1. **GIT 소개**

***GIT이란?***

: Linux소스를 관리하기 위해 Linus Torvalds가 개발한 소스 버전 관리 프로그램이다.

무료이자 오픈 소스이며 분산 버전 관리 시스템(DVCS)이다. 속도와 효율이 우수하고 작은 프로젝트부터 큰 프로젝트까지 다양하게 사용할 수 있도록 디자인 되었다.

Git Clone들은 full-fledged repository. 모든 history, revision추적 기능, 중앙 서버로부터 서로의존적이지 않다.

Git 은 Mercurial, Bazaar, Subversion, CVS, Perforce, Visual SourceSafe 툴과 같이 파일에 대한 버전 관리에 사용된다.

현재 GIT 프로그램은 Windows, Mac OSX, 기타 Linux 시스템을 위하여(Binary, Source) 로 제공이 된다.

Windows, Mac OSX 용 GIT 프로그램은 Google 에서 배포하고 있다.

***GIT의 역사***

1991~2002 : Linux 커널 유지보수는 소프트웨어의 패치와 압축된 형식으로 전달.

2002 : Linux 커널 프로젝트는 독점권이 있는 BitKeeper라는 분산 버전 관리 시스템(DVCS)을 사용.

2005 : Linux 커널 개발자 커뮤티니와 BitKeeper를 개발한 회사와의 관계가 나빠짐. Linux 개발자 커뮤니티(대표주자 Linus Torvalds)는 BitKeeper를 대체할 무료의 DVCS을 개발하기 시작.

***GIT 개발 목표***

|  |  |  |
| --- | --- | --- |
| 1. 속도 |  |  |
| 1. 디자인의 간결함 | | |
| 1. 완전한 분산 시스템 | | |
| 1. Non-linear하지 않은 개발에 대한 강력한 지원 (수천 개의 branch들을 지원할 수 있도록) | | |
| 1. Linux 커널과 같은 매우 큰 프로젝트를 효율적으로 관리할 수 있는 시스템 | | |

***GIT을 버전관리로 채택, 사용하는 Project***

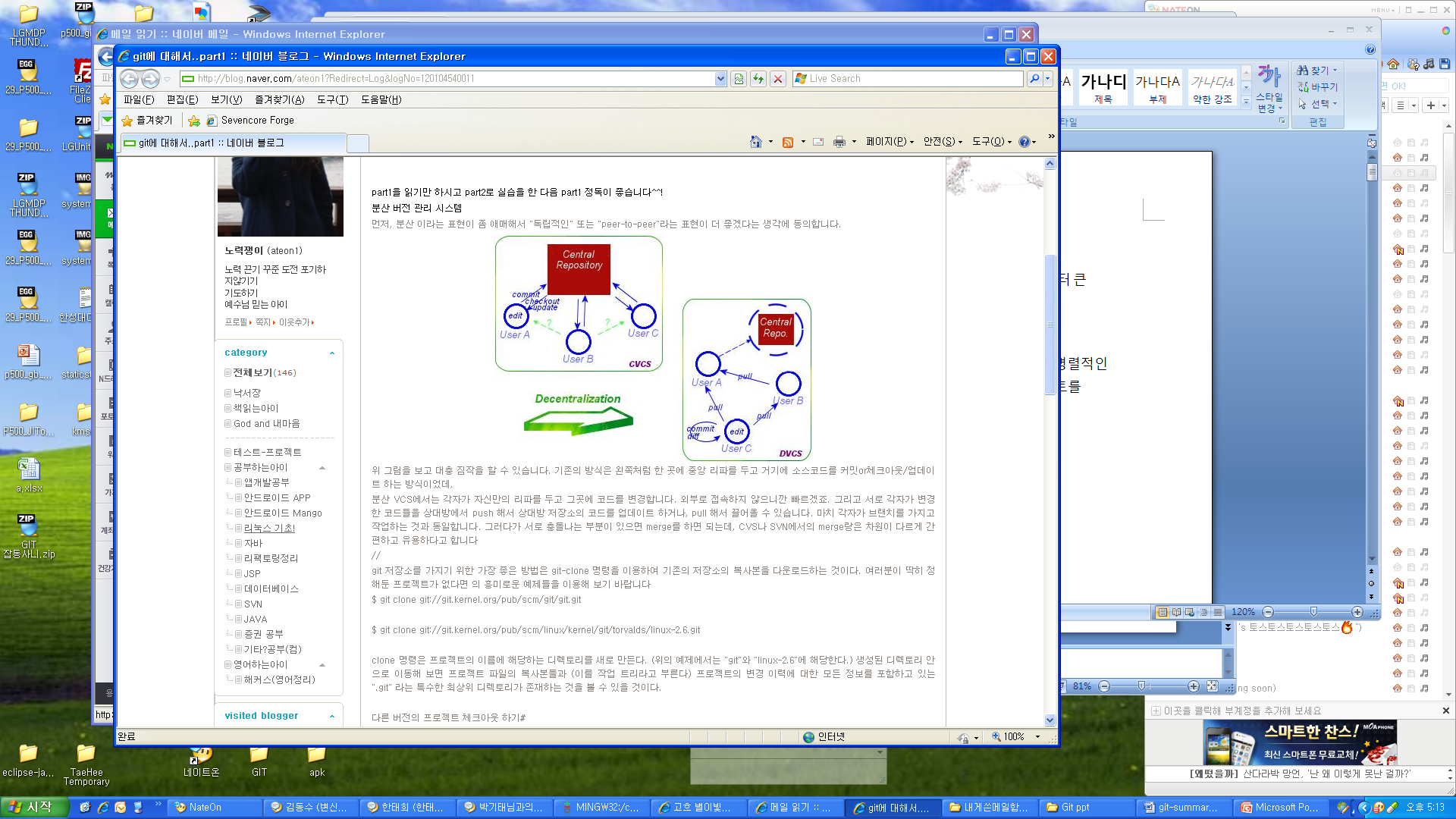
|  |  |  |  |
| --- | --- | --- | --- |
| 1. GIT | 1. Linux Kernel | 1. Perl | 1. Eclipse |
| 1. Gnome | 1. KDE | 1. Qt | 1. Ruby on Rails |
| 1. Android | 1. Debian | 1. X.org |  |

1. **GIT 특징**

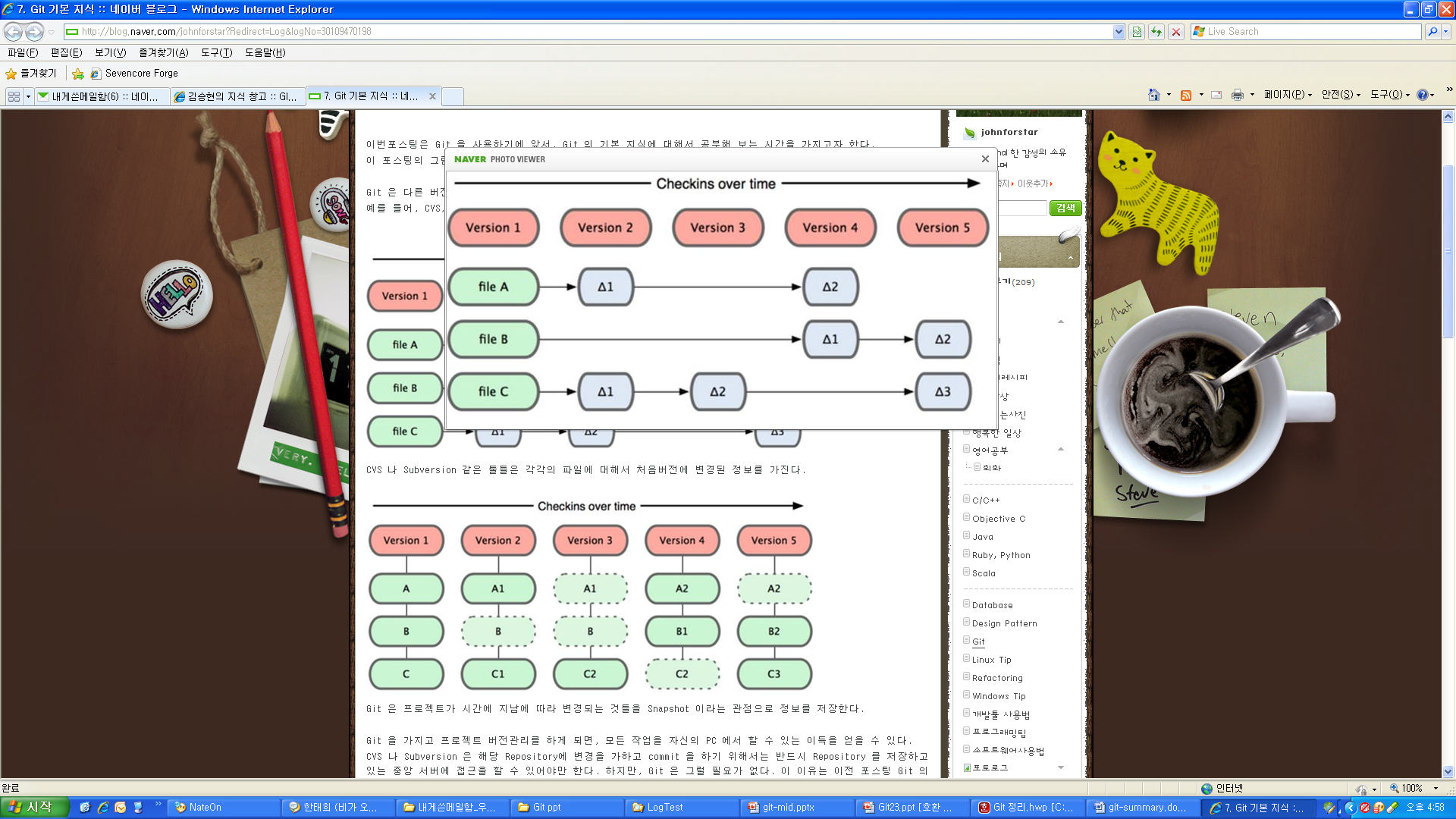
***<용어정리>***

|  |  |
| --- | --- |
| 1. *저장소*   *(Repository)* : | 사용자가 변경한 모든 내용을 추적하는 공간이다. 대부분의 VCS는 코드의 현재 상태는 물론이고 변경이 언제 발생했는지, 누가 변경했는지 변경사항을 설명하는 텍스트 로그메시지까지 저장한다. |
| 1. *작업 트리*   *(Working Tree)* : | 저장소를 어느 한 시점을 바라보는 작업자의 현재 시점이다. |
| 1. *체크아웃*   *(Checkout)* : | 작업자의 작업 트리를 저장소의 특정 시점과 일치하도록 변경하는 작업이다. |
| 1. *스테이징 영역*   *(Staging area)* : | 저장소에 commit하기 전에 commit을 준비하는 위치. index라고도 불린다. (변경사항을 적용하기 전에 한번 더 변경사항을 정리하고 다듬을 수 있는 기회를 제공한다. 변경사항을 추가하기 위해서는 git add 를 사용한다. commit예정인 변경사항이 있다고 보면 됨.) |
|  | |
| 1. *master* : | master branch는 복사해온 저장소 내의 HEAD의 복사본이다. |
| 1. *브랜치*   *(Branch)* : | 하나의 개발라인을 의미한다. 가장 기본이 되는 master branch에서 버그 수정이나 특정 기능을 추가하기 위해서 개발라인을 따로 두고 작업할 수 있다. 한 개의 프로젝트에서도 여러 개의 개발라인이 존재할 수 있다.  하나의 git 저장소는 여러 branch의 개발과정을 관리할 수 있다. 이는 한 개의 branch내에서 가장 최근에 commit된 reference인 HEAD(branch head)의 목록을 관리하는 방식으로 수행된다.  (Ex. branch apple에 3개의 commit이 있는데 이 중 가장 최근에 추가된 commit이 HEAD.)  Branch를 생성하면 파일이 분기하는 위치가 repository에 기록된다. branch는 다른 branch와 분리하여 내용에 대한 변경사항을 지속적으로 추적한다. 합쳐지거나 여러 개로 나뉠 수 있다. |
|  | |
| 1. *Origin* : | 단지 GIT가 복사해 온 저장소를 가리키기 위해 기본적으로 사용하는 이름이다. |
| 1. *Pushing* : | 개발자가 직접 변경사항을 중앙 저장소에 전송하려고 접속하는 것이다. |
| 1. *Pulling* : | 변경사항을 이용하기 위해 GIT 저장소에 변경사항을 가져온 후, 변경이력을 지역이력과 합치는 것이다. Subversion의 update와 유사하다. |

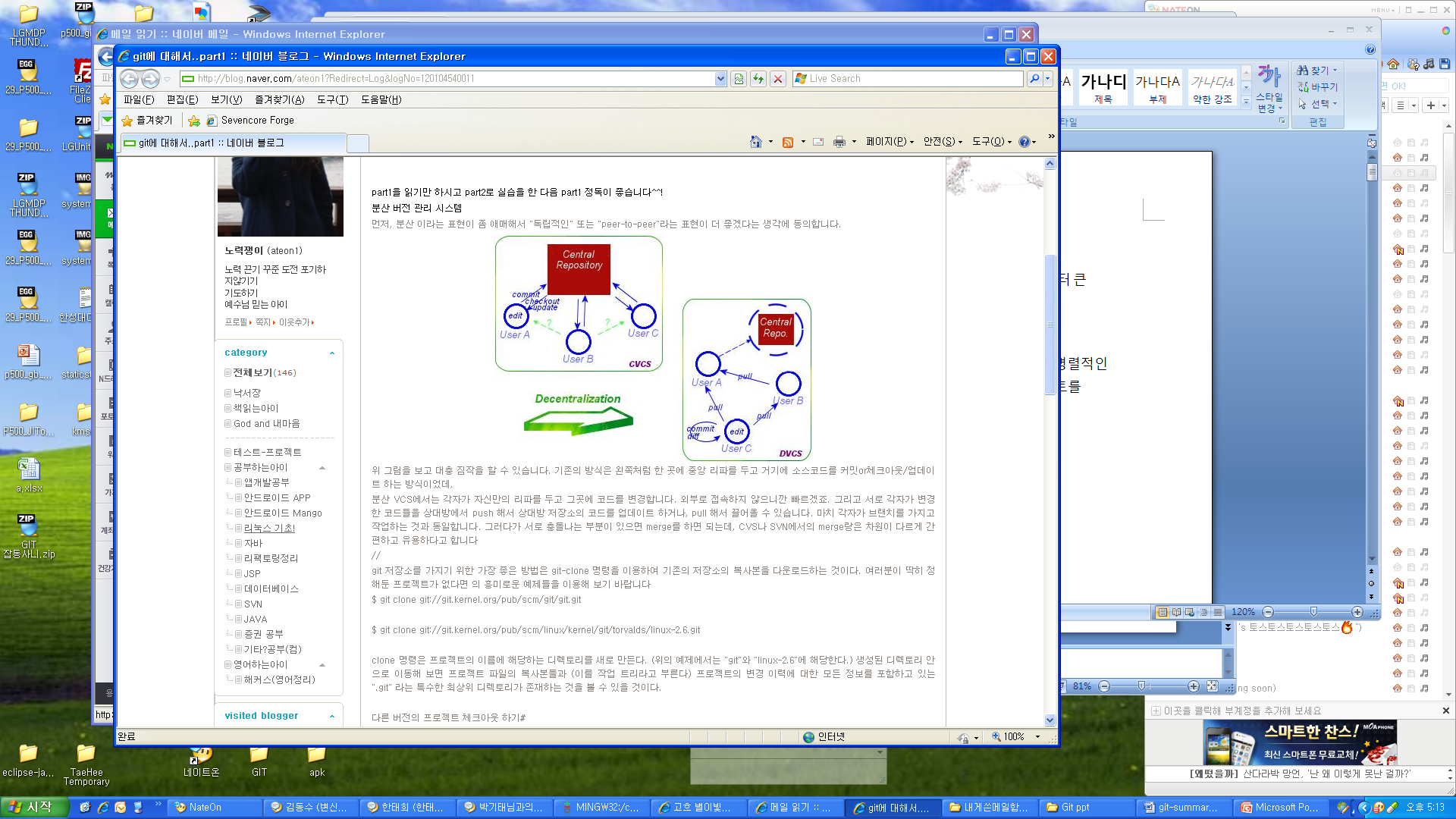
***CVS, Subversion 비교***



|  |
| --- |
| **CVS, Subversion** |
| * **Centralized** * Network너머 **SVN Server에 저장소 존재.** * **모질라의 경우**   : 12GB   * 개발자들은 원격에서 작업한 후 네트워크 연결을 통해 저장소에 변경 사항을 전송. * **중앙 집중식 저장모델**   : 하나의 중앙 저장소가 있고, 모두가 이 중앙 저장소에 변경 사항을 전송. 각 개발자는 저장소의 최신버전을 복사해서 가지고 있고, 이 복사본을 변경한 후에는 다시 중앙 저장소에 전송.   * **한계**   : 사용자는 최신버전의 코드만 가짐. 변경이력을 보려면 저장소에 정보를 요청.(즉, 네트워크 상에서 원격 저장소에 항상 접근 되어야 함) |



***GIT***



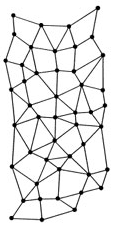
|  |
| --- |
| **GIT** |
| * **분산 버전 관리 시스템 (DVCS)** * File들의 변경이력을 Snapshot에 압축하여 모아두는 방식.   프로젝트의 Snapshot을 찍어 데이터를 저장한다.  소스코드를 commit할 때 또는 프로젝트의 현재 상태를 저장하고자 할 때, 기본적으로 그 시점의 모든 File상태에 대한 Snapshot을 찍어 그 Snapshot에 대한 정보를 저장한다.     * 효율성을 위해 변경되지 않은 파일은 새로운 Snapshot에 저장하지 않고, 기존의 파일에 대한 링크만 저장한다. (현재의 Snapshot에 링크만 함.) * GIT은 **완전한 분산환경**이므로 pushing하고 pulling하는 repository가 여러 개 일 수 있다. * 작업 트리를 처음 가져오기 위해 자신의 프로젝트를 저장소에 초기화하도록 요청하거나 기존 저장소의 프로젝트를 복제.      * 변경사항을 commit하면서 저장소에 새로운 revision추가, 무엇을 변경했는지 log메시지 저장. * 상위 저장소에 변경 사항을 pushing하면 바뀐 내용을 다른 개발자와 공유 가능. * 변경사항을 가져와 이용하려면  1. 변경사항을 가져온다. (remote repository에 있는 변경사항의 복사본이 생성된다. Pushing과 반대개념으로 변경사항을 저장소에 전송하는 것이 아니라, remote repository가 가지고 있는 변경사항을 요청.) 2. 가져온 변경사항을 지역 이력과 합친다. 보통 변경사항을 가져오고(fetch) 합치는 작업(merge)을 동시에 진행하기 때문에 이 두 가지 작업을 한번에 하도록 풀링(Pulling) 이라는 기능을 지원한다. (Pulling – Subversion의 update와 유사. Pulling = fetch+ merge)  * GIT은 완전한 분산환경이므로 Pushing하고 Pulling하는 repository가 여러 개 일 수 있다.   **특징을 크게 5가지로 나누면**   1. **Distribute development** 2. **Strong support for non-linear development** 3. **Efficient handling of large projects** 4. **Cryptographic authentication of history** 5. **Toolkit design** |

* 정리하면

|  |  |
| --- | --- |
| **GIT** | **SVN** |
| Distribute | Centralized |
| Repository : local에 있다. | Repository : network너머 svn server에 있다. |
| * GIT의 performance가 더 압도적이다. | |
| 모질라의 경우 : 420MB | 모질라의 경우 : 12GB |
| * GIT이 더 작은 file space를 차지한다. | |
| GIT가 브랜치 컨트롤이 더 좋다. | |
| SVN의 user interface가 강력하다. | |

***? ) Distributed***

* Linus’ Vision of Git



***GIT 특징 5가지***

1. **Distributed development**

각자가 프로젝트의 전체 이력이 있는 자신만의 저장소를 가진다. (local에 저장소를 가진다.)

Commit 할 때는 원격 저장소에 연결할 필요가 없이 변경사항을 지역 저장소에 기록한다. (프로젝트 변경 이력을 저장하기 위해 네트워크에 접근할 필요가 거의 없다.)

따라서 모든 작업을 자신의 PC에서 할 수 있다.

중앙 저장소에 전송하는 것을 Pushing이라고 한다. 변경 사항을 모아둔 패치를 만들어 프로젝트 관리자에게 제출할 수도 있다.



1. **Strong support for non-linear development**

빠르고 편한 branch와 merge기능을 지원한다.

Non-linear한 개발 이력을 시각화하고 탐색할 수 있는 강력한 틀(gitk)을 제공한다.

1. **Efficient handling of large projects**

매우 빠르고, 크고 history내역이 긴 프로젝트에 매우 잘 작동한다.

다른 오픈 소스 VCS보다 빠르며, 효율적인 revision 관리를 수행한다.

1. **Cryptographic authentication of history**

GIT history는 revision이란 특수한 이름으로 저장한다.

revision은 그 commit까지의 완전한 개발 history에 의존한다. 일단 만들어지면, 오래된 버전의 revision은 통지 없인 변경이 불가능하다. 또한 tag들은 암호화되어 저장된다.

1. **Toolkit design**

GIT은 C언어로 작성된 많은 Tool들의 집합체이다.

개발자가 사용하기 쉽도록, 또 새로운 기능을 수행할 수 있도록 script 작성을 쉽게 하기 위해 tool들을 제공한다.

***GIT의 강점***

* **분산 아키텍처**

: 완전히 연결이 끊어진 상태에서, 항상 인터넷에 연결 되어야 하는 고통 없이 동작한다.

* **쉬운 branch 만들기와 합치기**

: 다른 VCS와는 달리 branch를 여러 번 나눈 경우라도 간단히 합칠 수 있다.

* **서브 버전과 통신**

: GIT은 서브버전 repository의 모든 이력을 가져올 수 있으며, 변경사항을 다시 보낼 수 있다.

***GIT Workflow***

* Fetch or clone
* Modify the files in the local branch
* Stage the file
* Commit the files locally
* Push change to remote repository