

# CVS 와 Subversion 을 이용한 소스 코드 버전 관리

2003 년 12 월 17 일

방준영

<junyoung@NetBSD.org>

# 차례

- 버전 관리 시스템 개요
- CVS 사용법 둘러보기
- CVS 를 이용한 프로젝트 관리 사례
  - NetBSD 프로젝트
- Subversion 소개
  - CVS 와의 차이점을 중심으로

# 시작하기 전에 ...

“CVS 없이 프로그램 짜는 것은  
낙하산 없이 하늘에서 뛰어내리는 것과 같다.”

-- 브라이언 피츠패트릭

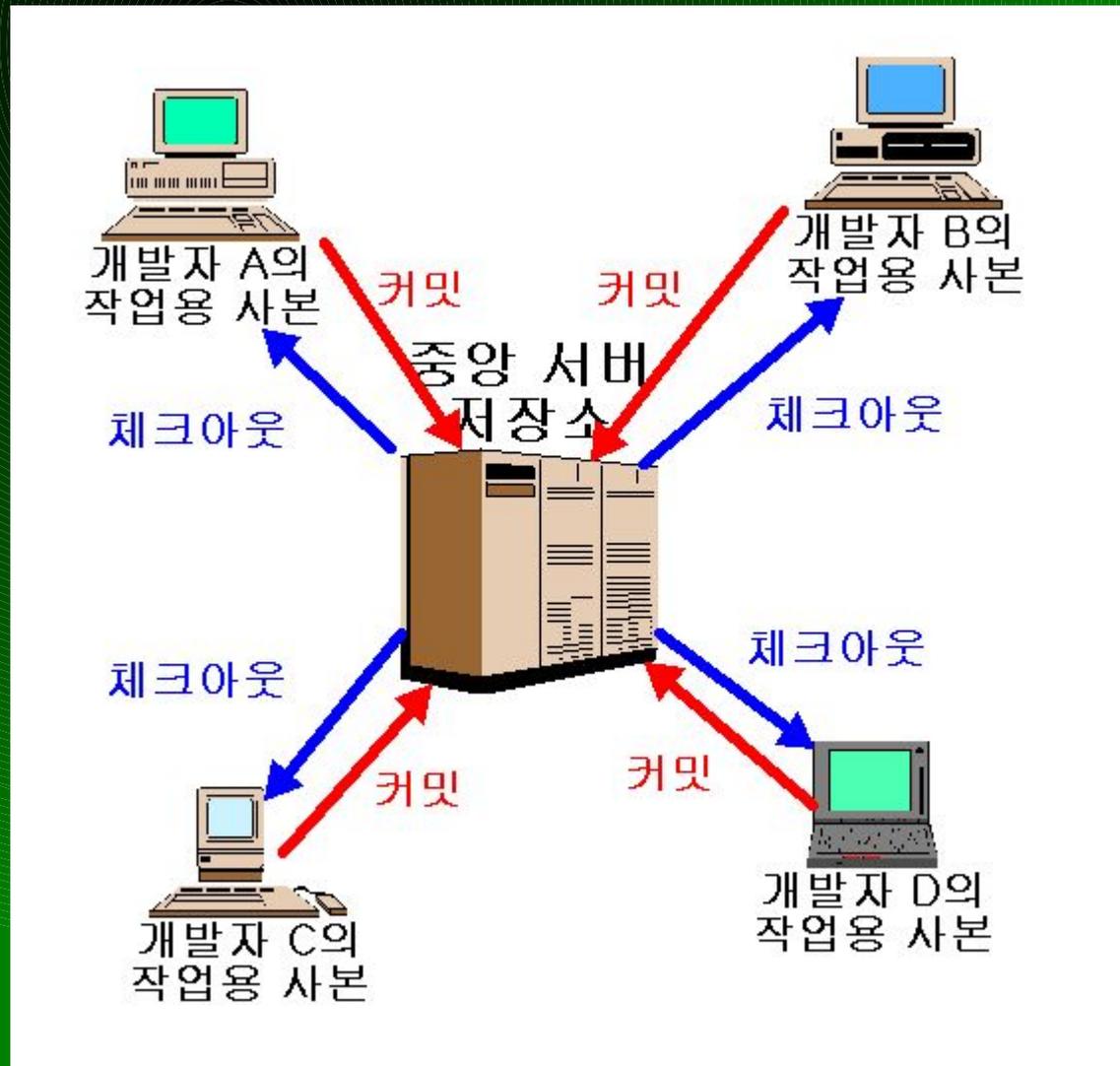
# 버전 관리 시스템의 목적

- 소스 코드의 변경 사항을 보존하기 위해
  - 과거 특정 시점의 소스 파일 및 디렉토리의 내용을 언제든지 손쉽게 확인 가능
  - 버그 및 문제점 추적에 매우 유용
- 협동 작업을 가능케 하기 위해
  - 대부분의 프로젝트는 팀 단위 - 전체 팀원들이 하나의 소스 트리상에서 효율적으로 작업할 수 있도록 지원해 주는 도구 필요
  - 인터넷을 통한 전세계적인 오픈 소스 프로젝트에서는 절대적으로 필수

# 용어

- 저장소 (repository): 프로젝트 중앙 서버에 보관되어 있는 마스터 소스 코드의 데이터베이스
- 작업용 사본 (working copy): 저장소의 소스 코드 트리를 각 개발자의 클라이언트에 내려 받아둔 것
- 체크 아웃 (checkout): 저장소의 소스 트리를 개발자 머신으로 내려받는 것
- 커밋 (commit): 작업용 사본에 가한 변경 사항을 저장소에 적용하는 것
- 리비전 (revision): 갱신 번호

# 저장소와 작업용 사본의 관계



# CVS

- Concurrent Versions System 의 약자
- 이전 RCS 버전 관리 시스템의 확장
  - 여러 명의 개발자가 한 파일에 동시 접근 가능
- 오픈 소스 공동체의 “사실상 표준” 버전 관리 시스템
- 대부분의 유닉스 계열 OS 및 윈도우 상에서 실행 가능
- <http://www.cvshome.org>

# 저장소 생성 및 초기화

- `cvsc init` : 서버에 저장소 생성
  - 저장소에 대한 사용자와 그룹 권한을 먼저 지정
  - 저장소 안에 `CVSROOT` 디렉토리 생성됨
  - 서버에서만 실행 가능 (이후 명령은 각 클라이언트에서 가능)
- 예  
# `cvsc init /cvsroot`

# 소스 트리를 저장소에 읽어 들임

- `cv`s import : 프로젝트를 저장소에 수입
  - 프로젝트와 동일한 디렉토리 트리 생성
  - 각 소스 파일에 대해 ,v 접미사가 붙은 파일 생성
  - 저장소 파일들은 일반 평문이므로 `cat` 이나 `more` 등으로 내부를 볼 수 있음 (편집도 가능)
- 예

```
$ cd myproj ; ls
README subdir-a/ subdir-b/ hello.c
$ cvs import myproj MOGUA myproj-base
```

# 작업용 사본 생성

- `cvs checkout` : 저장소에 있는 프로젝트 소스 트리를 개발자의 머신으로 가져옴
  - 각 디렉토리마다 `CVS` 라는 이름의 서브디렉토리 생성
- ```
$ pwd
/home/junyoung/myproj
$ cd .. ; mv myproj myproj.old
$ cvs checkout myproj
U myproj/README
U myproj/hello.c
...
```

# 변경 사항을 저장소에 반영

- `cv commit` : 각 개발자가 작업용 사본에 행한 변경 사항을 저장소에 반영
  - 파일별로 리비전 끝자리가 .1 씩 증가
  - 한꺼번에 여러 파일과 디렉토리를 커밋하더라도 CVS 에서 커밋 단위는 항상 파일 단위임
- ```
$ cv commit README  
/cvsroot/myproj/README,v <-- README  
new revision: 1.2; previous revision:  
1.1  
done
```

# 기타 CVS 명령

- `cv update`
  - 다른 개발자들이 저장소에 적용한 변경 사항을 내 작업용 사본에 반영
  - 작업용 사본을 과거 임의의 순간으로 되돌릴 때
  - 과거 변경 사항을 철회 (철회후 커밋 필요)
- `cv diff` : 파일별 차이점을 비교
- `cv log` : 파일의 커밋 로그 조회
- `cv status` : 파일의 현재 상태 조회

# 충돌 문제

- 때때로 개발자들간에 같은 파일을 동시 변경하는 일이 생길 수 있음
- 충돌 (conflict) : 같은 파일의 같은 부분을 두명 이상이 다른 형태로 변경한 뒤 커밋할 때 발생
  - 같은 파일이라도 변경 부분이 겹치지 않는다면 CVS는 이를 충돌로 간주하지 않음 (그러나 잠재적 위험 존재)
  - 충돌을 최소화하려면 평소 `cv update` 명령으로 작업용 사본을 최신 상태로 유지하는 것이 중요
  - 개발자들간의 충분한 대화도 필수!

# 태그

- 소스 트리의 특정 순간을 표시
- 태그 붙이기 :  
`cvstag version-1-5-1`
- 과거 특정 태그 지점으로 트리를 갱신 :  
`cvsupdate -r version-1-5-1`
- CVS 에서 태깅은 매우 시간이 오래 걸리는 작업
  - 프로젝트 크기가 커질 수록 비용이 급속도로 커짐

# 브랜치

- 프로젝트를 여러 개의 서브프로젝트로 분기하여 동시 개발
- 한 브랜치에 적용한 변경 사항은 다른 브랜치에 영향을 끼치지 않음 . 따라서 수동으로 변경 사항을 브랜치간에 적절히 반영해야 함
- 브랜치 생성 :  
`cvs tag -b myproj-newgui`
- 개발을 완료한 브랜치는 메인 브랜치로 병합
- 태그와 마찬가지로 매우 “비싼“ 작업

# NetBSD 프로젝트

- 250 명 이상의 대규모 개발자 그룹
- 보안 대책
  - CVS 저장소 서버를 메일 , FTP 서버와 분리 운영
  - CVS 서버는 소수 운영진만이 로그인 가능
  - 그외 모든 서버는 SSHv2 로만 로그인 가능
    - SSHv1, telnet 및 비밀번호를 통한 접속 불가
- 수 기가 바이트에 달하는 저장소 크기
  - 서버 사양 : 듀얼 펜티엄 4 2.4GHz, 4GB 메모리 , 수십 GB 이상의 UltraSCSI RAID

# 브랜치 구성

- 브랜치를 하는 경우
  - 각 릴리즈별 : -current ( 트렁크 ), netbsd-1-5, netbsd-1-6
  - 소스 트리의 대규모 변경이 필요한 서브프로젝트 : nathanw\_sa, kqueue 등
- 브랜치별 담당 개발자 존재
  - 담당 개발자의 허락없이 브랜치에 커밋은 불가능
  - 릴리즈 브랜치는 릴리즈 엔지니어링팀만이 커밋 가능
    - 풀업 (pullup) 요청 시스템 이용

# 태그의 사용

- 태그를 붙이는 경우
  - 릴리즈 지점 : netbsd-1-6-RELEASE, netbsd-1-6-PATCH1, netbsd-1-6-PATCH2-RC3 등
  - 브랜치 병합의 이전과 이후 : nathanw\_sa\_before\_merge, nathanw\_sa\_end 등

# 주기적 빌드

- 주기적 빌드를 통한 소스 코드 검증
  - 매일 40 여개의 기종에 대한 배포판을 i386 기종 상에서 크로스 빌드
  - 빌드 성공 / 실패 여부를 통해 문제점을 빠르게 수정 가능

# 발생한 문제

- 주관 개발자의 허락없이 임의의 소스 변경
  - 종종 격렬한 “플레임” ( 열전 ) 야기
  - 논란이 많은 변경은 코어 그룹이 최종 결정
- 저장소 서버 과부하
  - 고사양 하드웨어로 교체후 해결
- 파일 / 디렉토리 이동 및 이름 변경
- 빌드 과정중 문제

# CVS 의 문제점 (1)

- 체인지셋 개념 없음
  - 커밋은 항상 파일 단위로만 이루어짐
    - 원자적 커밋 보장되지 않음
  - 변경 사항을 추적할 때 매우 불편
- 파일 이동, 복사, 권한 변경 기능 없음
  - “저장소 복사 (repo copy)”와 같은 편법 고안
- 태그와 브랜치가 매우 (!) 비효율적
  - 비용이 프로젝트의 크기에 비례

## CVS 의 문제점 (2)

- 비효율적인 네트워크 대역폭 활용
  - 차이점을 오직 서버에서 클라이언트로만 전송
  - cvs update 명령도 프로젝트 크기에 비례
- 대소문자 비구별 파일시스템과 충돌
  - cvs 라는 이름의 파일이나 디렉토리 생성 불가
- 바이너리 파일 지원 미비
  - 리비전별 차이점 비교 불가

# Subversion

- 전 CVS 개발자들이 주축
  - CVS 대체가 기본 목표
- 현재 버전 0.34 (“영점 삼십 사”)
  - 베타 릴리즈 이전 단계
  - 최종 버전 1.0 은 2004 년초로 예상
  - 상당히 안정적으로 동작 : 프로젝트 자체도 Subversion 저장소 하에 관리
- 웹사이트 : [subversion.tigris.org](http://subversion.tigris.org)
- BSD 라이선스

# Subversion 의 특징 (1)

- 대부분의 명령이 CVS 와 유사
  - 기존 CVS 사용자들의 전환 부담 최소화
  - 더 일관적이고 편리해진 명령별 옵션
- 가상 파일시스템 개념의 저장소
  - 인터페이스와 저장소를 계층으로 분리
    - 다양한 프로토콜과 저장소 포맷 지원 가능
  - WebDAV 프로토콜 지원 (아파치 2 기반)
  - 저장소 포맷으로 현재 버클리 DB 사용

## Subversion 의 특징 (2)

- 디렉토리 , 파일 이동 , 복사 , 속성 지원
- 커밋 단위가 체인지셋
  - 리비전은 파일별이 아닌 커밋별로 증가
  - 함께 커밋하는 파일 , 디렉토리가 서로 연관성을 가짐
  - 원자적 커밋
- 효율적인 네트워크 대역폭 활용
  - 서버 - 클라이언트간 양방향 데이터 전송
  - 현재 리비전을 로컬 파일시스템에 저장

# Subversion 의 특징 (3)

- 저렴한 태깅 / 브랜칭 비용
  - 프로젝트 크기가 아닌 데이터 크기에 비례
  - CVS 와 달리 태깅 / 브랜칭이 “권장 사항”임
- 대소문자 비구별 파일시스템에서도 덜 위험
  - 관리 디렉토리명 앞에 '.' 부가 (.svn)
- 바이너리 파일 diff 지원
- 편리해진 환경 설정
  - 커밋 로그 통지, 접근 제어 리스트 등 기본 제공

# Subversion 저장소

- 저장소 포맷은 더이상 평문 파일이 아님
  - 데이터베이스를 관리하고 정보를 조회하기 위한 별도의 명령 필요 : svnadmin, svnlook 등
  - 손상, 변경 여부를 쉽게 알 수 있음
- 파일시스템 활용 측면에서 유리
  - CVS 보다 파일시스템 공간 및 메타데이터 낭비가 훨씬 적음
- 단점
  - 구조 변경에 따라 종종 데이터베이스 덤프 / 로드 과정 필요

# 저장소 접근 방식의 종류

- file:// : 로컬 파일시스템상의 저장소에 접근
- http:// : Subversion 을 지원하는 아파치 서버에 WebDAV 프로토콜을 통해 접근
- https:// : http:// 와 같으나 SSL 암호화
- svn:// : 비인증 접근 방식
- svn+ssh:// : ssh 로 암호화한 인증 접근

# 달라진 리비전 개념

- 리비전 번호는 0 부터 시작하는 정수
  - 예 : -r6458
- 파일별 리비전은 더 이상 존재하지 않음
- 장점
  - 변경 사항 철회가 매우 용이
  - 문제점 추적이 매우 용이
- 단점
  - 같은 저장소내 프로젝트들이 리비전 번호 공유
  - 파일별 리비전이 편리한 경우가 있음

# 달라진 태깅과 브랜칭 개념

- 태깅 / 브랜칭은 단순히 디렉토리 복사에 불과
  - 복사는 파일시스템 차원이 아닌 데이터베이스 차원에서 이루어지므로 디스크 공간 낭비 없음
  - 필요하다면 언제든지 활용 권장
- 태깅과 브랜칭도 각각 리비전 증가로 간주
- 공식적으로 권장하는 방법
  - 트렁크를 trunk/ 에 , 브랜치를 branches/ 에 , 태그를 tags/ 밑에 둠

# Subversion 프로젝트 트리 예

- /repos/mogua/
  - trunk/
  - branches/
    - newheapapi/
  - tags/
    - build-0/
    - build-1/
  - thirdparties/
    - MINGW/
      - mingw-runtime/
      - mingw-w32api/

# Subversion 명령 예

- `svnadmin` : 저장소 관리
  - `$ svnadmin create /repos/myproj`
- `svnlook` : 저장소 내용 조회
  - `$ svnlook tree /repos/myproj`
- `svn checkout` : 작업용 사본 생성
  - `$ svn checkout svn+ssh://beans/repos/mogua`
- `svn commit` : 변경 사항을 저장소에 반영
- `svn update` : 소스 트리 갱신

# Subversion 의 사용

- 각 운영체제별 패키지로 설치 권장
  - 리눅스 , FreeBSD, NetBSD, 윈도 등
- svn+ssh:// 프로토콜 사용 권장
- GUI 프론트엔드
  - RapidSVN (X11/ 윈도 )
  - TortoiseSVN ( 윈도 )
- 자세한 사용법은  
<http://svnbook.red-bean.com> 참고

# 어느 것을 써야 하나 ?

- CVS 가 더 편리한 경우
  - 연관성이 크지 않은 파일들을 통합 관리할 때
    - 각종 문서 등
  - 혼자 관리하는 아주 작고 간단한 프로젝트
- Subversion 이 더 편리한 경우
  - 그외 모든 경우 :-)

# 버전 관리 시스템이 해결하지 못하는 것

- 프로젝트 관리자가 없거나, 없느니만 못함
- 팀원들간 임의 커밋이 관례화
- 주기적 빌드 관리 부재
- 커밋 로그 메시지를 기록하지 않음
- 틀린 문법으로 쓴 영어 로그
  - 한글 사용 권장!
- 커밋 통지 메일링 리스트 운영하지 않음
  - CVS 의 경우 절대적으로 필수!