Controlul versiunilor

Studenti :

Nastase Alexandru **441A**

Sandru Traian **441A**

Cuprins

I. Introducere

Un sistem de control al versiunilor reprezinta un produs software ce ajuta dezvoltatorii dintr-o echipa software sa colaboreze in cadrul diferitelor proiecte si de asemenea pastreaza un jurnal complet al muncii fiecaruia.

Un sistem de control al versiunilor (SCV) are trei scopuri principale :

1. Sa ofere posibilitatea muncii simultane, nu seriale

   Pentru intelegerea perfomantei unei echipe din punct de vedere al colaborarii se poate face analogia cu un software care ruleaza pe mai multe fire de executie. In acest caz fiecare dezvoltator ar avea propriul fir de executie, iar modul in care s-ar putea obtine o performanta ridicata este similar sistemului cu mai multe fire de executie, si anume maximizarea concurentei.

2. Atunci cand mai multe persoane lucreaza in acelasi timp se asigura ca modificarile realizate de acestia nu intra in conflict unele cu altele

   In cazul programarii pe mai multe fire de executie este necesara existenta unor caracteristici precum zone critice, zavoare (locks) si instructiuni test-and-set pentru ca datele prelucrate de firele de executie sa nu intre in conflict. In mod similar un SCV trebuie sa aiba mecanisme asemanatoare care sa asigure lipsa de conflicte intre codul scris de fiecare dintre programatori.

3. Sa ofere o arhiva a fiecarei versiuni continand informatii despre cine, unde si din ce motiv a fost facuta fiecare modificare.

[1]

Istoria controlului versiunilor

In termeni generali sistemele de control al versiunilor pot fi impartite in trei generatii, fiecare noua generatie avand o aplecare tot mai mare catre cresterea folosirii concurente.

Generatie	Organizare	Operatii	Concurenta	Exemple
Prima	-	Fiecare fisier in parte	Zavoare	RCS, SCCS
A doua	Centralizata	Multi-fisier	Merge inainte de commit	CVS, SourceSafe, Subversion, Team Foundation Server
A treia	Distribuita	Changeset-uri	Commit inainte de merge	Bazaar, Git, Mercurial

In prima generatie dezvoltarea concurenta era gestionata in intregime de mecanisme de lock (rom. zavoare). Doar o singura persoana avea posibiliatea de a lucra la un anumit fisier intr-un moment dat.

In a doua generatie SCV-urile devin mai permisive din punct de vedere al modificarilor simultane, insa exista o restrictie semnificativa : utilizatorii trebuie sa isi combine (merge) reviziile curente in propriul cod inainte de a le fi permisa operatia de commit.

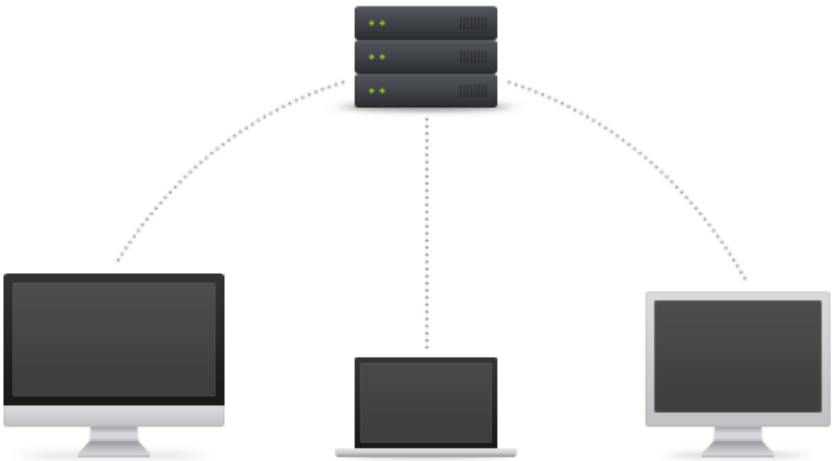
In a treia generatie SCV-urile permit ca operatiile de merge si commit sa fie separate.

[1] [2]

Cele mai folosite SCV-uri in momentul de fata sunt cele din a doua generatie. Printre cele mai notabile sunt TFS [4] (Team Foundation Server) dezvoltat de Microsoft si integrat cu tool-urile de dezvoltare Visual Studio si Apache Subversion cunoscut si sub denumirea de SVN, tool-ul open source cel mai folosit SCV in momentul de fata.[1][3]

Trebuie luat in considerare faptul ca SCV de generatie a treia au o crestere semnificativa in ultima perioada. Dintre acestea cele mai populare sunt : Git [5], un tool gratuit dezvoltat de Linus Torvalds pentru gestionarea codului Kernel-ului de Linux, Mercurial [6], o unealta ce pune accent pe performata marita si pe scalabilitate si Bazaar [7], o unealta sponsorizata de Canonical, compania responsabila de distributia de Linux Ubuntu, aceasta avand avantajul similaritatii de comenzi cu tool-ul Subversion si CVS.

II. Modelul centralizat de control al versiunilor



[8]

2.1. Concepte de baza

In momentul de fata cel mai popular sistem de control al versiunilor este Subversion sau SVN, care este considerat un SCV centralizat. Principiul de baza al sistemelor centralizate se bazeaza pe relatia client-server. Un depozit (repository) este situat intr-un singur loc iar mai multi clienti au acces la el. Aceasta organizare este similara cu protocolul FTP (File Transfer Protocol) in care exista un client FTP care se conecteaza la un server FTP. Toate modificarile utilizatorilor si toate informatiile legate de aceste modificari (utilizator, data, revizie) sunt transmise si preluate de la un depozit (repository) central.

Principalele beneficii ale lui Subversion :

- Este usor de inteles
- Exista un control mai riguros al utilizatorilor si al accesului avand in vedere ca schimbul de informatii se face cu o sursa centrala

- Exista mai multi clienti GUI si mai multe integrari cu IDE (Integrated Development Environment) datorita faptului ca acesta este de mai mult timp pe piata
- Este usor de inceput munca cu el

Principalele dezavantaje ale sistemului Subversion:

- Dependenta de accesul la server
- Mententanta si backup-urile serverului
- Poate fi mai greoi datorita necesitatii de comunicare cu serverul
- Tool-urile de branching si merging sunt greu de folosit

[8]

Cele mai importante operatii ale unui sistem de control al versiunilor centralizat sunt * :

* - O parte dintre acestea se aplica si sistemelor distribuite

1. **Create** : Creeaza un nou repository.

Repository-ul este locul unde tot lucrul dezvoltatorilor este stocat. El tine jurnalul arborelui, tuturor fisierelor utilizatorilor, precum si directoarelor in care aceste fisiere se afla. Pe langa acestea repository-ul stocheaza jurnalul tuturor modificarilor. In timp ce sistemul de fisiere este bidimensional, fiind definit de fisiere si directoare, repository-ul este tridimensional fiind definit de directoare fisiere si timp. Un repository contine toate versiunile unui cod sursa ce au aparut de la crearea acestuia.

2. **Checkout** : Creeaza o copie de lucru (working copy).

Operatie de checkout este invocata atunci cand este necesara creearea unei noi copii de lucru pentru un repository care exista deja. O copie de lucru este o copie integrala a repository-ului si reprezinta locul unde dezvoltatorul de software va face modificari. Repository-ul este impartit de mai multi dezvoltatori insa modificarile nu se fac direct asupra acestuia, ci fiecare face modificarile asupra copiei de lucru.

Ciclul de lucru al unui dezvoltator la un proiect poate fi rezumat in termeni generali prin 4 pasi :

1. Face o copie de lucru cu continutul repository-ului
2. Modifica repectiva copie de lucru
3. Actualizeaza repository-ul pentru a ingloba acele modificari
4. Revine la pasul 2

In mod uzual sistemul de control al versiunilor mai pastreaza si alte informatii suplimentare in copia de lucru. Spre exemplu :

- O unealta SCV poate sa stocheze marca de timp impreuna cu un fisier pentru a putea determina mai tarziu daca fisierul respectiv a fost modificat.
- Poate stoca numar de versiune al unui fisier luat din repository ca mai tarziu sa stie care este punctul de pornire de la care dezvoltatorul a inceput sa faca modificari.
- Poate stoca o intreaga copie a fisierului care a fost luat de pe repository pentru ca mai tarziu sa poate face operatia de diff fara sa mai fie nevoie de accesul la server.

3. **Commit** : Aplica repository-ului modificarile de pe copia de lucru, creand un nou changeset (set de schimbari).

Aceasta este o operatie care modifica propriu-zis repository-ul. Alte operatii modifica copia de lucru si adauga operatii setului de schimbari in asteptare (pending changeset), un loc unde operatiile asteapta operatia de commit. Operatia de commit foloseste setul de schimbari in asteptare pentru a creea o noua versiune a arborelului din repository.

Un aspect important al acestei operatii este ca ea se realizeaza in mod atomic, adica indiferent de numarul de schimbari din setul de schimbari in asteptare , ori se vor realizeaza toate aceste operatii daca nu apare nicio eroare ori nu se realizeaza nicuna in cazul in care una sau mai multe modificari nu sunt in regula.

4. **Update** : Se actualizeaza copia de lucru cu informatiile din repository.

Operatia de update actualizeaza repository-ul aplicand modificarile de pe acesta si combinandu-le cu alte modificari realizate pe copia de lucru daca este cazul.

5. **Operatii cu directoare si fisiere**

Add : Adauga un fisier sau un director

Operatia add se foloseste atunci cand dorim sa adaugam un fisier sau un director din copia de lucru care nu este inregistrat de SCV. Adaugarea nu se face imediat ci se adauga setului de schimbari in asteptare, urmand sa fie adaugat dupa commit.

Edit : Modifica un fisier.

Aceasta este una dintre cele mai comune operatii ale sistemului de control al versiunilor facand parte dintre principalele activitati din ciclul de lucru unui dezvoltator (vezi ciclul de lucru).

Delete : Sterge un fisier sau un director.

In mod uzual operatia de delete se aplica imediat pe copia de lucru insa stergerea propriu-zisa de pe repository este adaugata in setul de schimbari in asteptare(pending changeset).

Rename : Redenumeste un fisier sau un director.

Ca si in cazul altor operatii acesta este adaugata listei de schimbari in asteptare, fiind executata imediat doar pentru copia de lucru. Exista implementari diferite in ceea ce priveste operatia de redenumire. In timp ce unele tool-uri precum Bazaar si Veracity implementeaza redenumirea formal, numele fisierului sau directorului reprezentand o caracteristica ce se poate schimba pe parcursul timpului, altele precum Git au o abordare informala, detectand schimbarile ce apar in urma redenumirii.

Move : Muta un fisier sau un director.

Aceasta operatie este folosita cand se doreste mutarea unui fisier sau director dintr-un anumit loc al arborelui in altul. Si in acest caz schimbarea se face imediat in copia de

lucru, pentru repository fiind adaugata in setul de schimbari in asteptare. Unele tool-uri trateaza operatiile de rename si move ca pe aceeasi operatie respectand traditia sistemelor Unix, in timp ce altele pastreaza diferenta dintre aceste operatii.

6. **Status** : Listeaza modificarile facut copiei de lucru.

Pe masura ce sunt facute modificari asupra copiei de lucru acestea sunt adaugate setului de schimbari in asteptare. Operatia status e folosita pentru a arata setul de schimbari in asteptare.

7. **Diff** : Arata detaliile modificarilor facute asupra copiei de lucru.

Operatia status arata o lista cu schimbari insa nu ofera informatii despre acestea. Rezultatul operatiei de diff poate fi printat in consola sau afisat cu ajutorul unei aplicatii diff vizuala.

8. **Revert** : Revoca modificari facute asupra copiei de lucru.

Pot exista situatii in care in timpul rezolvarii unei probleme printr-o anumita abordare se poate gasi o solutie mai buna fiind nevoie ca schimbarile realizate in timpul incercarii sa fie revocate.

O operatie de revert a intregii copii de lucru este echivalenta cu stergerea setului de schimbari in asteptare. Aceasta ar aduce copia de lucru in aceasi stare ca dupa operatia de checkout.

9. **Log** : Arata un jurnal al schimbarilor de pe repository.

Repository-ul memoreza toate versiunile care au existat de-a lungul timpului. Operatia de log este modul de vizualizare al acestor versiuni. Aceasta afiseaza informatii despre setul de schimbari (changeset) impreuna cu informatii suplimentare despre persoana care a facut schimbarile, momentul in care au fost facute si un mesaj de loc atasat de persoana care le-a facut in care detaliaza rolul acestora.

10. **Tag** : Asociaza un nume cu semnificatie fiecarei versiuni specifice din repository.

SCV-urile ofera un mod de asociere a unei anumite instante din jurnal cu un nume cu semnificatie.

11. **Branch** : Creaza o noua linie de dezvoltare.

Operatia de branch este folosita atunci cand dezvoltarea unui anumit produs are mai multe ramuri. Spre exemplu daca avem un produs care sa afla la versiunea 1.0, putem crea un branch pe care sa se inceapa dezvoltarea versiunii 2.0 si altul pe care sa se faca repararea de bug-uri a primei versiuni.

12. **Merge** : Aplica schimbari de pe o ramura pe alta.

In mod uzual dupa ce s-a facut impartirea pe ramuri a muncii se doreste o convergenta a acestora, cel putin partiala. Tinand cont de exemplul anterior, este de dorit ca bug-urile reparate din prima versiune sa fie reparata si in cea de-a doua versiune, acest lucru s-ar putea face manual, acum acele bug-uri fiind cunoscute, insa merge ofera posibilitatea automatizarii acestei operatii.

13. **Resolve** : Gestioneaza conflicte aparute in urma operatiei merge.

In unele cazuri operatia de merge necesita interventia unei persoane. In general aceasta realizeaza toate modificarile care sunt vazute ca sigure, in caz contrar aceste modificari sunt considerate conflicte. Spre exemplu daca un fisier a fost modificat pe un anumit branch si pe un altul a fost sters. Intr-o astfel de situatie persoanele implicate in dezvoltarea pe acele ramuri trebuie sa cada de comun acord si sa rezolve conflictul.

14. **Lock** : Previne alte persone din a modifica un fisier.

Operatia de lock este folosita cand se doreste dreptul exclusiv asupra unui anumit fisier. Nu toate sistemele de control al versiunilor acesta operatie.

[1]

2.2. Exemplu folosind SVN

Se va prezenta un scenariu in care se doreste realizarea unui produs ce calculeaza probabilitatea(ca un procent intreg) de castigare a unei loterii numita Powerball pentru orice set de numere. Aceasta loterie implica extragerea de 5 bile albe si o bila rosie. Se presupune ca firma responsabila a desemnat doi dezvoltatori pentru acest produs : Harry, aflat in Birmingham, Marea Britanie si altul Sally aflata in Birmingham, Statele Unite, iar serverul central se afla la sediul firmei in Cleveland, SUA.

Se vor prezenta operatiile de Create, Checkout, Add, Status si Commit

Sally incepe proiectul prin crearea unui nou repository :

```
~ server$ cd
~ server$ mkdir repos
~ server$ svnadmin create repos/lottery
~ server$ svnserve -d --root=/Users/sally/repos
```

Dupa aceasta Harry creeaza o copie de lucru facand operatia de checkout:

```
~ harry$ svn checkout svn://server.company.com/lottery
Checked out revision 0.
```

Apoi verifica daca Sally a facut ceva in noul repository :

```
~ harry$ cd lottery
lottery harry$ ls -al
total 0
drwxr-xr-x 3 harry staff 102 Apr 6 11:40 .
drwxr-xr-x 3 harry staff 102 Apr 6 11:40 ..
drwxr-xr-x 7 harry staff 238 Apr 6 11:40  .svn
```

Observa ca nu si incepe sa scrie codul noului produs :

```
#include <stdio.h>
#include <stdlib.h>
int calculate_result(int white_balls[5], int power_ball)
{
return 0;
}
int main(int argc, char** argv)
```

```
{
if (argc != 7)
{
fprintf(stderr, "Usage: %s power_ball (5 white balls)\n", argv[0]);
return -1;
}
int power_ball = atoi(argv[1]);
int white_balls[5];
for (int i=0; i<5; i++)
{
white_balls[i] = atoi(argv[2+i]);
}
int result = calculate_result(white_balls, power_ball);

printf("%d percent chance of winning\n", result);
return 0;
}
```

Inainte sa faca operatia de commit acesta vrea sa verifice daca programul compileaza si ruleaza corect :

```
lottery harry$ gcc -std=c99 lottery.c
lottery harry$ ls -l
total 32
-rwxr-xr-x 1 harry staff 8904 Apr 6 12:15 a.out
-rw-r--r-- 1 harry staff 555 Apr 6 12:15 lottery.c
lottery harry$ ./a.out
Usage: ./a.out power_ball (5 white balls)
lottery harry$ ./a.out 42 1 2 3 4 5
0 percent chance of winning
```

Dupa aceasta decide sa adauge fisierul pe repository. Acesta va fi adaugat setului de schimbari in asteptare.

```
lottery harry$ svn add lottery.c
A lottery.c
```

Verifica daca a fost adaugat setului de schimbari in asteptare :

```
lottery harry$ svn status
? a.out
A lottery.c
```

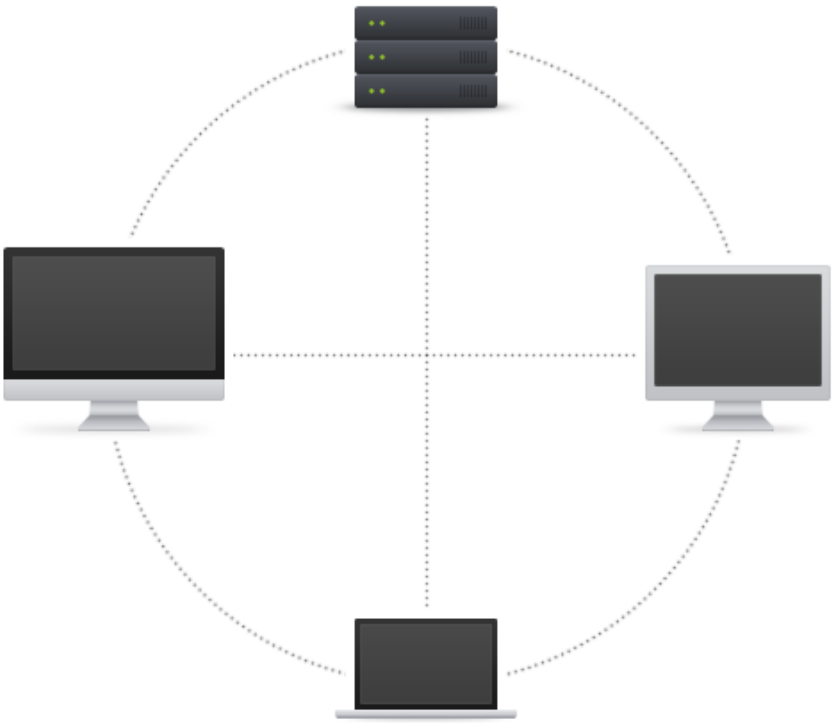
SVN il atentioneaza pe Harry ca nu stie ce sa faca in privinta fisierului a.out care este un fisier executabil si nu trebuie adaugat pe repository.

Harry face commit fisierului :

```
lottery harry$ svn commit -m "initial implementation"
Adding lottery.c
Transmitting file data .
Committed revision 1
```

[1]

III. Modelul distribuit de control al versiunilor



[8]

3.1. Concepte de baza

Sistemele de control al versiunilor distribuite sunt o optiune mai noua. In cadrul acestora fiecare utilizator are propria copie a intregului repository, nu doar fisiere, ci intregul jurnal. Aceasta abordare foloseste modelul peer-to-peer spre deosebire de modelul client-server folosit de sistemele centralizate. In acest caz sincronizarea dintre repository-uri este realizata prin schimbul de changeset-uri sau patch-uri dintre statii. Doua dintre cele mai folosite SCV-uri de acest fel sunt Git si Mercurial.

Principalele beneficii ale sistemelor Git si Mercurial sunt :

- O urmarire a schimbarilor mai avansata si mai detaliata, lucru care duce la mai putine conflicte
- Lipsa necesitatii unui server – toate operatiile cu exceptia schimbului de informatii intre repository-uri se realizeaza local
- Operatiile de branching si merging sunt mai sigure, si prin urmare folosite mai des
- Rapiditate mai mare a operatiilor datorita lipsei necesitatii comunicarii cu serverul

Principalele dezavantaje ale sistemelor Git si Mercurial :

- Modelul distribuit este mai greu de inteles
- Nu exista asa de multi clienti GUI datorita faptului ca aceste sisteme sunt mai noi
- Reviziile nu sunt numere incrementale, lucru ce le face mai greu de referentiat
- Riscul aparitiei de greseli este mare daca modelul nu este familiar

[8][9]

Operatii importante ale unui sistem de control al versiunilor distribuit :

1. **Clone** : Creeaza o noua instanta de repository, care este copia unui alt repository.

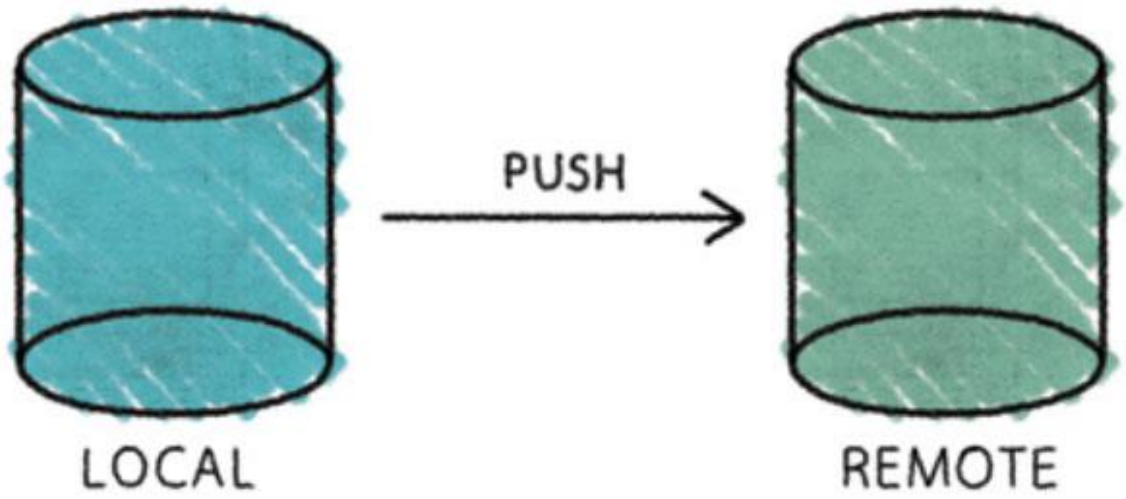
Un aspect important al diferentei dintre SCV-urile centralizate si cele distribuie il reprezinta notiunea de instanta de repository.

Ca si in cazul modelului centralizat cand se creeaza un repository pentru prima data se foloseste operatia create. Dupa aceasta, lucrul care deosebeste modelul distribuit de cel centralizat este faptul ca putem avea mai multe instante ale acelui repository. Modul in care se creeaza aceste instante este folosind operatia clone.

Nu numai ca modelul distribuit permite existenta mai multor instante, ci acesta este in general modul in care functioneaza. Majoritatea operatiilor folosesc o instanta locala a repository-ului si nu comunica cu serverul, singura data cand este necesara comunicarea

cu serverul este atunci cand se doreste sincronizarea acestor instante. Fiecare dezvoltator are propria instanta de repository. Un aspect care este inteles gresit este acela ca modelul distribuit nu elimina serverul central ci doar ofera mai multa flexibilitate. In cazul modelului centralizat, serverul central este singurul loc unde se gaseste instanta repository-ului, in timp ce modelul distribuit are mai multe instante ce pot avea fiecare un scop anume.

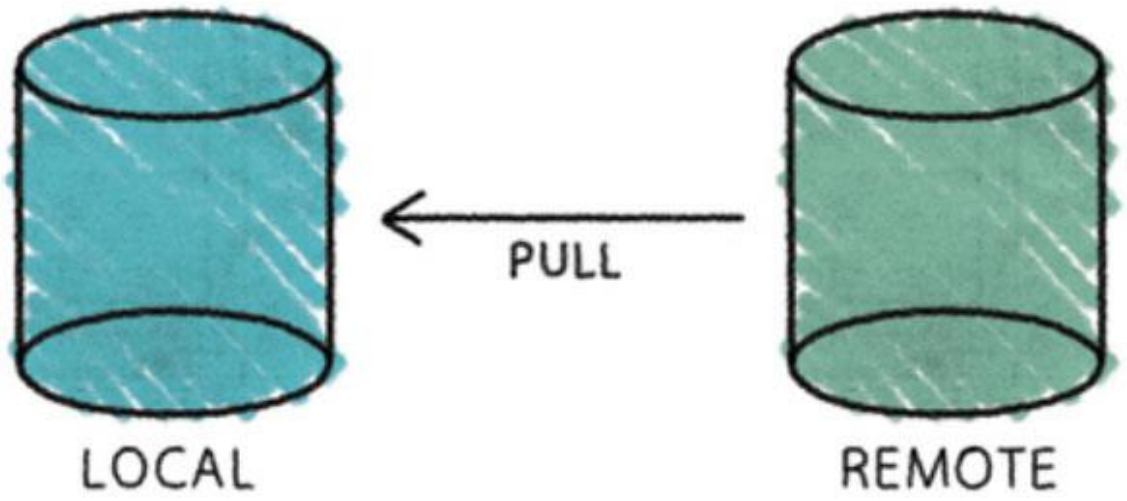
2. **Push** : Copiaza seturi de schimbari (changesets) de pe o instante locala de repository pe una straina.



Operatia de push este folosita pentru sincronizarea dintre doua instante de repository. Mai exact aceasta operatie se face din perpectiva instantei locale de repository care vrea sa copieze o parte din seturile de schimbari intr-o instanta straina (remote). Uzual, instanta straina este aceea de pe care instanta locala a fost clonata.

Este de mentionat faptul ca cele doua instante de repository nu sunt obligatoriu identice dupa operatia de push. Putem limita operatia de push la doar o parte dintre schimbarile locale. De asemenea mai exista posibiliatea ca instanta straina sa contina informatii care nu se afla pe instanta locala.

3. **Pull** : Copiaza seturi de schimbari de pe o instanta straina pe una locala.



Similar cu operatia de push si operatia de pull are ca scop sincronizarea dintre doua instante de repository. Mai exact aceasta operatie se face din prespectiva instantei locale de repository care vrea sa copieze aceleasi seturi de schimbari de pe instanta straina. In mod uzual instanta straina este cea de pe care a fost clonata instanta locala.

Un alt aspect imporant este faptul ca instantele nu sunt in mod obligatoriu identice dupa operatia de pull. Pot exista restrictii asupra operatiei de pull (nu includ toate schimbarile de pe instanta straina). O alta situatie poate fi aceea in care instanta locala poate contine informatii care nu se afla pe cea straina. Daca se doreste o sincronizare a celor doua instante va fi necesara de o operatie de pull a tuturor informatiilor aflate pe instanta straina si o operatiile de push a tuturor informatiilor de pe instanta locala.

[1]

3.2. Exemplu folosind Git

Se va prezenta un scenariu in care se doreste realizarea unui produs ce calculeaza probabilitatea(ca un procent intreg) de castigare a unei loterii numita Powerball pentru orice set de numere. Aceasta loterie implica extragerea de 5 bile albe si o bila rosie. Se presupune ca firma responsabila a desemnat doi dezvoltatori pentru acest produs : Harry, aflat in Birmingham, Marea Britanie si Sally aflata in Birmingham, Statele Unite, iar serverul central se afla la sediul firmei in Cleveland, SUA.

Se vor prezenta operatiile Create, Clone, Add, Status, Commit, Push, Pull, Log si Diff.

1. **Create**

Sally incepe proiectul prin crearea unui nou repository :

```
~ server$ mkdir lottery
~ server$ cd lottery
lottery server$ git init --bare lottery.git
```

2. **Clone, Add, Status, Commit**

Dupa aceasta Harry vrea sa inceapa sa scrie codul pentru produsul software. Fiind prima data cand foloseste Git, mai intai seteaza fisierul .gitconfig cu informatiile pe care le va folosi pentru identificarea operatiilor de commit in log.

```
[user]
name = Harry
email = harry@company.com
```

Dupa aceasta isi creeaza propria instanta de repository.

```
~ harry$  git clone http://server.company.com:8000/ ./lottery
Cloning into lottery...
warning: You appear to have cloned an empty repository.
```

Este imporant de observat faptul ca git nu are comanda de checkout. Comanda git checkout exista dar este echivalenta cu operatia de update. Git pastreaza instanta de repository in aria administratica a copiei de lucru, astfel git clone realizand atat operatia de clone cat si pe cea de checkout.

Harry verifica daca Sally a mai facut vreo modificare repository-ului

```
~ harry$ ls -al lottery
```

```
total 0
drwxr-xr-x 3 harry staff 102 February 01 07:55 .
drwxr-xr-x 21 harry staff 714 February 01 07:55 ..
drwxr-xr-x 8 harry staff 272 February 01 07:55  .git
```

Observa ca nu e cazul si incepe sa scrie codul pentru software-ul cerut.

```
#include <stdio.h>
#include <stdlib.h>
int calculate_result(int white_balls[5], int power_ball) {
     return 0;
}
int main(int argc, char** argv)
{
if (argc != 7) {
fprintf(stderr, "Usage: %s power_ball (5 white balls)\n", argv[0]);
     return -1;
}
     int power_ball = atoi(argv[1]);
int white_balls[5];
for (int i=0; i<5; i++) {
     white_balls[i] = atoi(argv[2+i]);
}
     int result = calculate_result(white_balls, power_ball);
     printf("%d percent chance of winning\n", result);
     return 0;
}
```

Inainte sa faca operatia de commit acesta vrea sa verifice daca programul compileaza si ruleaza corect :

```
lottery harry$ gcc -std=c99 lottery.c
lottery harry$ ls -l
total 32
-rwxr-xr-x 1 harry staff 8904 February 01 07:56 a.out
-rw-r--r-- 1 harry staff 555 February 01 07:56 lottery.c
lottery harry$ ./a.out
Usage: ./a.out power_ball (5 white balls)
lottery harry$ ./a.out 42 1 2 3 4 5
0 percent chance of winning
```

Apoi Harry trebuie sa adauge fisierul in zona de staging Git, cunoscuta si sub denumirea de index in terminologia Git-ului. Aceasta este similara notiunii de set de schimbari in asteptare (pending changeset), insa intelesul este diferit pentru ca in timp ce setul de schimbari in asteptare reprezinta o lista de schimbari in copia de lucru zona de staging Git poate contine elemente care nu se afla nici in copia de lucru nici pe instanta de repository.

```
lottery harry$ git add lottery.c
```

Dupa aceasta Harry foloseste operatia status pentru a verifica schimbarile ce urmeaza a fi aplicate in urma operatiei de commit.

```
lottery harry$ git status -s
A lottery.c
?? a.out
```

Se observa ca Git avertizeaza faptul ca a.out este un fisier executabil si nu ar trebui pus pe repository.

Dupa aceasta se poate realiza operatia de commit :

```
lottery harry$ git commit -a -m "initial implementation"
[master (root-commit) 9a0ca10] initial implementation
1 files changed, 30 insertions(+), 0 deletions(-)
create mode 100644 lottery.c
```

3. **Push, Pull, Log si Diff**

Pentru ca si in cazul lui Sally este prima data cand foloseste Git pe respectiva masina trebuie sa configureze fisierului .gitconfig.

```
[user]
name = Sally
email = sally@company.com
```

Dupa aceasta isi creeaza propria instanta de repository.

```
~ sally$ git clone http://server.company.com:8000/ ./lottery
Cloning into lottery...
warning: You appear to have cloned an  empty repository .
~ sally$ cd lottery
lottery sally$ ls -al
total 0
drwxr-xr-x 3 sally staff 102 February 01 08:00 .
drwxr-xr-x 19 sally staff 646 February 01 08:00 ..
drwxr-xr-x 8 sally staff 272 February 01 08:00 .git
```

Harry ar fi trebuit sa faca commit codului pe care l-a scris insa se observa ca acesta nu exista pe repository. Aceasta problema e datorata faptului ca Harry a uitat sa faca operatia de push. Sally il informeaza pe Harry sa faca push, iar accesta se conformeaza.

```
lottery harry$ git push
No refs in common and none specified; doing nothing.
Perhaps you should specify a branch such as 'master'.
fatal: The remote end hung up unexpectedly
error: failed to push some refs to
'http://server.company.com:8000/lottery'
```

In mod implicit Git face push doar pe ramurile corespunzatoare, adica pentru fiecare ramura care exista pe instanta locala, instanta straina va fi actualizata daca exista o ramura cu acelasi nume. Asta inseamna ca prima data trebuie facuta operatia de push catre o ramura specificata explicit.

```
lottery harry$ git push --all
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 484 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
To http://server.company.com:8000/lottery
* [new branch] master -> master
```

In acest moment Sally poate face operatia de pull :

```
lottery sally$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://server.company.com:8000/lottery
* [new branch] master -> origin/master
```

Astfel Sally obtine codul scris de Harry.

```
lottery sally$ ls -al
total 8
drwxr-xr-x 4 sally staff 136 February 01 08:07 .
drwxr-xr-x 20 sally staff 680 February 01 08:06 ..
drwxr-xr-x 12 sally staff 408 February 01 08:07 .git
-rw-r--r-- 1 sally staff 555 February 01 08:07 lottery.c
```

O alta distinctie importanta ce trebuie facuta este aceea ca in general operatia de pull nu face decat sa aduca setul de schimbari in instanta locala de repository insa copia de lucru nu este actualizata. In cazul Git operatia de pull este echivalenta operatie de pull urmata de operatia de update. Git fetch este echivalenta cu definitia generala a operatiei pull.

Acum ca are codul, Sally doreste sa verifice jurnalul pentru a vedea detaliile :

```
lottery sally$ git log
commit bcb39bee268a92a6d2930cc8a27ec3402ebecf0d
Author: Harry <harry@company.com>
Date: Sat Feb 01 12:55:52 2014 +0200
initial implementation
```

Sally observa ca ar putea sa aduca o imbunatatire codului scris de Harry, modificand cateva linii de cod din main().

```
if (argc != 7)
{
fprintf(stderr, "Usage: %s (5 white balls) power_ball\n", argv[0]);
return -1;
}
```

```
int power_ball = atoi(argv[6]);
int white_balls[5];
for (int i=0; i<5; i++)
{
white_balls[i] = atoi(argv[1+i]);
}
```

Dupa aceasta foloseste operatia de status pentru a verifica schimbarile ce urmeaza a fi implementate.

```
lottery sally$ git status -s
M lottery.c
```

Pentru a verifica inca o data modificarile decide sa faca o operatie de diff :

```
lottery sally$ git diff
diff --git a/lottery.c b/lottery.c
index e59c732..adf47a7 100644
--- a/lottery.c
+++ b/lottery.c
@@ -11,16 +11,16 @@
{
if (argc != 7)
{
- fprintf(stderr, "Usage: %s power_ball (5 white balls)\n", argv[0]);
+ fprintf(stderr, "Usage: %s (5 white balls) power_ball\n", argv[0]);
return -1;
}
- int power_ball = atoi(argv[1]);
+ int power_ball = atoi(argv[6]);
int white_balls[5];
for (int i=0; i<5; i++)
{
- white_balls[i] = atoi(argv[2+i]);
+ white_balls[i] = atoi(argv[1+i]);
}
int result = calculate_result(white_balls, power_ball);
```

[1]

IV. Comparatie intre modelul distribuit si cel centralizat

[8]

Motivul pentru care modelul distribuit este considerat facand parte din a treia generatie este datorita faptului ca aduce anumite imbunatatiri fata de cel centralizat. Unele dintre aceste avantaje sunt :

1. **Spatiu de lucru privat**

Spatiul de lucru privat este una dintre ideile principale ale tuturor sistemelor de control al versiunilor. In timp ce SCV-urile centralizate ofera acest spatiu de lucru privat prin copia de lucru, SCV-urile distribuite merg mai departe oferindu-le dezvoltatorilor si o copie proprie a intregului repository.

Un dezvoltator este cel mai productiv atunci cand lucreaza singur. In momentul in care trebuie sa se coordoneze cu alte persoane in cadrul unui proiect se produce o munca in plus (overhead). Efectele acestui overhead trebuie sa fie minimizate de catre SCV pe cat de mult posibil. Prin faptul ca un dezvoltator are o copie locala a intregului repository acestuia i se ofera mai multa flexibilitate, putand realiza astfel majoritatea operatiilor posibile ale SCV-ului. Spre exemplu un dezvoltator ce foloseste un SCV distribuit are avantajul ca poate face operatia de commit ori de cate ori doreste, aceasta nefiind echivalenta cu o publicare a schimbarilor catre restul echipei. Acest lucru ii poate crea

impresia unui dezvoltator ca lucreaza singur si poate amana operatie de sincronizare a schimbarilor pana in momentul in care este pregatit sa realizeze operatie de push.

2. **Rapiditate**

In general dezvoltatorii nu realizeaza rapiditatea SCV-urilor distribuite pana in momentul in care le-au incercat. Acest lucru se datoreaza faptului ca majoritatea operatiilor se realizeaza folosind instanta locala de repository in loc de cea aflata pe server.

Exemplu de Commit a 3143 de fisiere, avand o dimensiune totala de 42MB.

Operatie	Subversion (svnserve pe 127.0.0.1)	Bazaar	Mercurial	Veracity	Git
Commit	21.9 s	5.2 s	4.6 s	3.7 s	3.2 s

[1]

3. **Lucru Offline**

In anumite situatii poate reprezenta un avantaj faptul ca putem face commit pe instanta locala de repository, acest lucru fiind cu atat mai vizibil atunci cand nu avem o conexiune cu serverul pe care se afla repository-ul principal. Un scenariu in care un SCV distribuit si-ar arata avantajele este acela in care nu am avea conexiune la internet si am dori repararea a doua bug-uri care nu au legatura intre ele si am dori sa facem operatia de commit folosind seturi de schimbari diferite. In cazul unui SCV centralizat, acest lucru nu ar fi posibil pentru ca atunci cand am dori sa facem commit ambele reparari de bug-uri ar aparea in acelasi set de schimbari in asteptare.

4. **Distribuire geografica**

In situatia in care s-ar dori colaborarea intre doua echipe de dezvoltare aflate in orase diferite daca am folosi un SCV centralizat ar trebui sa decidem asupra unui orasein care sa se afle serverul central iar accesul la acesta sa se faca folosind o conexiune la internet. In cazul unui SCV distribuit in schimb se poate afla cate un server central in fiecare dintre orase iar acestea pot fi sincronizate folosind operatii de push si pull.

5. **Fluxuri de lucru flexibile**

Distribuirea geografica nu este singurul mod in care flexibiliatea modelului distribuit poate fi utila. Ar putea exista contexte in care dorim ca fiecare instanta de repository sa aibe un anumit scop, sau am dori sa avem ramuri ce gestioneaza simultan lucrul la mai mult de un singur release. In toate aceste cazuri modelul distribuit se dovedeste a fi mult mai flexibil si se poate mula pe modul de lucru al echipei si al companiei, lucru care poate avea benificii considerabile.

6. **Operatii de merge mai usoare**

Operatia de branching sau de creare de ramuri este una usoara, insa nu se poate spune acelasi lucru si despre operatia de merge sau de combinare a ramurilor.

SCV-urile centralizate tind sa nu fie foarte eficiente la operatia de merge, fiind eclipsate de SCV-urile distribuite care au mai multe avantaje :

- Folosesc grafuri orientate aciclice, lucru care ofera algoritmilor de merge informatii utile despre istorie si de stramosi comuni. Grafurile orientate aciclice reprezinta un mod mult mai bun de reprezentare al acestui gen de informatie decat metodele folosite in mod obisnuit de SCV-urile centralizate.
- Pastreaza schimbarile dezvoltatorului separate de operatia de merge necesara realizarii commit-ului. Aceasta abordare este mai putin vulnerabila la erori, avand in vedere ca schimbarile dezvoltatorului sunt intr-un set de schimbari propriu. Astfel singura operatie care mai trebuie facuta este merge-ul propriu-zis si astfel aceasta primeste maximul de atentie.
- Se gestioneaza ramuri ale intregului arbore si nu ramuri de directoare. Calea fisierelor este independenta de ramura pe care se afla.

7. **Backup implicit**

Avand in vedere ca modelul distribuit are mai multe instante ale repository-ului sansa de a pierde date se diminueaza. Acest lucru nu presupune insa suficienta din acest punct de vedere;o strategie de backup tot trebuie sa existe pentru protejarea datelor.

8. **Scalare pe orizontala, nu doar pe verticala**

In cazul unui SCV centralizat, serverul ce tine repository-ul central trebuie sa fie suficient de puternic pentru a-i putea servi pe toti membrii echipei. In cazul unor echipe de 10 oameni, acest lucru nu ar fi un impediment insa in cazul unor echipe mari limitarile hardware pot constitui o problema serioasa de performata.

In contrast SCV-ul distribuit are pretentii mult mai reduse din punct de vedere hardware. Utilizatorii nu interactioneaza cu serverul decat in momentul in care fac operatii de push si pull, in rest majoritatea operatiilor facandu-se pe masinile utilizatorilor. In locul unui server central foarte performant se pot folosi mai multe servere cu performante mai reduse care sunt conectate la serverul central si a caror sincronizare este realizata cu ajutorul unor script-uri.

Cu toate ca modelul distribuit are multe avantaje fata de cel centralizat, acesta poate avea si anumite dezavantaje ce ar putea face ca modelul centralizat sa fie de preferat. Printre acestea se numara :

1. **Zavoarele (Locks)**

In general operatia de lock nu este una foarte utilizata in cadrul lucrului cu un SCV, insa pot exista situatii in care aceasta este necesara iar in acest caz se prefera tool-uri ce folosesc modelul centralizat mai degraba decat cele ce il folosesc pe cel distribuit, datorita faptului ca SCV-urile distribuite nu suporta in general aceasta operatie. Un astfel de scenariu ar fi acela in care o echipa doreste dezvoltarea unui joc ce are o multime de fisiere binare cum ar fi imagini, acestea fiind versionate alaturi de cod. Cea mai buna abordare ar fi in acest caz ca asupra acelor imagini sa se realizeze operatia de lock.

2. **Repository-uri de dimensiuni foarte mari**

Daca avem un repository de un gigaoctet, acesta poate fi tinut fara probleme pe o masina locala insa in situatia in care dimensiunea acestuia ar fi de un teraoctet ar fi prea mult. Cea mai importanta problema este viteza de retea. Daca operatia initiala de clonare implica faptul ca trebuie copiat un teraoctet de informatie pe masina locala se poate estima ca aceata operatie va dura un timp indelungat.

O buna practica in ceea ce priveste folosirea SCV-urilor distribuite este impartirea acestora in repository-uri mai mici. Daca exista insa situatia in care repository-ul are dimensiunea de un teraoctet si nu exista posibilitatea impartirii acestuia in repository-uri mai mici, vor aparea multe probleme iar folosirea unui SCV distribuit nu este fezabila.

3. **Integrare**

La nivel de industrie se doreste o integrare cat mai mare intre controlul versiunilor si lucruri precum, progresul proiectului, wiki-uri, discutii, gestiunea build-urilor, s.a. De obicei operatia de commit a unui dezvoltator nu este singura lui responsabilitate, acesta fiind nevoit sa mai foloseasca alte unelte care sa ajute colaborarea cu ceilalti membrii ai echipei. La modul ideal se doreste o integrare transparenta intre toate aceste unelte folosite de programatori, acest concept fiind cunoscut sub denumirea de ALM sau Application Lifecycle Management.

SCV-urile distribuite au avantajul lucrului offline dar acesta aduce cu sine dezavantajul cresterii complexitatii problemei de integrare cu alte tool-uri. Avand in vedere ca la SCV-urile centralizate stim ca este nevoie de o conexiunea cu serverul la operatiile de commit operatie de integrare cu alte unelte este semnificativ mai usoara.

4. **Stergerea**

Avantajul ca modelul distribuit are back-up implicit datorita faptului ca datele se afla in mai multe locuri are si reversul medaliei in sensul ca acest lucru face ca datele sa fie mai greu de distrus. Unele SCV-uri au posibiliatea alterarii istoriei unui repository, aceasta functie putand fi necesara in unele situatii cum ar fi obligatii legale de a face acest lucru. In astfel de cazuri operatia delete nu este suficienta, datele ramanand in istoria repository-ul iar in cazul SCV-urilor distribuite aceasta sarcina este cu atat mai dificila cu cat exista mai multe copii ale informatiei.

5. **Administrarea**

Un avantaj al modelului centralizat este acela ca furnizeaza un loc centralizat unde se pot face operatii de administrare cum ar fi controlul accesului, permisiunile, si gestiunea conturilor utilizatorilor, s.a. Modelul centralizat este mai slab din acest punct de vedere,

identificarea utilizatorilor fiind facuta in general folosind un sir de caractere care este folosit in jurnalul repository-ului.

6. **Controlul accesului bazat de cai (paths)**

SCV-urile distribuite nu ofera in general un mod de control al accesului la fisiere sau directoare in cadrul arborelui repository-ului, astfel utilizatorii acestor unelte trebuie sa isi aranjeze lucrurile astfel incat controlul accesului bazat pe repository sa fie suficient.

La SCV-urile centralizate majoraritatea operatiilor lucreaza pe o parte din repository iar serverul este implicat in toate incercarile de citire a datelor din repository. Acest lucru face ca accesul controlului sa fie mult mai usor, folosind caile arborelului din repository.

7. **Usurinta de folosire**

Aceasta poate fi considerata o caracteristica subiectiva insa modelul distribuit are anumite aspecte care il poate face greu de folosit. Printre aceste aspecte se numara :

- Faptul ca modelul distribuit este mai greu de inteles conceptual decat modelul centralizat
- Lucrul cu mai multe instante de repository poate crea confuzie
- Faptul ca operatia de commit se poate face local si nu implica in mod obligatoriu serverul central
- Faptul ca in loc de numere de revizie (revision numbers) se folosesc hash-uri criptografice pentru a identifica reviziile.

[1]

V. Bibliografie

[1] - Version Control by Example by Eric Sink

[2] - http://www.catb.org/~esr/writings/version-control/version-control.html

[3] - http://zeroturnaround.com/rebellabs/devprod-report-revisited-version-control-systems-in-2013/

[4] - http://en.wikipedia.org/wiki/Team_foundation_server

[5] - http://en.wikipedia.org/wiki/Git_(software)

[6] - http://en.wikipedia.org/wiki/Mercurial

[7] - http://en.wikipedia.org/wiki/Bazaar_(software)

[8] - http://guides.beanstalkapp.com/version-control/intro-to-version-control.html

[9] - http://en.wikipedia.org/wiki/Distributed_revision_control