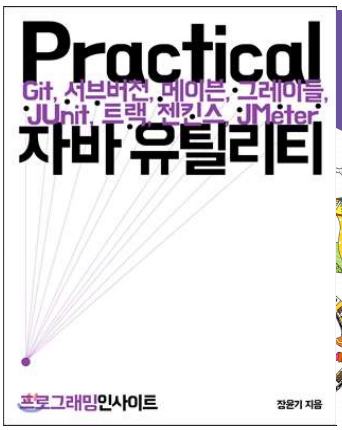
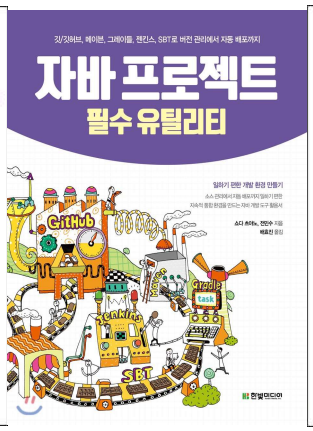
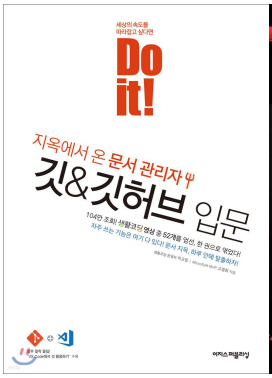
Git 소개

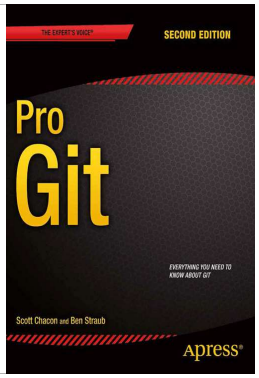
Refer to :

1. https://backlog.com/git-tutorial/kr/
2. https://git-scm.com/book/ko/v2
3. https://docs.github.com/en/get-started
4. https://learngitbranching.js.org/?Locale=ko
5. https://git-school.github.io/visualizing-git/
6. https://tacademy.skplanet.com/live/player/onlineLectureDetail.action?seq=171
7. https://www.inflearn.com/course/%EC%A7%80%EC%98%A5%EC%97%90%EC%84%9C-%EC%98%A8-git
8. https://www.inflearn.com/course/git-2
9. https://olc.kr/course/course_online_view.jsp?id=10160









1. Git History
    1) Linux Kernel project 를 위한 Version 관리 System 으로 개발
    2) Birth year : 2005 년
    3) Linus Benedict Torvalds 를 포함한 Linux 개발 community 에서 상용 도구인 Bitkeeper 의 대용으로 만듦.
    4) A Short History of Git
        - https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git

2. Git 과 SVN 의 trend 비교
    - https://trends.google.com/trends/explore?geo=US&q=git,svn

3. Version 관리 System 이 필요한 이유
    1) File 의 Version 관리를 어떻게 하는가?
    2) 가장 간단한 방법은 파일을 미리 복사해두는 것이다.
    3) 예를 들면, File 과 Folder 명 뒤에 편집한 날짜를 붙여주는 방식같은 것이다.
    4) 하지만 File 을 편집할 때마다 매번 복사하는 일은 번거롭기도 하고 실수할 가능성도 많다.
    5) 또한 특별한 규칙 없이 마음대로 이름을 붙이는 경우 어느 File 이 최신인지, 또 File 의 어떤 부분이 변경된 것인지 파악하기 어렵다.
    6) 바로 이런 문제를 해결하기 위해 만들어진 것이 Version 관리 System(VMS)이다.
    7) Version 관리 System 을 사용하면 개별 File 이나, project 를 이전 상태로 되돌릴 수 있다.
    8) 시간에 따른 수정 내용을 비교할 수도 있고, 공동 작업을 하는 경우에는 누가 문제를 일으켰는지도 알 수 있다.

4. Version 관리 System 의 유형
    1) Local Version Control System
        - User 의 Local 환경에서 Version 관리를 한다.
        - 일반적으로 날짜별로 File 을 구분한다.

    2) Centralized Version Control System, CVCS
        - 별도의 Version 관리 Server 를 두고 모든 사용자가 접근하는 방식
        - 공동작업을 하는 경우 Local Version 관리 System 으로는 협업이 원활한 대응이 힘들기 때문에 개발

- 중앙 server 에 문제가 발생했을 때 Version 관리를 계속할 수 없다는 단점이 있다.
- CVS, SVN, Perforce

3) Distributed VCS, DVCS
   - 위의 CVCS 의 단점을 극복하는 방식
   - Client 는 File 의 마지막 Snapshot 를 Check 하는 것으로 끝나지 않고, 저장소 전부를 복제하여 Local 저장소에 관리한다.
   - 그럼 중앙의 Version 관리 System 에 문제가 발생하더라도 Local 저장소에서 복구할 수 있다.
   - Git, Mercurial, Bazaar, Darcs

5. Git?
   1) Source code 를 보다 효과적으로 효율적으로 관리하기 위해 개발된 분산형 Version 관리 System 이다.
   2) Git 에서는 Source code 가 변경된 이력을 쉽게 확인할 수 있다.
   3) 특정 시점에 저장된 Version 과 비교하거나 특정 시점으로 되돌아갈 수도 있다.
   4) 또 내가 올리려는 File 이 누군가 편집한 내용과 충돌한다면, Server 에 Upload 할 때 경고 메시지가 발생한다.
   5) 협업관리
   6) 효율적으로 백업하기

6. Version 이 만들어지는 단계 - Version 이 되기까지 거쳐가는 세 개의 공간
   1) Working Directory(작업공간)
      - 내가 코드 작업을 하는 공간
      - File 들이 생성/수정/삭제되는 공간
      - 즉, 변경사항이 생기는 공간
      - 과연, Working Directory 의 모든 변경사항들을 Version 으로 만들어야 하는가? ⇒ 변경사항들 중 다음 버전이 될 File 들을 **선별해서** 선별된 File 들을 Version 으로 만든다.
      - 그래서 Staging Area 가 필요한 것이다.

   2) Staging Area
      - Version 이 될 후보들이 올라오는 공간
      - Working Directory 에서 선별

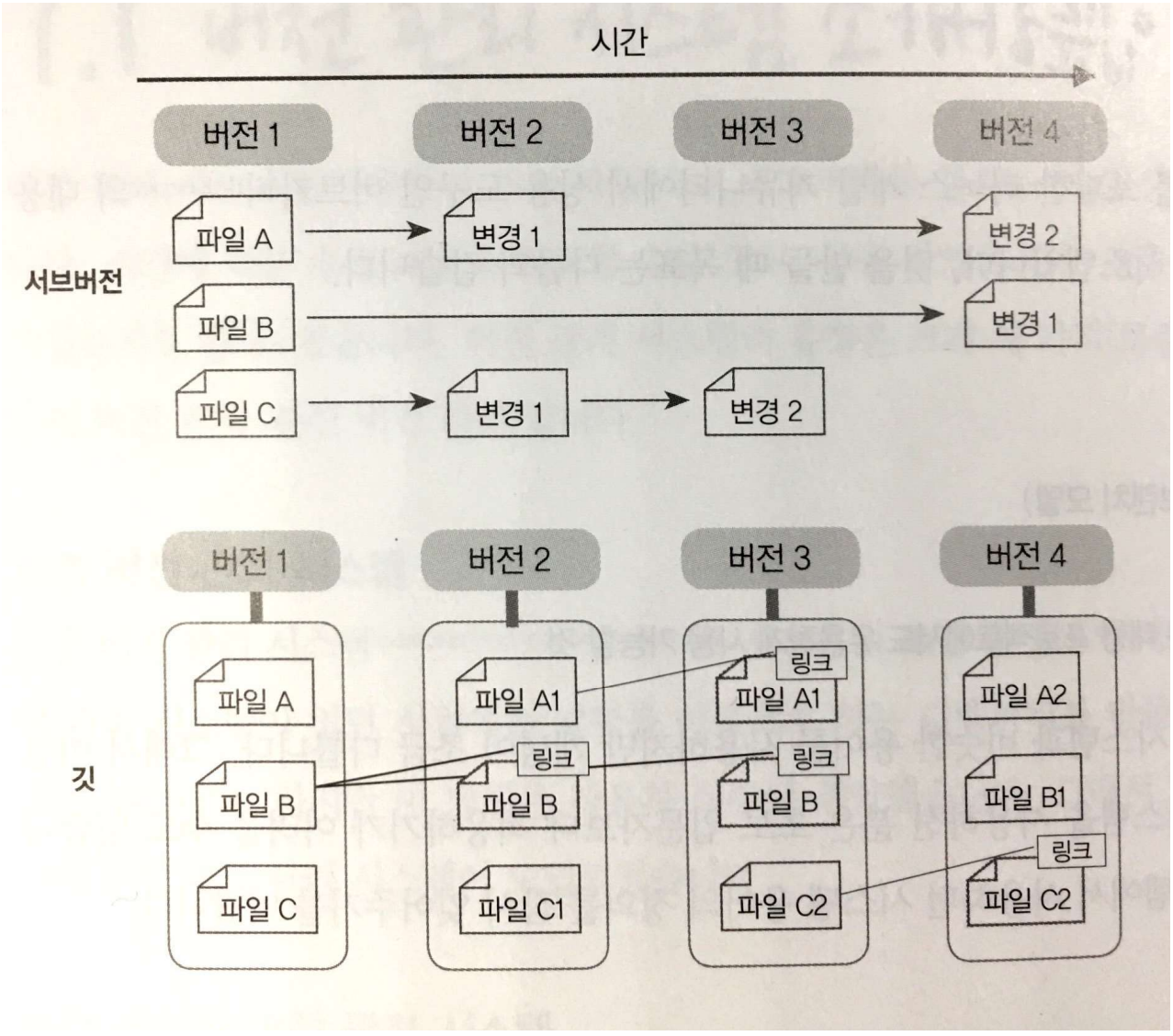
3) Repository
   - Version 들이 저장되어 있는 공간

7. 초기 목표
   1) 속도(Network 및 File 처리)
   2) 단순한 구조
   3) 동시 다발적인 개발
   4) 비선형적인 개발(branch model)
   5) 완벽한 분산
   6) 책임성
   7) 대형 project 를 효율적으로 지원

8. 저장소
   1) 저장소(Git repository)란 File 이나 folder 를 저장해 두는 곳.
   2) Git 저장소는 File 이 변경 이력별로 구분되어 저장된다.
   3) 비슷한 File 이라도 실제 내용 일부 문구가 서로 다르면 다른 File 로 인식하기 때문에 File 을 변경사항별로 구분해 저장할 수 있다.
   4) Git 은 원격 저장소와 Local 저장소 두 종류의 저장소를 제공한다.
   5) 원격 저장소(Remote Repository)는 File 이 원격 저장소 전용 Server 에서 관리되며 여러 사람이 함께 공유하기 위한 저장소를 말한다.
   6) 반면 로컬 저장소(Local Repository)는 내 PC 에 File 이 저장되는 개인 전용 저장소이다.
   7) 평소에는 내 PC 의 Local 저장소에서 작업하다가 작업한 내용을 공개하고 싶을 때에 원격 저장소에 Upload 를 하면 된다.
   8) 반대로 원격 저장소에서 다른 사람이 작업한 File 을 Local 저장소로 가져올 수도 있다.
   9) 저장소를 만드는 방법은
      - 저장소를 새로 생성한다.
      - 이미 만들어져 있는 원격 저장소를 Local 저장소로 복사해서 사용한다.

9. Git 의 File 관리 방법
   1) Git 이전의 System 에서는 각 File 의 변화를 시간순으로 관리하면서 Version 관리를 했다.
   2) 예를 들면 SubVersion 은 변경 내용만 저장한다.
   3) Git 에서는 data 를 File System snapshot 으로 취급한다.

4) File 에 변경이 없을 때는 File 을 새롭게 저장하지 않고 기존 File 에 대한 link 만 저장한다.
5) 아래의 그림을 보면, 4 번째 Version 을 생성하는 경우, SubVersion 은 각 File 의 변경 내용을 다 적용한 후에 최신 File 을 만든다(일종의 Differential Backup 같은 개념).
6) 반면, Git 은 File B1 와 File C1 에 이미 변경 내용이 적용되어 있다.
7) 변경이 없는 File A1 의 link 를 따라 File 을 취득해서 'Version 4'를 생성하게 된다.
8) 즉 SubVersion 은 Version 을 확정(SubVersion 에서는 commit, 혹은 tag)할 경우에 앞의 변경(revision) 내용을 순차적으로 적용해야 하므로 Git 에 비해서 시간이 많이 걸리는 단점이 있다.
9) Git 은 상대적으로 용량을 많이 필요로 하지만, 빠르게 최신 Version 을 가져올 수 있다.
10) 참고로, 아래의 그림에서 SubVersion 에서 화살표는 Merge 의 의미이고, Git 의 선은 서로 연결되어 있다는 의미이다.



[쇼다 츠야노 저, 전민수 저, 배효진 역, 자바 프로젝트 필수 유틸리티, (한빛미디어;2018), p.408]

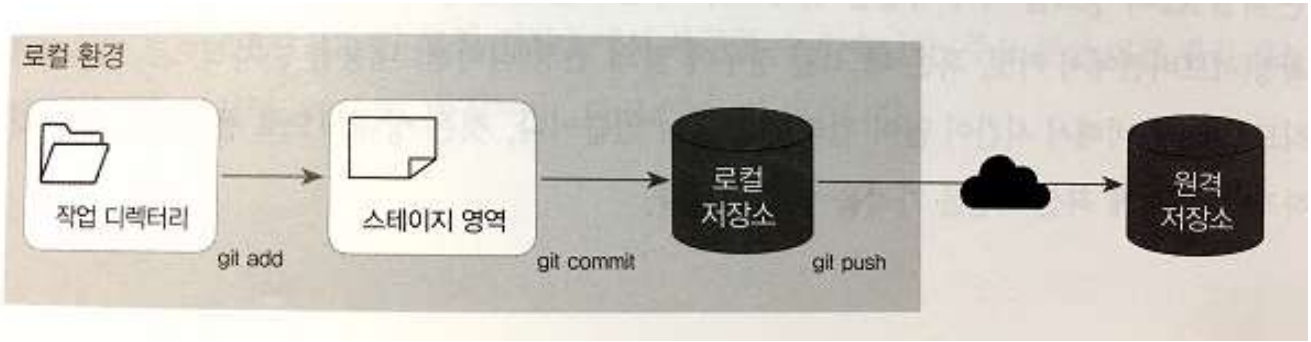
10. 동작원리

1) Snapshot
   - Data 를 가져오거나 Project 를 저장할 때마다 그 시점의 File 에 대해 Snapshot 을 저장한다.

- 변경되지 않은 File 은 다시 File 에 저장하지 않고 지정한 동일한 File 을 Link 한다.

2) Checksum
   - Data 를 저장하기 전에 Checksum 을 구하고 이 Checksum 을 통해 Data 를 관리한다.
   - SHA-1 hash 사용
     - 16 진수 문자 40 개로 구성된 문자열
     - File 의 내용 또는 Directory 구조를 기반으로 계산
   - File 이름이 아닌 Contents 의 Hash 값을 저장한다.
     - File 의 이름이 변경되더라도 내용이 동일하면 같은 Hash 를 갖는다.

3) Sections(Work Tree)
   - Working Directory
     - Repository 의 project 를 checkout 하거나 수동으로 생성
   - Staging Area(Index)
     - Repository 에 commit 하기 위한 중간 저장소
   - Repository
   - Commit 은 working directory 에 있는 변경 내용을 저장소에 upload 하는 과정이다.
   - 그런데, 바로 upload 하는 것이 아니라 중간 단계인 Staging area(Index)에 File 상태를 기록해야 한다.
   - 따라서 저장소에 변경 사항을 기록하기 위해서는, 기록하고자 하는 모든 변경 사항들이 Staging area(Index)에 존재해야 한다.



[쇼다 츠야노 저, 전민수 저, 배효진 역, 자바 프로젝트 필수 유틸리티, (한빛미디어;2018), p.408]