제품소프트웨어 패키징
(Git과 GitHub를 활용한 소스코드 관리)

다양한 소스코드 관리 시스템

학습내용

- 소스코드 관리 시스템의 구분
- 소스코드 관리 시스템의 종류 및 특징
- 통합 프로젝트 관리 툴 알아보기

학습목표

- 소스코드 관리 시스템의 종류 4가지를 말할 수 있다.
- 다양한 소스코드 관리 시스템의 특징을 알고, 자신의 환경에 맞는 소스코드 관리 시스템을 선정할 수 있다.
- 통합 프로젝트 관리 툴의 종류를 말할 수 있다.

소스코드 관리 시스템의 구분

1. 소스코드 버전 관리 시스템

1) 기존 기능들을 활용한 단순한 방법

- 문서를 만들 때 다른 이름으로 파일을 저장함
- 문서를 만들어서 변경점이 있을 때 마다 상세히 기록함
- 하나의 PC를 원격 드라이브로 지정 후, 매일 압축하여 해당 폴더에 업로드하고 저장함
- 공유 드라이브에서 파일을 불러와서 작업하고 바로 저장
- CD등의 데이터 장치에 매일 저장
- 웹하드에 주기적으로 저장

2) 소스코드 관리 시스템을 사용하는 방법

- 단순히 파일을 저장해 주는 툴
  - 특정 폴더를 생성 시 압축파일을 쌓아두는 등의 업무 자동화
  - 새로운 파일을 나누는데 필요한 노력 감소, 업무 효율 향상
- 파일의 변경 사항마다 별도의 메모를 할 수 있도록 지원해 주는 툴
  - 새로운 파일이나 변경된 파일을 저장할 때 마다 간략하게 메모 추가 가능
  - 파일을 변경할 때의 상황이나 목적 등 다양한 정보를 메모할 수 있음
- 서버에서만 동작하는 툴
  - 공용 저장공간을 만들고 서로 업로드하여 관리하는 방법
  - 서버 접속이 불가능할 수 있음
- 서버와 로컬에서 모두 동작하는 툴
  - 로컬에서 작업을 하고, 나중에 서버와 연결하여 통합하는 방법
- 저장 형태에 따른 분류
  - 로컬형태 : 순수 내 컴퓨터에서만 저장하고 소스코드를 관리하는 형태
  - 중앙 집중식 형태 : 오로지 서버에서만 소스코드를 관리하여 작업 중에는 항상 서버와 접속하여 실시간으로 소스코드를 관리하도록 하는 형태
  - 분산형태 : 로컬형태와 중앙 집중식 형태의 장점을 결합하여 로컬에서 작업하다가 서버와 동기화 하는 형태

2. 로컬 버전 관리

1) 로컬 버전 관리 시스템(VCS)란?

- VCS(Version Control System) : 본인의 PC에서 직접 소스코드의 변경 사항을 저장하고 관리할 수 있는 방법
- 로컬 버전 관리의 특징
  - 소스코드 관리 시스템의 가장 초기에 만들어진 방법
  - 파일의 정보를 시간의 흐름에 따라 차곡차곡 쌓아서 보관하는 방법을 자동화 한 형태
  - 주로 파일이 수정되는 시점을 기록하고 관리하는 기능 위주로 구현됨

2) 로컬 버전 관리 시스템의 대표적인 도구 RCS

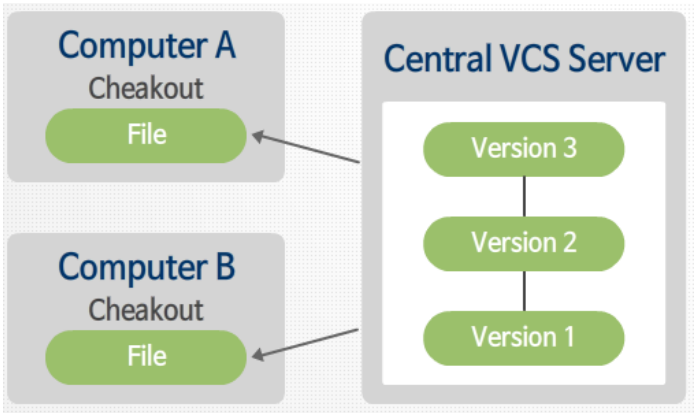
- 새로운 파일이나 Patch Set이라는 단위로 파일의 변경점을 관리하여 Patch Set 단위로 모든 변경 사항을 저장하는 방법
  - 파일이 수정 완료된 시점에 간략한 정보가 저장되어, 향후 소스코드 관리 시 정보를 제공함
  - 책을 읽다가 중지하거나 중요한 내용에 꽂아 놓는 책갈피와 같은 기능을 자동화 한 소스코드 관리 시스템
- **RCS Patch Set의 장점**
  - Patch Set 단위로 파일이 변경된 것을 되돌리는 작업을 수행할 수 있음
- **RCS Patch Set의 단점**
  - 본인 PC에서만 동작하므로 하드웨어 손상 시 위험성이 있음
  - 다른 사람과 협업에는 사용하기 힘듦

3. 중앙 집중식 버전 관리

1) 중앙 집중식 버전 관리(CVCS)란?

- CVCS(Central Version Control System) : 중앙에서 관리하는 버전 관리 시스템

  - 여러 명의 개발자가 동시에 같은 소프트웨어를 개발할 때 유용하게 사용되는 방법



  - 다수의 개발자가 소스코드를 공유하여 변경사항을 관리할 수 있음

  - 혼자 사용할 경우에도 서버에 저장하게 되므로 백업을 하는 역할로도 사용 가능

- 장점

  - 누구나 다른 사람이 무엇을 하는지 알 수 있음

  - 관리자는 각 개발자의 작업 내용을 꼼꼼히 체크 → 대형 프로젝트일수록 관리가 편함

  - 항상 최신의 소스코드를 함께 공유하여 개발 수행 → 다른 기능들과의 연동을 위해 각 개발자의 로컬 소스를 전달받아 수정할 필요가 없음

- 단점

  - 중앙 서버가 잘못되면 모든 작업이 중단되며 유실됨

  - 서버가 복구될 때 까지 모든 개발 작업은 중지됨

  - 서버가 손실되면 각자 작업 중이던 내용 외에는 별도로 관리된 내용이 없음

4. 분산 버전 관리

1) 분산 버전 관리(DVCS)란?

- DVCS(Distributed Version Control System) : 중앙 집중식 버전 관리를 개선한 방법
- 분산 버전 관리의 특징
  - 저장소의 마지막 수정사항만 가져오는 것이 아닌, 해당 저장소를 통째로 복사해 옴
  - 서버가 손실되어도 각 저장소의 복제된 부분만 가져오면 서버가 복구됨
  - 다수의 연결 구조로 인해, 다수의 원격 저장소를 활용, 동시에 여러 그룹과 작업이 가능

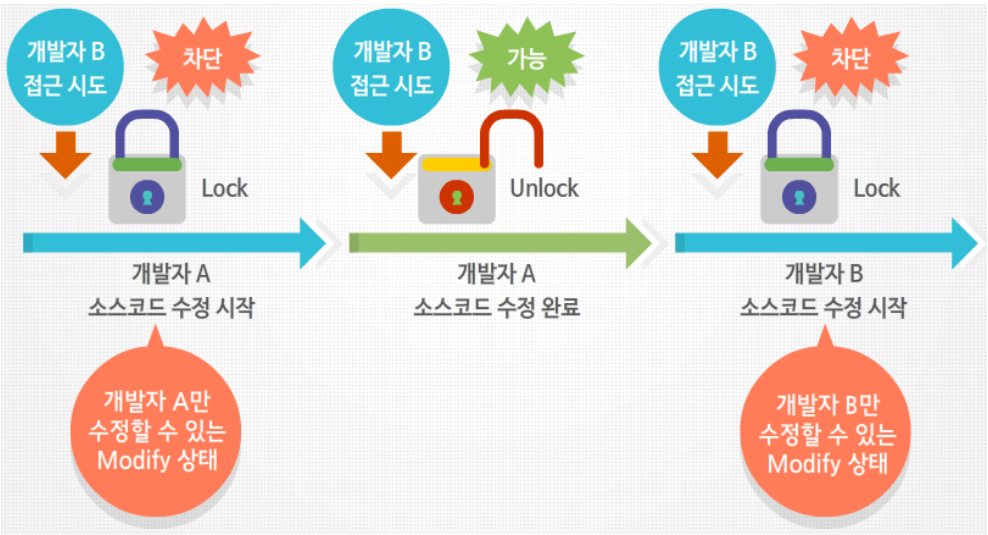
소스코드 관리 시스템의 종류 및 특징

1. Source safe

1) Source safe란?

- Lock - Modify - Unlock의 순서로 수행되는 방식
- 처음에 소스코드 수정을 시작하면 해당 파일이 잠겨져 다른 사람이 수정할 수 없고, 수정이 완료되면 Unlock되어 다른 사람이 컨트롤 할 수 있는 상태로 만들어 줌

2) Source safe의 코드 관리 방법



3) Source safe의 특징

- 로컬 방식의 대표적인 프로그램
- 여러 사람이 동시에 작업하기에는 어려움이 많음
- 혼자 개발하는 프로젝트에 가장 적합한 방식
- 마이크로 소프트 계열의 소프트웨어 개발 시에 가장 편리하게 사용

2. CVS

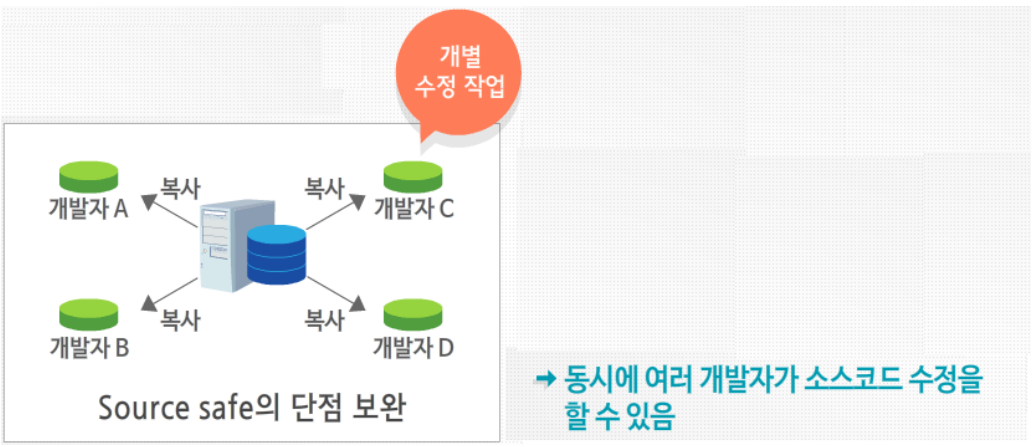
1) CVS란?

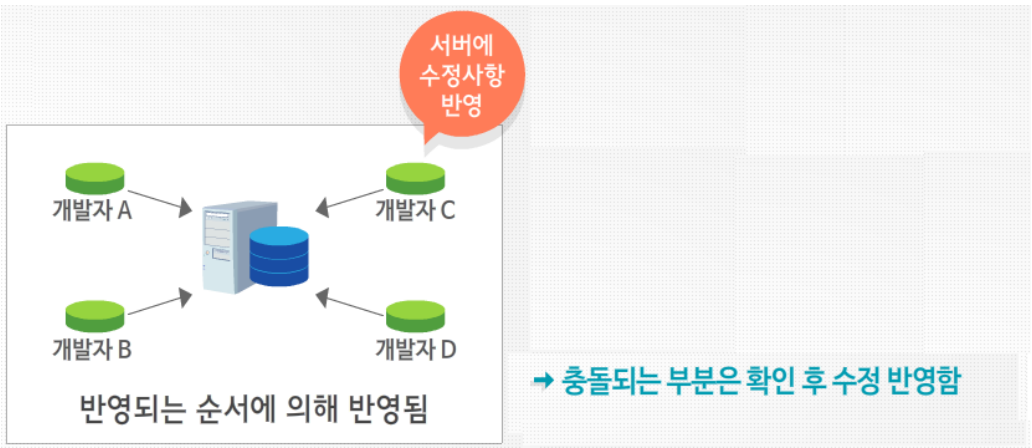
- 오픈소스 프로젝트에서 많이 활용되는 소스코드 관리 시스템
- 서버에 저장 및 수정 후에 반영하는 중앙 집중식의 시스템

2) CVS 장점

- 서버에서 소스를 복사 후 수정하고 다시 서버에 반영하므로, 하나의 파일을 수정하고 있어도 다른 개발자가 접근할 수 있음
- 파일 단위로 수정 이력을 관리할 수 있어 좀더 세분화된 관리가 가능함

3) CVS의 코드 관리 방법





2. CVS

4) CVS 단점

- 디렉터리와 파일 이름을 변경하는 것을 지원하지 않아, 변경되는 것을 관리할 수 없음
- 오로지 파일 내용의 변화만 관리가 가능함
- 네트워크가 끊어지거나 서버에 문제가 발생하면 소프트웨어 개발 프로젝트가 중지될 수 있음
  - 단순한 사용 방법과 서버에 저장을 하여 협업이 가능하므로 오픈소스에서 많이 활용되고 있음
  - 현재는 단점을 보완한 다양한 형태의 프로그램이 개발되어 진화하고 있음

3. SVN

1) SVN이란?

- CVS의 단점을 수정하기 위해 CVS를 개발한 개발자들이 참여해 만든 시스템
- CVS가 버전업 된 것

2) SVN특징

- Copy-Modify-Merge 방식
- Revision 단위의 이력 관리
  - 파일과 디렉터리 이름이 변경되는 것도 관리가 가능함
  - 저장소 내 일부 디렉토리 경로만 골라서 체크할 수 있음
- 변경 정보도 항상 로컬 컴퓨터에 저장하기 때문에 네트워크가 연결되지 않아도 소스코드의 모든 변경 정보를 확인할 수 있음

3) CVS와 SVN의 공통 단점

- 네트워크가 끊어지면 실제 수정 및 반영을 할 수 있는 작업은 불가능한 중앙 집중식의 형태를 활용함
  - 네트워크가 끊어진 상황에서 소스코드 작업을 계속 수행하더라도 변경점을 저장하고 반영할 수 없음
- 서버가 연결되지 않는 환경에서는 사용하기 힘든 단점이 있음
  - 네트워크가 끊어진 상황에서 소스코드 작업을 계속 수행하더라도 변경점을 저장하고 반영할 수 없음

4. Mercurial

1) Mercurial이란?

- 로컬과 중앙 집중식 방식의 단점을 보완한 분산 방식의 소스코드 관리 시스템
- CVS와 SVN의 가장 큰 문제점(네트워크가 끊겼을 때 작업이 불가능한 것)을 해결한 시스템

2) Mercurial 특징

- 로컬 PC에 저장소를 두고 오프라인에서 작업하고 이력관리를 할 수 있음
  - 그 후에 서버에 반영하여 다른 소스들과 변경점을 적용하면 됨

3) Mercuria 장점

- 규모가 큰 프로젝트에서도 적용이 가능함
- 일시적으로 서버에 문제가 생겨도 모든 개발이 중지되지는 않음

4) Mercuria 단점

- 규모가 큰 프로젝트에서 처음 소스코드를 가져올 때는 많은 시간이 소요됨
- 여러 군데에 복사를 하게 되면 그만큼의 용량을 차지함

5. Git

1) Git란?

- Mercurial과 같은 분산 방식으로 실제 동작 방식에는 큰 차이가 없음
- 리누스 토발즈라는 리눅스 개발자가 리눅스 개발 관리를 위해 만든 소스코드 관리 시스템
- Mercurial과 비슷하고, 좀 더 많은 기능 지원으로 인해 사용에는 복잡한 부분이 있음
- 대형 프로젝트에서의 효과적인 관리를 위해서 가장 많이 사용되고 있는 소스코드 관리 시스템

2) Git의 핵심 목표

- 빠른 속도와 단순한 구조
- 비선형적인 개발과 완벽한 분산을 통해 대형 프로젝트에서 유용하게 사용하는 것

통합 프로젝트 관리 툴 알아보기

1. JIRA

1) JIRA란?

- Atlassian 에서 개발한 통합 프로젝트 관리 툴
- 유료 서비스 - 강력한 기능과 고객 지원으로 상업 소프트웨어를 개발하는 회사에서 많이 활용
- Java(자바) 언어로 개발
- 2003년도에 처음으로 런칭
- 실제 시스템을 도입하기 위한 비용 등의 정보는 공식 사이트에 접속하여 확인
- 대형 상업 프로젝트를 추진하며 예산에 부담이 없다면 가장 편리하게 사용
- 공식 사이트 : https://www.atlassian.com/software/jira

2) JIRA의 특징

- JQL(JIRA Query Language)로 SQL 쿼리문과 비슷하게 검색 - 확장성이 높고 쉽게 사용 가능
- 기능이 많아 UI가 복잡함
- WYSWYG editor가 강력 - 문서 생성 시 다양한 표와 글자형태, 이모티콘 등도 표현 가능
- 하나의 issue를 다른 project로 이동할 수 있음
- JIRA가 업데이트 될 때마다 플러그인 검증이 잘 되어 오류가 적고, 고객지원이 됨
- 코드리뷰를 위한 툴을 설치하면 연동 가능
- 애자일 기능 연동 가능 - 애자일 방식에 맞게 프로젝트 관리

2. Redmine

1) Redmine란?

- 무료로 사용 가능
- 오픈소스로 발표됨
- Ruby on Rails 언어로 개발
- 2006년에 발표함
- 공식 사이트 : http://www.redmine.org

2) Redmine 기능 및 장단점

- 다른 오픈소스 프로젝트의 모습이나 기능을 그대로 가져와 사용한 것이 많음
- 오픈소스 프로젝트에 많이 참여해본 사람
  - Redmine의 화면 구성이나 기능들이 매우 친숙하게 다가옴
- 오픈소스 프로젝트를 많이 접하지 않은 사람
  - 약간 불편하게 느껴짐
- 프로젝트별 Wiki 게시판 - 구성원간 자유로운 토론과 정보 축적, 프로젝트 내 소규모 백과사전 구성
- 검색방식 - 15가지 검색 필드로 검색 가능, JIRA에 비해 검색기능 약함
- Editor는 심플한 기본 형식, 문서 내용을 꾸미는 데 한계
- 프로젝트를 트리 구조와 같이 서브 프로젝트를 계속 생성 및 관리가 편함
- 프로젝트간의 Issue 이동은 불가, 서브 프로젝트 사이에는 이동 가능
- 플러그인을 통해 기능 확장은 가능하나 오류 발생 가능성이 높음
  - 사용자들은 검증이 된 것을 위주로 사용
  - 상황에 따라서는 수정하거나 만들어서 사용할 수 있는 확장성이 있음

3. Trello

1) Trello 장단점

- Fog Creek Software에서 출시한 프로젝트 협업 툴
- 화이트보드에 포스트잇을 붙이듯 심플하고 간단 - 사용방법이 쉽고 간편
- 심플한 화면 - 프로젝트 현황 파악이 아주 쉽고 직관적으로 모니터링 가능
- 라벨 기능 - 쉽게 분류
- 카드 형태로 이슈를 쉽게 복사하거나 이동 가능
- 온라인 호스팅 서비스 지원 - 가장 친근하고 쉽게 접근하여 프로젝트 관리
- 카드가 많아지게 되면 관리가 어려움
- 너무 단순한 특성으로 인해 큰 프로젝트나 복잡한 구조에서는 활용하기 쉽지 않음

2) Trello 특징

- 크지 않은 프로젝트에서 구성원간의 커뮤니케이션과 현황 파악을 위해 간단하게 사용하기에 적합
- 사이트 : http://trello.com

4. Asana

1) Asana란?

- 대쉬보드 형태의 실시간 확인 툴
- 팀원들간에 대쉬보드를 공유하기 때문에 이메일을 대체할 수 있음
- 깔끔한 UI로 직관적이고 현황 파악이 쉬움
  - 복잡하거나 크지 않은 프로젝트에서 팀원들 간의 소통을 위해 사용하기 매우 좋고 친숙한 툴
- 리스트형식이므로 일정이 많아져도 관리가 쉬움
- 직관적이지 못함
- 공식 사이트 : http://asana.com

핵심정리

소스코드 관리 시스템의 구분

1. 소스코드 관리 시스템의 구분

- 수동으로 하던 소스코드 백업 기능을 좀더 간편하고 똑똑하게 관리해 줌
- 단순히 파일을 저장해 주는 툴
- 파일의 변경 사항마다 별도의 메모를 할 수 있도록 지원하는 툴
- 서버에서만 동작하는 툴과 서버와 로컬에서 모두 동작이 되는 툴

2. 로컬 버전 관리

- 단순 파일 복사 형태의 기능을 자동화 한 버전 관리 시스템

3. 중앙 집중식 버전 관리

- 여러 명의 개발자가 동시에 같은 소프트웨어를 개발할 때 유용하게 사용됨

4. 분산 버전 관리

- 저장소의 마지막 수정사항만 가져오는 것이 아닌, 해당 저장소를 통째로 복사해 옴

핵심정리

소스코드 관리 시스템의 종류 및 특징

1. Source safe - 로컬 버전 관리
2. CVS - 중앙 집중식 버전 관리
3. SVN - 중앙 집중식 버전 관리
4. Mercurial - 분산 버전 관리
5. Git - 분산 버전 관리

핵심정리

통합 프로젝트 관리 툴 알아보기

1. JIRA

- Atlassian 에서 개발한 통합 프로젝트 관리 툴
- 2003년 Java 언어로 개발된 유료 서비스

2. Redmine

- 2006년 Ruby on Rails 언어로 개발된 무료 서비스

3. Trello

- Fog Creek Software에서 출시한 프로젝트 협업 툴

4. Asana

- 대쉬보드 형태의 실시간 확인 툴