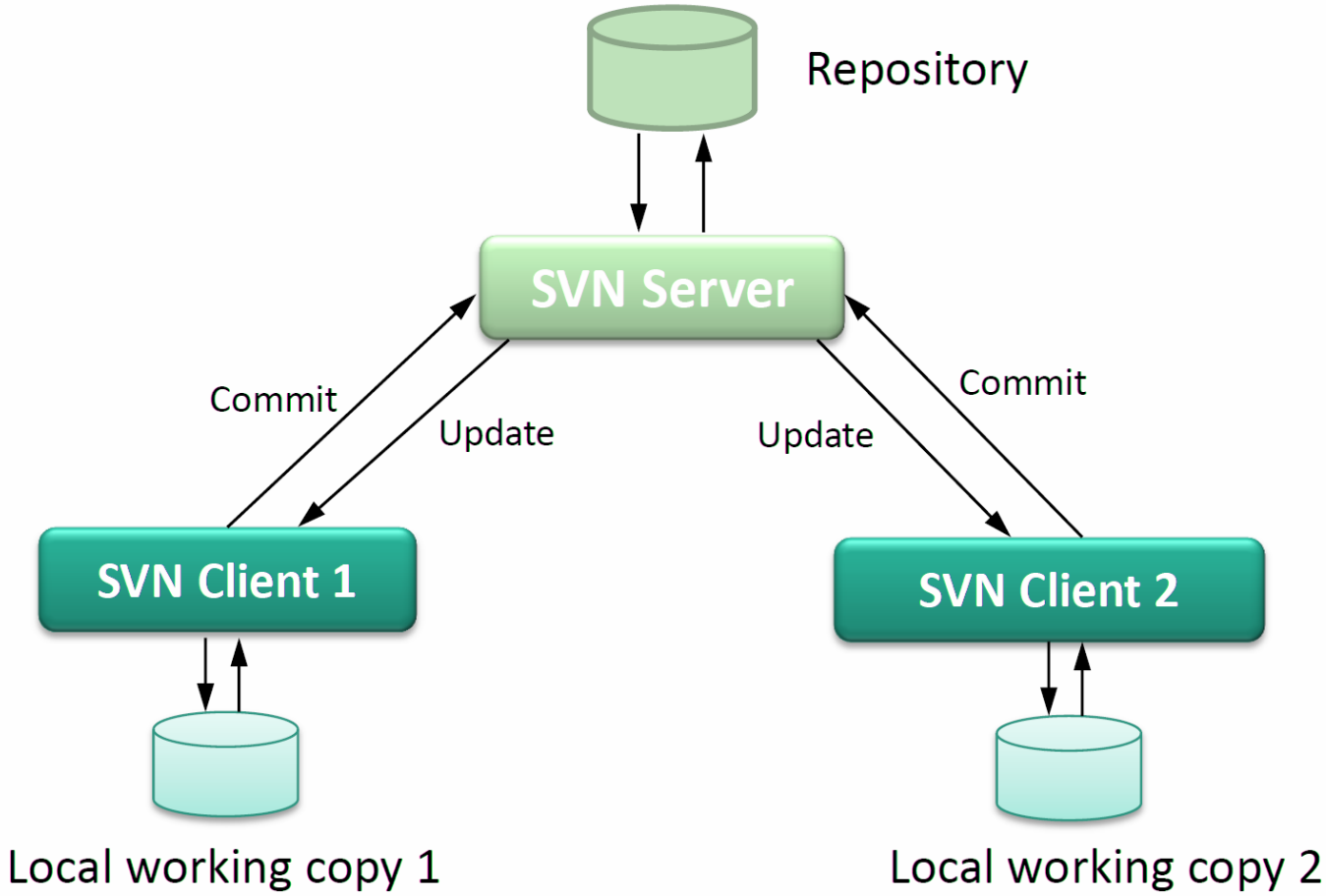
Modele pracy c.d.

- Model Rozproszony
  - GIT
  - Mercurial
  - Bazaar



- Możliwość pracy na wielu zdalnych repozytoriach.
- Wsparcie dla lokalnych/prywatnych branch'y.
- Nie potrzebuje dostępu do serwera dla większości operacji
- Pełna historia zmian dostępna lokalnie.

Subversion


Repository
SVN Server
Commit
Update
Update
Commit
SVN Client 1
SVN Client 2
Local working copy 1
Local working copy 2

Subversion

- Scentralizowany
- Prosty w obsłudze
- Śledzi historie zmian pliku
- Open-Source
- Operacje są atomowe (kończą się całkowicie albo w ogóle)
- Możliwość blokowania edytowanych plików

- Podstawowe komendy (na podstawie TortoiseSVN)
  - SVN Checkout – pozwala na pobranie kopii repozytorium z sieci
  - Create repository here – tworzy nowe repozytorium w danym miejscu
    - Create folder structure – tworzy od razu katalogi tags/branches/trunk
    - Start repobrowser – pozwala przeglądać aktualne repozytorium
- SVN Commit – Wrzucenie zmian na serwer
- SVN Update – Pobranie zmian z serwera
- Revert – Przywraca obecny stan lokalnej kopii do stanu z serwera
- Merge – Otwiera edytor który umożliwia łączenie plików (konflikty)
- Add – Dodaje plik do wersjonowania
- Resolve – Pokazuje pliki w konflikcie i próbuje sam je rozwiązać (zmergeować)
- Update to revision – pozwala powrócić do poprzedniej rewizji

Git

- Rozproszony system kontroli wersji
- Lokalne repozytoria
- Napisany przez Linusa Torvaldsa (Twórca Linuksa)
- Lokalne branche
- Bulk Commit
- Open – Source

Podstawowe komendy:

- Git create repository here – tworzy nowe repozytorium
    - Make it bare – Tworzy czyste repozytorium (Jak w systemach scentralizowanych)
- Clone - Pobranie lokalnej kopii repozytorium
- Pull – Zaciągnięcie zmian w repo (razem z merge)
- Commit – Wrzucenie zmian do lokalnego repo
- Push – Wypchnięcie zmian do repo zdalnego
- Resolve – Proponuje jedno z kilku rozwiązań merge-owania zmian
- Revert – Przywraca obecną wersję plików (z repo)
- Repo browser – Pozwala przeglądać repozytorium
- Diff – Pokazuje różnice między konkretnymi wersjami plików
- Merge – Pozwala ręcznie rozwiązać konflikty
- Add – Dodaje nowy plik do wersjonowania

Kilka słów o terminologii

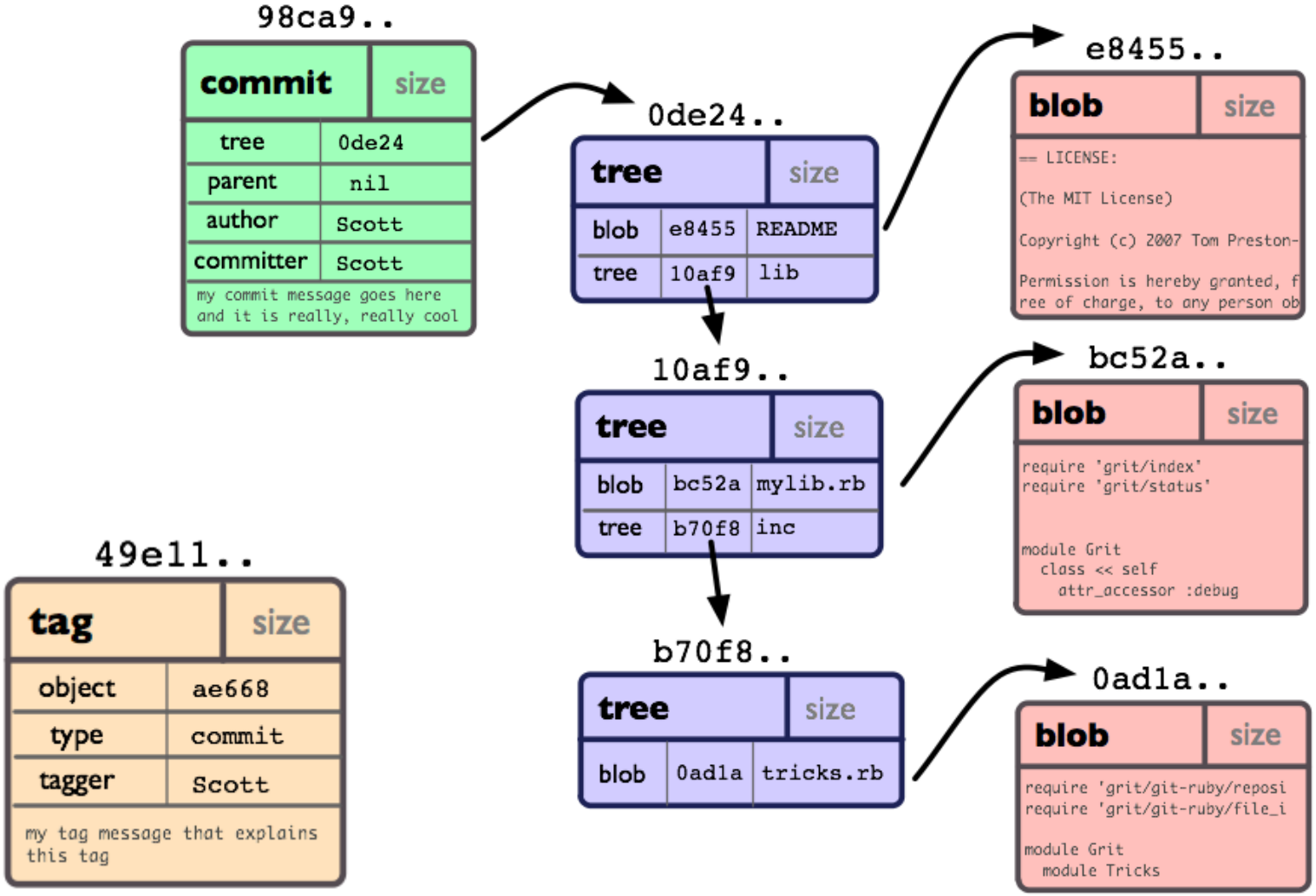
- Branch - równoległa gałąź projektu rozwijana oddzielnie od głównej.
- Tag – marker konkretnej wersji (rewizja w SVN'ie) projektu.

- Working Dir – katalog roboczy na którym pracujemy
- Index – rodzaj „cache", czyli miejsca gdzie trzymane są zmiany do commita
- Master Branch – główny branch z którym łączymy (merge) nasze zmiany przed wysłaniem do zdalnego repozytorium.

Obiekty GIT’a

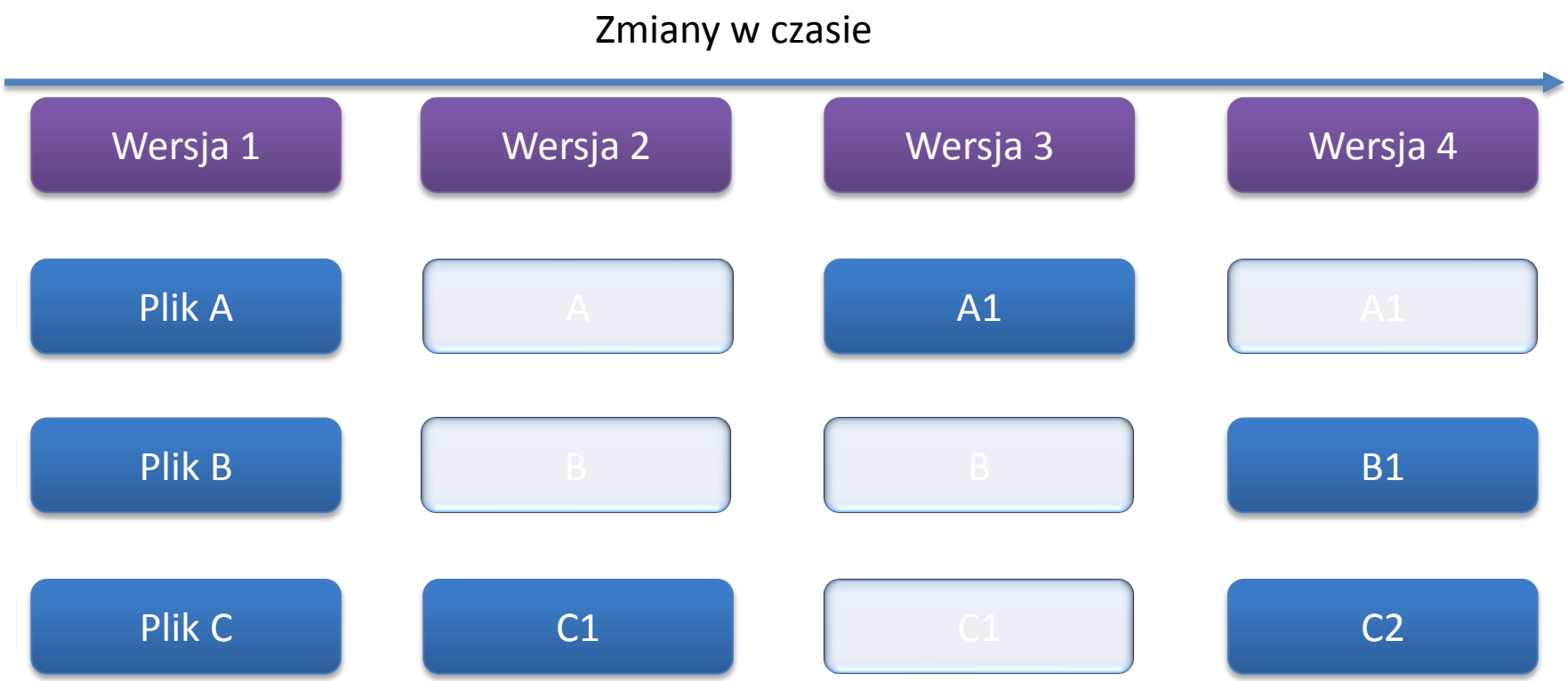
- Commit – wskazuje na tree oraz ojca, zawiera przykładowo takie informacje jak autor, data i treść wiadomości.
- Tree – reprezentuje stan pojedynczego katalogu (lista obiektów blob oraz zagnieżdżonych obiektów tree)
- Blob – zawiera zawartość pliku bez żadnej dodatkowej struktury
- Tag – wskazuje na konkretny commit oraz zawiera opis taga.

Obiekty GIT’a cd.



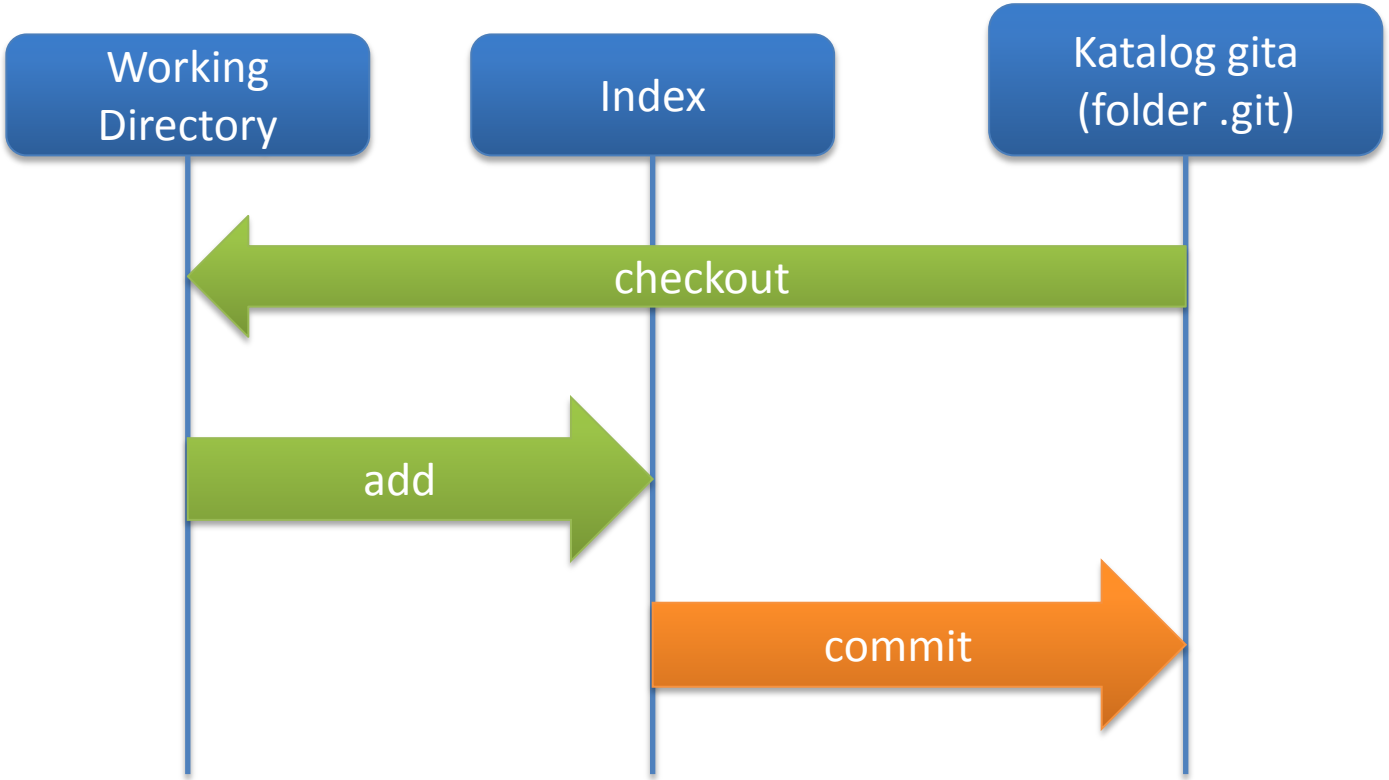
Struktura zmian w GIT'ie

1. Przechowywanie zawartości projektu jako „snapshot'ów".
2. Kompresowanie zawartości projektu.



Schemat prac w GIT'ie

- Większość operacji wykonywanych jest na lokalnym repozytorium.



Podstawowe operacje

- git init – stworzenie nowego repozytorium
- git add – dodanie zawartości pliku do Index'u
- git rm – usuwa plik z indexu
  (plik zniknie z working directory po commit'ie)
- git mv – przenosi plik
- git status – pokazuje status katalogu roboczego i poczekalni

- git config – pobiera i ustawia opcje globalne GIT'a lub tylko repozytorium

Operacje zdalne

- Przykładowymi operacjami zdalnymi są:

- git clone – pobiera zdalne repozytorium do podanego folderu
- git fetch – pobiera obiekty i wskaźniki z innego repozytorium
- git pull – pobiera i integruje obiekty i wskaźniki z innego repozytorium
- git push – aktualizuje zdalne repozytorium o wskaźniki i powiązane obiekty.

Podstawowe operacje cd.

- git commit – zapisuje zmiany do repozytorium lokalnego
- git log – wyświetla logi z commit'ów
- git show – wyświetla obiekt

- git fetch zdalnego – pobiera zmiany z repozytorium
- git pull – wywołuje polecenia fetch i merge
- git push repozytorium – wysyła zmiany do zdalnego

Podstawowe operacje cd.

- git branch – do zarządzania branch'ami
- git checkout – przełączanie się między branch'ami
- git merge – łączy podane branch'e
- git rebase – zmienia punkt startu dla branch'a
- git reset – przywraca stan katalogu roboczego
- git stash – zapisuje/odczytuje zmiany z przestrzeni

  tymczasowej (rodzaj schowka)
- git gc – porządkowanie i optymalizacja repozytorium

Prezentacja wykorzystania

- Stworzenie zdalnego repozytorium :

  > git init

- Dodanie nowych plików:

  > git add .
  > git add readme.txt

- Struktura projektu. (Plik .git/config)
  > git config

Prezentacja wykorzystania cd.

- Pierwszy commit:
- > git commit -m „Treść wiadomości”

- Historia :
- > git log

Prezentacja wykorzystania

- Stworzenie tag'a:
- > git tag v1.00
- > git tag v2.00 -m „Wersja druga”

- Wypisanie tagów:
- > git tag

- Wysłanie do zdalnego repozytorium tagów:
- > git push -tags

- Usunięcie taga:
- > git tag -d v2.00

Prezentacja wykorzystania

- Tworzenie brancha:
- > git branch myBranch

- Wyświetlenie branchy:
- > git branch

- Przełączanie branchy:
- > git checkout myBranch

- Usuwanie brancha:
- > git branch –d myBranch

Prezentacja wykorzystania

- Łączenie (merge) branchy:

  > git merge branchName

- W przypadku konfliktów po poprawkach w łatwy sposób można kontynuować pracę:

  >git commit -a -m „Merge branchy”

Prezentacja wykorzystania

- Historia zmian:

  > git log

  > git log --after=22.06.2014.19:20

  > git log <nazwa_pliku>

- Wyszukiwanie odpowiedzialnej osoby:

  > git blame <nazwa_pliku>

  > git blame –L 12,3 <nazwa_pliku>