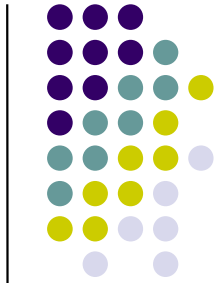


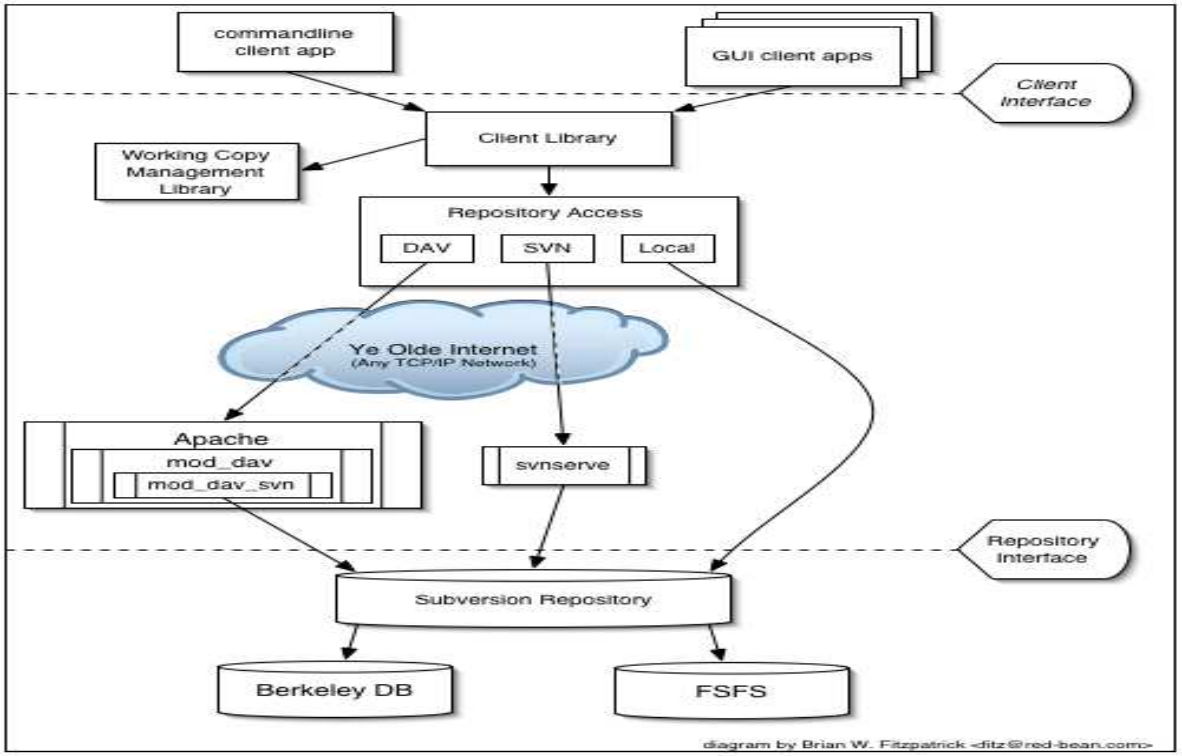
Алати за контролу верзија

Контрола верзија

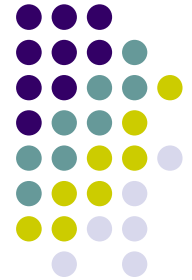
- Контрола верзија је назив за процес управљања дигиталним датотекама.
- Код развоја програмске подршке контрола верзија се користи не само за изворни код, већ и за документацију и конфигурационе датотеке.
- Алати за контролу верзија обично омогућавају следеће ствари:
  - Памћење свих верзија датотеке, до њеног настајања до садашњег тренутка
  - Поређење различитих верзија
  - Безбедну промену истих датотека од стране различитих корисника
  - Чување информација о томе ко је и зашто унео одређене измене
  - Одржавање паралелних верзија истих датотаке и њихово касније спајање (тзв. гранање)
  - Разне напредне упите и модификације кода (нпр. „Ко је последњи мењао ову линију кода?”, или „Измене које су настале између ове и ове верзије избаци из тренутне верзије”)
- Све наведене ствари алати за контролу верзија одрађују организовано, систематично и аутоматски.
- Данас се ниједан озбиљнији софтверски пројекат не може замислити без алата за контролу верзија.

Subversion (SVN)

- Subversion бесплатни програм, отвореног кода, за контролу верзија.
- Представља један од најпопуларнијих алата за ту сврху (разлог: пружа велики спектар могућности кроз једноставну употребу)
- SVN ради на мрежи, тако да омогућава удаљен приступ.
- Ради са свим врстама датотека (иако је фокус на разним варијантама текстуалних датотека: изворни код, конфугурационе датотеке, итд.)



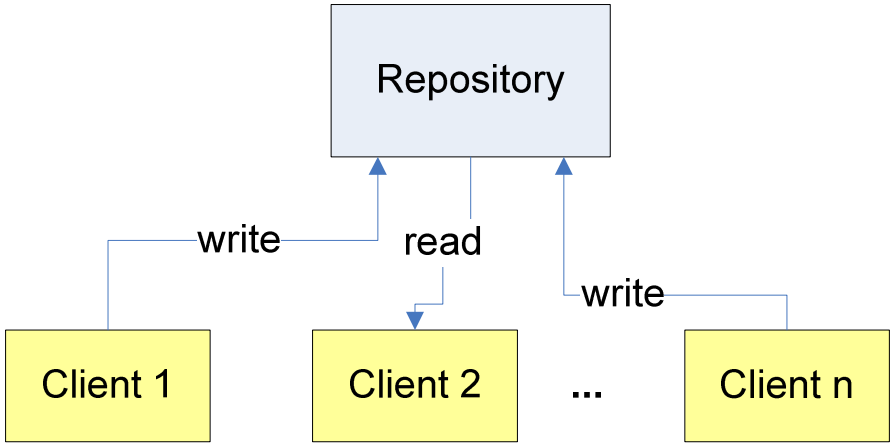
Основне карактериситке SVN-а



- Диференцијално памћење - Уместо памћења комплетних садржаја свих верзија памте се само разлике.
- Централизовани клијент-сервер систем - Подаци о верзијама се чувају на централизованом серверу, а клијент, обично са удаљене машине, може да им приступа. Садржај на серверу се назива **репозиторијум**.
- Дискретно, кориснички захтевано обављање измена - Измене се обављају само у дискретним тренуцима када корисник (клијент) то нареди. Процес измене репозиторијума назива се **комитовање**.
- Вишекориснички рад - Различити корисници могу мењати код у исто време. Механизми који обезбеђују одржавање конзистентности биће објашњени.
- Груписање измена у логичке целине - Уместо одвојеног праћења измена за сваку датотеку понаособ, SVN омогућава кориснику да групише измене различитих датотека и примени их одједном на репозиторијум. Свако комитовање на репозиторијум прави нову верзију комплетног репозиторијума. Верзије које се памте на репозиторијуму називају се и **ревизије**.
- Атомско комитовање целина - Измене различитих датотека које су груписане у целину могу се применити само одједном, недељиво.

Репозиторијум (складиште)

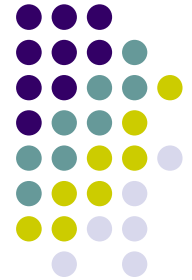
- SVN је централизован систем. У средишту је репозиторијум.
- **Репозиторијум**
  - Централно складиште података
  - Представља директоријуме и датотеке
  - Произвољан број клијената може му приступати
  - Све измене се чувају у репозиторијуму и свака верзија може бити реконструисана у било ком тренутку
  - Уједно, чувају се и информације о томе ко је и зашто обавио измену
  - Најсвежија ревизија на репозиторијуму назива се **глава** (HEAD)



Дељење датотека 1/3

- **Опис проблема - слично као проблем критичне секције:**
  - Клијент 1 и Клијент 2 узимају копију исте датотеке са репозиторијума
  - Оба клијента направе измене у датотеци
  - Клијент 1 постави измене на репозиторијум
  - Клијент 2 постави измене на репозиторијум
- Проблем је што постављање измена од стране Клијента 2 поништава измене које је унео Клијент 1
- Решење 1: закључај – измени – откључај
  - Клијент 1 преузме датотеку и закључа је
  - Клијент 1 измени датотеку
  - Клијент 1 постави измене и откључа датотеку
  - Клијент 2 не може преузети датотеку док је закључана
- Недостаци овог решења:
  - Чекање је можда непотребно јер клијенти не раде на истом делу датотеке
  - Много зависи од клијента - рецимо, шта ако клијент заборави да откључа
  - Ако је датотека тренутно закључана, када ће бити откључана - колико дуго клијент да чека?
  - Нису избегнуте неконзистентности које настају услед некомпатибилних измена различитих датотека

Дељење датотека 2/3

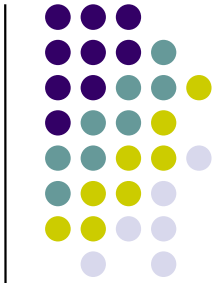


- **Решење 2: копирај - измени - уклопи**
  - Клијент 1 и Клијент 2 узимају копију исте датотеке са репозиторијума
  - Клијенти мењају различите делове датотеке
  - Клијент 1 поставља измене на репозиторијум, правећи тако нову верзију датотеке
  - Клијент 2 поставља измене на репозиторијум, алат уклапа измене са изменама које је направио Клијент 1, правећи тако још новију верзију датотеке
- Решење функционише савршено ако клијенти не мењају исти део датотеке
- Ако мењају исти део датотеке, настаје проблем
- Због тога се примењује модификована верзија горњег решења (прва 3 корака су иста):
  - Клијент 2 покушава да постави измене на репозиторијум, али га алат обавештава да је сада на репозиторијум новија верзија датотеке од оне коју је он првобитно преузео
  - Клијент 2 онда мора да свуче ту најновију верзију, а алат обавља уклапање на клијентовој локалној копији
  - Сада постоје два случаја:
  - 1. Уклапање је успешно прошло - измењени су различити делови датотеке
    - Клијент 2 након уклапања поново поставља измене на репозиторијум, овога пута успешно
  - 2. Дошло је до конфликта током уклапања - измењени су исти делови датотеке
    - Клијент 2 прво мора решити конфликте ручно, па тек онда може поновити постављање на репозиторијум

Дељење датотека 3/3

- Описано решење ради добро у случају текстуалних датотека (подељених на линије)
- Када нема конфликта све се обавља аутоматски
- Конфликти су веома ретки, зато што различити клијенти обично раде на различитим стварима у коду, а и када се јаве обично су лако решиви
- SVN је првенствено систем базиран на копирај-измени-уклопи концепту, али омогућава и закључавање датотека
- Закључавање се чешће примењује на бинарне датотеке јер је уклапање тих врста датотека обично немогуће

Радна копија 1/2



- Радна копија представља скуп директоријума и датотека на локалној машини, који осликавају стање на репозиторијум
- Радна копија је независна од репозиторијума и све измене на радној копији не утичу на репозиторијум све док се на уради експлицитна акција постављања измена на репозиториујм
- Овако изгледа типичан рад са радном копијом:

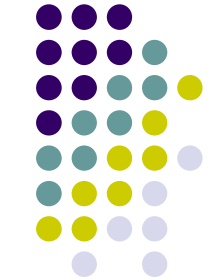
1) На почетку се прави радна копија коричћење **checkout** наредбе.

   Обавља се иницијално копирање тренутног садржаја на репозиторијуму (са главе), са репозиторијума на локлану машину.

```
svn checkout http://svn.example.com/repos/example
```

```
A inc/factorial.h
A inc/math.h
A src/app.c
A src/factorial.c
A src/math.c
A lib/libwrite.so
A libinc/libwrite.h
Checked out revision 29.
```

Радна копија 2/2



2) Мења се садржај једне, или више, датотека (нпр. factorial.c)

3) Када су измене локално потврђена као добре (билдовањем и неком врстом испитивања) онда се измене постављају назад на репозиторијум коришћењем **commit** наредбе. Ако локално измењене датотеке нису у међувремену измењене на репозитријуму, постављање (комитовање) ће проћи успешно

```
svn commit factorial.c -m “Fixed bug #123 from Bugzilla"
```

```
Sending factorial.c
Transmitting file data .
Committed revision 30.
```

- Неки други клијент унете измене на репозиторијум може добавити коришћењем **update** наредбе:

```
svn update
```

```
U factorial.c
Updated to revision 30.
```

- Нареба **checkout** се примењује само на самом почетку рада. Наредни циклуси измена почињу са коришћењем команде **update**.

Увид у промене у радној копији

- Провера унесених измена пре њиховог комитовања представља добру праксу
- На тај начин се спречава комитовање случајних измена
- Уједно, сагледавање свих измена помаже приликом формулисања доброг описа шта је комитовано (да, и овде се мора коментарисати)
- Коришћењем **status** наредбе могуће је добити листу датотека које су измењене у односу на верзије на репозиторијуму, подразумевано у односу на главу.

```
svn status app
```

```
Output:
M       inc/factorial.h
M       src/factorial.c
```

- Иста наредба даје информације и о датоткема које су измењене на репозиторијуму након момента када је радна копија последњи пут освежена (апдејтована).

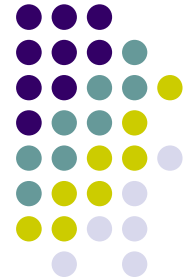
```
svn status --show-updates app
```

```
Output:
*       src/app.c
```

- Детаљније информације о изменама се могу добити **diff** наредбом.

```
svn diff src/factorial.c
```

Постављање измена - комитовање



- Најважнији тренутак при раду са SVN-ом је комитовање измена
- Најпре треба, као што је на претходним слајдовима објашњено, проверити које датотеке су све измењене и шта је у њима измењено.
- Уколико се процени да у изменама на радној копији има више логичких целина онда их треба поделити у више одвојених комитова!
- Приликом комитовања аутоматски ће остати информација шта је комитовано, ко је комитовао и када је комитовао.
- Осим тога, од корисника се очекује да да кратак опис измена које је унео.
- Опис измена је врло важан!!!
- Застати, размислити и сажето срочити коментар.
- На пример: „Исправка бага који се јавља при раду са непарним бројевима.”, „Промењен алгоритам за сортирање. Сада се користи бабл сорт јер се испоставило да је одржавање поретка важно за правилно функционисање програма.”, „Само мало дотеран код и додати неки коментари.”, „Привремено избачен код који проверава дубину стабла.” итд.
- Коментари некада требају бити и опширнији. Никада немојте бити лењи по том питању!

Конфликти 1/3

- Конфликти се дешавају када се апдејтује са репозиторијума
- Пример:

```
Files changed in repository:
inc/math.h
src/app.c
src/factorial.c
Files changed in working copy:
src/app.c – different part changed compared to
    repository
src/factorial.c – same part changed as in
    repository
```

```
svn update
```

```
Output:
U inc/math.h
G src/app.c
Conflict discovered in 'src/factorial.c'.
```

```
app
|
|-- inc
|    |-- app.h
|    |-- factorial.h
|    |-- math.h
|
|-- src
|    |-- app.c
|    |-- factorial.c
|    |-- math.c
|
|-- lib
|    |-- libwrite.so
|
|-- libinc
|    |-- libwrite.h
|
|-- build
```

- U – значи да је верзија у радној копији замењена новијом са репозиторијума
- G – значи да обављено аутоматско уклапање
- Конфликти су очигледно пријављени

Конфликти 2/3

- У случају конфликта локална копија датотеке бива измењена на следећи начин

```
line 1
line 2
<<<<<<< .mine
local line 3
local line 4
=======
repository line 3
repository line 4
>>>>>>> .r29
line 5
line 6
```

- Делови кода обележени са <<<<, ===== и >>>> су разлог конфликта
- Тај део кода мора бити ручно модификован и обележивачи морају бити уклоњени
- Код из радне копије се налази између <<<< и ====
- Код на репозиторијуму се налази између ==== и >>>>

```
Working copy:

<<<<<<< .mine
local line 3
local line 4
=======
```

```
Repository copy:

=======
repository line 3
repository line 4
>>>>>>> .r29
```

Конфликти 3/3

- Конфликти се обично решавају кроз консултације особа које су заједно промениле датотеку. (Додуше, некада је и очевидно шта треба урадити и без консултације са другом особом)
- Најчешће решавање конфликта се своди на избор верзије која ће бити употребљена на месту конфликта, али неретко се мора направити комбинација две верзије
- Када су конфликти решени (не заборавите да у једној датотеци може бити више конфликта) и из датотеке су избачени обележивачи конфликта, мора се употребити **resolve** наредба да би се SVN обавестио о томе:

```
svn resolve src/factorial.c
```

- Опрез: након обележавања датотеке као разрешене могуће ју је комитовати без обзира на њен садржај!
- На крају је важно комитовати датотеку, јер тек тада измене постају видљиве и на репозиторијуму

```
svn commit -m “Comment describing modification."
```

Гледање историје 1/2

- SVN памти све измене икад направљене на репозиторијуму
- SVN нуди могућност гледања свих измена поређаних хронолошки, као и добављање сваке од претходних верзија
- **svn log** – приказује информације о свим верзијама/ревизијама кода

```
svn log
```

```
------------------------------------------------------------------------
r3 | client1 | 2010-07-12 23:09:28 -0500 (Mon, 12 July 2010) | 1 line
Added include lines and corrected comments.
------------------------------------------------------------------------
r2 | client2 | 2010-07-07 18:43:15 -0500 (Wed, 07 July 2010) | 1 line
Added calculate() function.
------------------------------------------------------------------------
r1 | client1 | 2010-07-02 19:50:31 -0500 (Fry, 10 July 2010) | 1 line
Initial import
------------------------------------------------------------------------
```

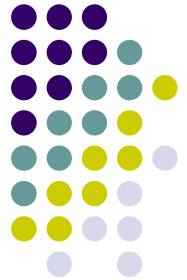
- Могуће је посматрати податке само за једну ревизију или опсег ревизија

```
svn log –r 2
svn log –r 1:3
```

- Такође, могуће је гледати историју само за једну датотеку

```
svn log app.c
```

Гледање историје 2/2



- **svn diff** наредба даје детаљан приказ измена за сваку датотеку
- Подразумевано ова наредба приказује разлике између радне копије и последње ревизије на репозиторијуму (главе)
- Са опцијом -r се може проследити и број ревизије са којом радна копија треба да се упореди

```
svn diff –r 2
```

- Могу се поредити и две различите ревизије на репозиторијуму

```
svn diff –r 1:2
```

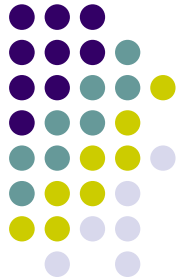
- **svn cat** наредба приказује верзију датотеке из одређене ревизије

```
svn cat –r 2
```

```
#include "avglib.h"

int main()
{
    printf("average value %lf\n", avg(3.7, 4.6));
    return 0;
}
```

Добављање ранијих верзија



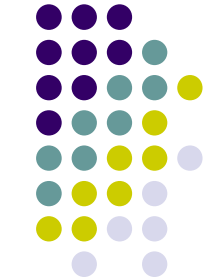
- Могуће је добавити ранију верзију са репозиторијума коришћењем checkout и update наредбе, додавењем опције -r, као са наредбама за гледање историје. Без те опције ове наредбе раде подразумевано са главом
- Добавља нову радну копију од ревизије 2:

```
svn checkout –r 2
```

- Апдејтује радну верзију на ревизију 2:

```
svn update –r 2
```

Прављење новог репозиторијума



- Нови SVN се прави коришћењем наредбе **svnadmin create**

```
svnadmin create /data/repositories/projects
```

- Ово се покреће на серверу на којем ће бити репозиторијум
- Не може са удаљене машине
- Када је репозиторијум направљен треба га попунити садржајем. Најбржи начин за то је **import** наредба

```
svn import app /data/repositories/projects/app/ -m "Initial import"
```

```
app
|
|-- inc
|   |-- app.h
|   |-- factorial.h
|   |-- math.h
|
|-- src
|   |-- app.c
|   |-- factorial.c
|   |-- math.c
|
|-- lib
|    |-- libwrite.so
|
|-- libinc
|    |-- libwrite.h
|
|-- build
```

```
Adding inc/factorial.h
Adding inc/math.h
Adding src/app.c
Adding src/factorial.c
Adding src/math.c
Adding lib/libwrite.so
Adding libinc/libwrite.h

Committed revision 1
```

- У овом примеру app директоријум одакле се импортује је најобичнији директоријум на локалном диску

Прављење измена на репозиторијуму

- Наредба **add** – обележава датотеку за додавање на репозиторијум

```
svn add appnew.c
```

- Датотека ће бити додата тек када се обави комитовање
- У случају прослеђивања директоријума као параметра, све датотеке из тог директоријума ће бити додате
- Наредба **delete** – обележава датотеку за брисање са репозиторијума

```
svn delete appnew.c
```

- Датотека ће бити обрисана у радној копији одмах, али на репозиторијум тек након комитовања.
- Наредба **copy:**

```
svn copy app.c appnew.c
```

- Прави дупликат датотеке и аутоматски га обележава за додавање.
- Копира се и историја датотеке, тако да ће бити везана и за нову датотеку
- Наредба **move** - као copy али оригинална датотека се обележава за брисање

```
svn move app.c appnew.c
```

Гранање 1/2

- **Грана** је ток развоја програма који је независан од неког другог, паралелног тока (или токова), и са њим има заједничку историју.

- Три су уобичајена сценарија када долази до гранања:
  - Унос значајније промене у код

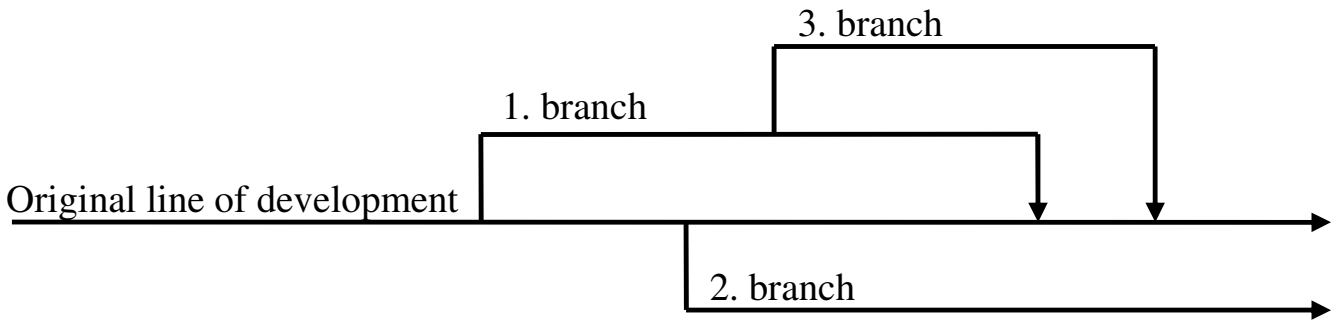
    Значајна измена кода (зарад увођења нових могућности, на пример) чија имплементација би дуго трајала могла би током тог периода имати неповољан утицај на рад на осталим изменама. Пребацивањем тог посла на грану, привремено се обезбеђује несметан рад и на тој крупној измени и на остатку кода.
  - Увођење неких измена само за одређене употребе

    Рецимо, само за неку муштерију је потребно имплементирати одређену могућност, а за остале муштерије она није потребна, или је чак непожељна.
  - Припремање за испоруку

    Што се тиче могућности које пружа, код је спреман за испоруку али још га треба утегнути, исправити багове и слично. Уколико поред тога још постоји потреба развоја за наредне испоруке, онда се припрема кода за предстојећу испоруку одваја у посебну грану. (Или се ради обрнуто: припрема се ради на стаблу, а даљи развој се одваја у грану.)

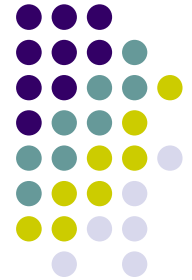
Гранање 2/2

- Грана увек почиње као копија неке ревизије, а затим наставља своју историју
- Основни ток развоја се назива **стабло** (eng. trunk)



- SVN омогућава:
  - Прављење копија ревизије
  - Поновно придруживање грана и повремено синхронизовање
  - Апдејтовање радне копије са верзијама са разних грана

Прављење гране



- Грана је напросто копија неке ревизије
- За то се користи svn copy наредба

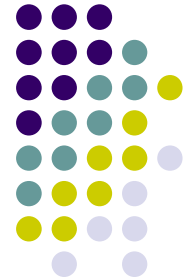
```
svn copy http://svn.example.com/repos/example/trunk \
http://svn.example.com/repos/example/branches/my-app \
    -m “creating new branch of app project”
```

```
Output:
Committed revision 17.
```

- Копирање се обавља независно од радне копије
- Све копије поседују историју оригиналних датотека
- Додатни коментар у вези са гранањем се памти на репозиторијуму
- Код SVN-а грана је само логичка конструкција, сам SVN је не види ништа посебније од копије датотека

```
|-- app
    |
    |-- trunk
    |   |-- inc
    |   |   |-- app.h
    |   |   |-- factorial.h
    |   |   |-- math.h
    |   |
    |   |-- src
    |   |   |-- app.c
    |   |   |-- factorial.c
    |   |   |-- math.c
    |   |
    |   |-- lib
    |   |   |-- libwrite.so
    |   |
    |   |-- libinc
    |   |   |-- libwrite.h
    |
    |-- branches
        |-- my-app
            |-- inc
            |   |-- app.h
            |   |-- factorial.h
            |   |-- math.h
            |
            |-- src
            |   |-- app.c
            |   |-- factorial.c
            |   |-- math.c
            |
            |-- lib
            |   |-- libwrite.so
            |
            |-- libinc
            |   |-- libwrite.h
```

Синхронизовање гране



- Ако грана постоји дуго добра пракса је да се повремено синхронизује са стаблом, да не би дошло до превелике разлике у кодовима, јер у том случају припајање гране назад на стабло може бити мукотрпан посао (услед великог броја могућих конфликата).

- Поступак:

1) Апдејтовати радну копију везану за грану на главу гране, а затим ово позвати:

```
svn merge http://svn.example.com/repos/example/trunk
```

```
Output:
--- Merging r17 through r19 into '.':
U app.c
U math.c
```

2) Све измене услед уклапања гране са стаблом налазе се у радној копији. Сада прегледати измене и решити све конфликте који су се можда јавили.

3) На крају, обавити комитовање назад на грану:

```
svn commit –m “synchronized with trunk”
```

```
Sending app.c
Sending math.c
Transmitting file data ..
Committed revision 20.
```

Припајање гране

- Када је развој на грани завршен онда у одређеним случајевима постоји потреба за припајањем гране назад на стабло.
- Ово се обавља коришћењем **svn merge** наредбе
- Поступак:

1) У радној копији везаној за грану обавити синхронизацију са стаблом:

```
svn merge http://svn.example.com/repos/example/trunk
```

```
svn commit –m “synchronized with trunk”
```

2) Затим направити нову радну копију везану за стабло. Употребом **checkout** наредбе.

3) Сада поново употребити merge наредбу, али овог пута у радној копији везаној за стабло. Овога пута употребљава се опција --reintegrated да би се SVN-у саопштило да је дошло до припајања гране.

```
svn merge --reintegrate   \
http://svn.example.com/repos/example/branches/my-app
```

4) Поново проверити све измене и решити могуће конфликте.

5) Радну верзију комитовани на стабло.

```
svn commit –m “merged changes from my-app branch”
```