Controlul versiunilor

Utilizarea Sistemelor de Operare

Paul Irofti

Universitatea din București
Facultatea de Matematică și Informatică
Department de Informatică
Email: paul.irofti@fmi.unibuc.ro

Motivație

Un scenariu des întâlnit

1. avem un program bun, stabil pe care vrem să-l schimbăm
2. schimbarea dorită se dovedește dificilă sau imposibilă
3. revinirea la vechiul program

Alternative punctul 2

- schimbarea e bună, dorim ambele versiuni ale programului
- schimbarea se poate aplica și altor programe existente

În orice caz este evident că trebuie să salvăm una sau mai multe copii ale programului de-a lungul evoluției sale.

Hello, World!

```
#include <stdio.h>
int main()
{
  printf("Hello, World!\n");
  return 0;
}
```

Vrem să adăugăm funcționalitate nouă: salută o persoană!

Hello, Alex!

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  if (argc != 2) {
     printf("USAGE: %s name\n", argv[0]);
     return -1;
  }
  printf("Hello, %s!\n", argv[1]);
  return 0;
}
```

diff(1)

Compară fișiere și directoare

- `diff -u file1 file2` – arată diferențele între fișierul 1 și 2
- diferențele pot fi prezentate în mai multe feluri
- `-u` – modul uniform, cel mai des întâlnit
- fiecare linie începe fie cu un spațiu, un plus sau un minus
- `+` – linii adăugate
- `-` – linii înlăturate
- spațiu – linii ce conferă contextul schimbării
- dacă în fișier apar mai multe schimbări la distanțe mari (sute de linii) atunci fiecare schimbare este prezentată local
- începutul unei schimbări este marcat de două semne arond `@@`

Exemplu

```
$ diff -u hello.c.orig hello.c
--- hello.c.orig Wed Apr 11 12:31:55 2018
+++ hello.c      Wed Apr 11 12:33:09 2018
@@ -1,6 +1,10 @@
 #include <stdio.h>
-int main()
+int main(int argc, char *argv[])
 {
-  printf("Hello, World!\n");
+  if (argc != 2) {
+      printf("USAGE: %s name\n", argv[0]);
+      return -1;
+  }
+  printf("Hello, %s!\n", argv[1]);
   return 0;
 }
```

patch(1)

Un `patch` este un fișier în care s-au salvat diferențele între versiuna veche și cea nouă a unui program.
Se generează cu ajutorul comenzii `diff(1)`

```
diff -u hello.c.orig hello.c > hello_name.patch
```

Se aplică cu ajutorul comenzii `patch(1)`

```
$ patch -p0 < hello_name.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|--- hello.c.orig Wed Apr 11 12:31:55 2018
|+++ hello.c      Wed Apr 11 12:33:09 2018
--------------------------
Patching file hello.c.orig using Plan A...
Hunk #1 succeeded at 1.
done
```

`-pNUM` specifică câte directoare din cale să fie eliminate

Controlul versiunilor

Revision Control în limbajul de specialitate

- gestionarea mai multor versiuni ale aceluiași fișier
- versiunile se mai numesc și revizii
- util în dezvoltarea programelor
- o revizie poate reprezenta
  - o versiune scoasă pe piață
  - rezolvarea unui defect
  - adăugarea unei noi funcționalități
- avantaje:
  - pași înainte și înapoi prin revizii
  - metoda bisecției pentru a descoperi când a apărut un defect
  - motivarea existenței unui fragment de cod
  - lucrul în echipă
- dezavantaje:
  - necesită meticulozitate și organizare a muncii
  - lucru în plus față de programarea efectivă
  - complexitate sporită

Repository

- conține toate versiunile fișierelor și directoarelor unui proiect
- de obicei centralizat pe un server
- mod de lucru utilizatori (clienți)
  1. conectare și copie locală a *repository*-ului
  2. modifică unul sau mai multe fișiere
  3. transmiterea modificărilor serverului
  4. serverul crează o versiune nouă a proiectului

Checkout

- copie locală a unui repository
- se mai numește și *working-copy*
- crează pe disk ultima versiune a tuturor fișierelor din repository
- poate conține și o bază de date locală cu versiunile anterioare
- o versiune este memorată de obicei prin diferențele față de cea anterioară

Commit

- înregistrează schimbările locale în *repository*
- conține și o descriere a modificărilor făcute
- este foarte important să detaliați cât mai mult
- ajută oamenii cu care colaborați și vă ajută și pe voi din viitor
- formatul des întâlnit este subiect urmat de o descriere de un paragraf

Example

[BUGFIX] Avoid NULL dereference during Emulator destruction.

Found during MemoryUnit!pmap_remove while the Emulator dtor has already executed and the MemoryUnit dtor was in progress.

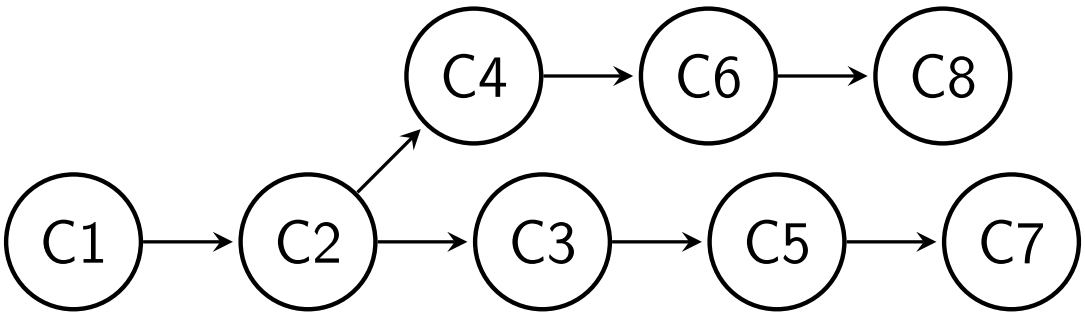
Sample:
tests/libs/pinky/win32api/ntdll/test_RtlGetNtVersionNumbers.exe

Update

- actualizarea copiei locale
- aduce versiunile noi din repository în copia locală
- dacă există modificări locale pot apărea conflicte
- exemplu: doi programatori au modificat aceiași linie dintr-un fișier
- conflictele se rezolvă întâi automat
- dacă procesul eșuează este necesară intervenția utilizatorului
- pot apărea situații când există versiuni locale și versiuni noi în repository, atunci procesul este mai complex (vezi *merge*)

Branch

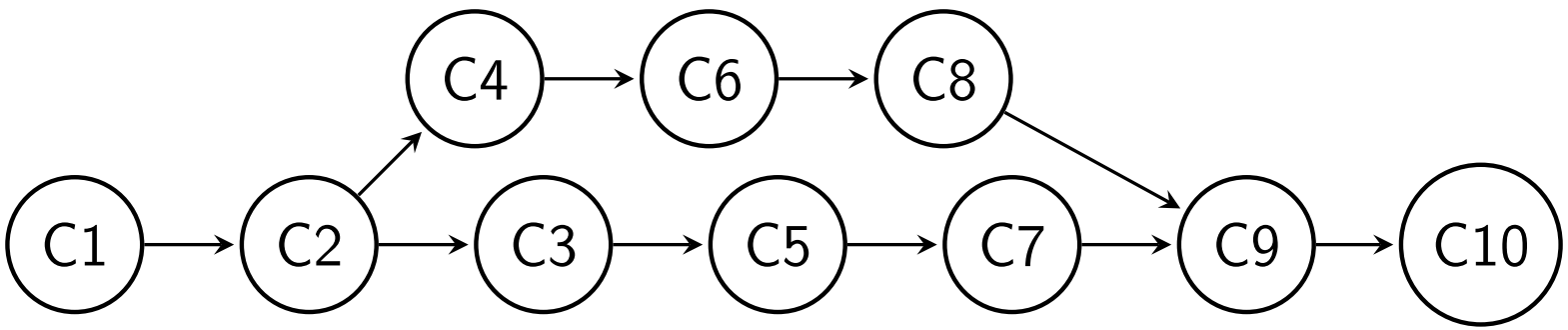
- liniar: fiecare commit depinde de cel precedent
- fie înlănțuirea C1 $\rightarrow$ C2 $\rightarrow$ C3
- un branch apare când cineva poreşte de la C2 și face un commit diferit C4 în continuarea lui C2 nu a lui C3



- branch, bifurcație, ramură
- ramura principală se mai numește *trunk*

Merge

- operația *merge* integrează două sau mai multe schimbări
- acestea pot fi conflictuale sau nu

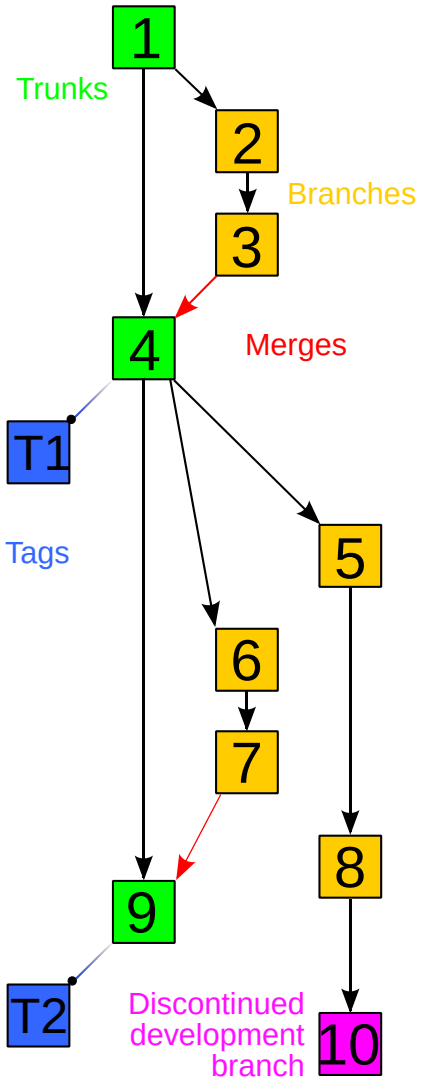


- dacă există conflict între cele două ramuri, C9 este o versiunea ce integrează branch-ul C4 și rezolvă conflictele
- dacă nu există nici un conflict, C9 este pur și simplu o modificare nouă

Tag

- *tag*: etichetă aplicată unui anumit *commit*
- folosită pentru a marca o versiune a produsului
- nomenclatură: alpha, beta, release candidate, release
- alpha – prima etapă înainte de lansarea unui produs, cerc redus (intern) de testare
- beta – a doua etapă îmbunătățită, cerc extins (extern) de testare
- release candidate (rc) – candidat pentru release
- rc1, rc2 – echivalenți alpha, beta
- release – lansarea produsului
- versiunea x.y.z – versiune majoră, minoră, patch (ex. 6.3.12)
- exemple *tag*: v1.1, v2.0alpha, v4.2beta, v5.6rc3

Exemplu istoric proiect



https://en.wikipedia.org/wiki/Version_control

Lucru centralizat

- un singur *repository* pe un server
- fiecare utilizator trebuie să aibă acces
- versiunile anterioare și istoricul sunt păstrate pe server
- local copiem ultima versiune
- este necesar să fim conectați la server pentru a face un *commit*
- exemple: `cvs(1)`, `svn(1)`

Lucru descentralizat

- fiecare utilizator are *repository* propriu
- versiunile anterioare și istoricul sunt păstrate local
- nu este necesară rețeaua
- *commit*-urile se fac local
- sincronizare în rețea prin operații de *push* și *pull* între utilizatori
- *push* – trimite modificările către un alt *repository* (similar unui șir de operații *commit*)
- *pull* – preia modificările de la un alt *repository* (similar operației de *update*)
- cele două operații implică un *merge* și pot necesita intervenții suplimentare din partea utilizatorului pentru a rezolva conflicte
- tendință de centralizare *de-facto*
- exemple: `git(1)`, `hg(1)`