

# Git의 구조와 주요 명령어 이해하기

2021963057 장문용



## Chapter 1: Git 소개

# Git이란 무엇인가?

정의: 분산형 버전 관리 시스템 (DVCS)

**Git**은 소프트웨어 개발 과정에서 소스 코드의 변경 이력을 효율적으로 관리하기 위해 설계된 시스템입니다. 중앙 서버에 의존하지 않고 각 개발자가 완전한 저장소를 가지는 분산형 구조가 특징입니다.

속도

로컬 작업으로 인한 빠른 성능.



무결성

데이터 변경 이력의 안전한 관리.

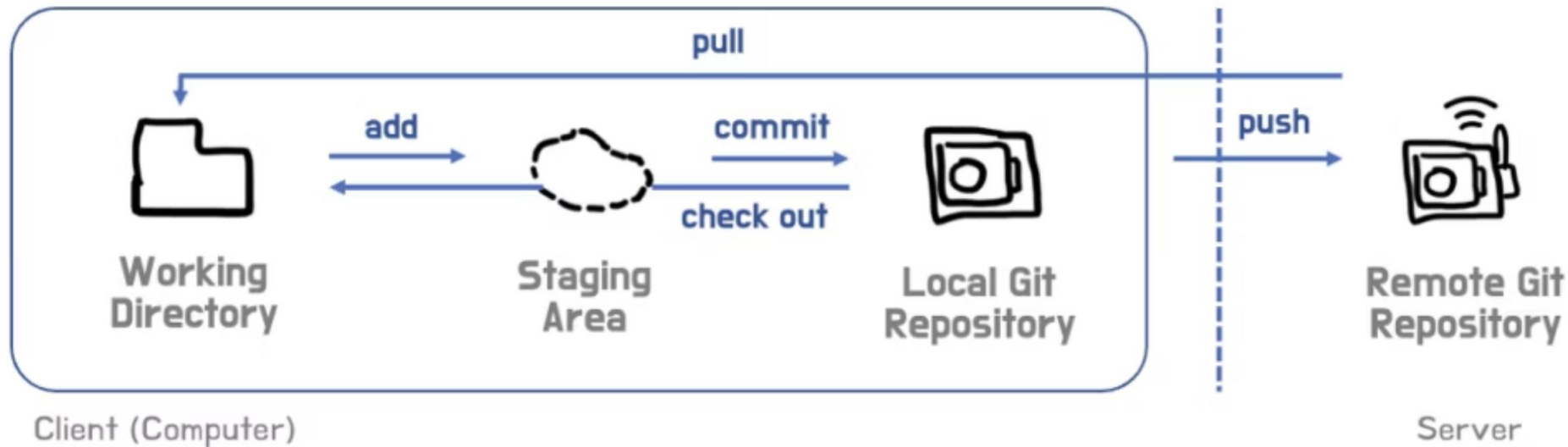


비선형 개발

유연한 브랜치 및 병합 지원.

# Git의 핵심 구조: 3단계 영역 이해하기

**Git**은 효율적인 버전 관리를 위해 세 가지 주요 작업 영역을 사용합니다. 이 영역들을 이해하는 것은 **Git**의 동작 원리를 파악하는 데 필수적입니다.



## 작업 디렉토리 (Working Directory)

실제 파일을 수정하고 개발하는 공간입니다. 로컬 컴퓨터에 있는 프로젝트 파일들이 여기에 해당합니다.



## 스테이징 영역 (Staging Area / Index)

커밋할 변경 사항들을 임시로 모아두는 공간입니다. `git add` 명령어를 통해 이 영역으로 파일을 추가합니다.



## 로컬 저장소 (Local Repository)

스테이징 영역에 있는 변경 사항들이 최종적으로 저장되는 곳입니다. `git commit` 명령어를 통해 버전 이력으로 기록됩니다.

각 영역은 서로 다른 목적을 가지며, 이들 사이의 전환을 통해 파일의 변경 이력이 체계적으로 관리됩니다.

# 로컬 저장소 관리의 핵심

새로운 프로젝트를 시작하거나, 기존 프로젝트에 참여하고, 변경 사항을 저장소에 기록하는 데 필요한 기본 명령어입니다.

<code>git init</code>	현재 디렉토리에 새 <b>Git</b> 저장소 생성	<code>git init</code> : 새로운 프로젝트의 첫 단계를 시작합니다. 숨겨진 <code>.git</code> 디렉토리가 생성됩니다.
<code>git clone</code>	원격 저장소의 모든 이력 복제	<code>git clone [URL]</code> : 원격 서버에서 프로젝트 전체를 로컬로 가져와 작업을 시작합니다.
<code>git add</code>	변경 파일을 <b>Staging Area</b> 에 등록	<code>git add .</code> : 현재 작업 디렉토리의 모든 변경 사항을 커밋 대기 상태로 만듭니다.
<code>git commit</code>	<b>Staging Area</b> 의 변경 사항을 영구 저장	<code>git commit -m "초기 설정 완료"</code> : 스냅샷을 찍고 메시지를 추가하여 저장소에 기록합니다.

📌 **팁:** 커밋 메시지는 왜(**Why**) 변경했는지에 초점을 맞추는 것이 좋습니다.

# 원격 저장소와의 동기화 (협업의 시작)

분산형 시스템의 장점을 극대화하기 위해, 로컬 저장소와 원격 저장소(**Remote Repository**) 간의 동기화는 필수적입니다.

## git pull

원격 저장소의 변경 사항을 가져와 (**fetch**) 로컬 브랜치에 자동으로 병합 (**merge**) 합니다.

```
| git pull origin main
```

## git push

로컬 저장소에 생성된 커밋들을 원격 저장소로 업로드합니다.  
협업자들과 자신의 변경 사항을 공유합니다.

```
| git push origin main
```

협업 시 가장 자주 쓰이는 명령어이며, 동시에 작업한 파일에 변경 사항이 있을 경우 **충돌(Conflict)**이 발생할 수 있습니다.  
이 경우 수동으로 병합을 처리해야 합니다.

## Chapter 3: 이력 되돌리기 및 관리

# 시간을 되돌리는 세 가지 방법

실수를 수정하거나 프로젝트 이력을 정리할 때 사용하는 세 가지 핵심 명령어의 역할과 특징을 비교합니다.



### git reset (위험도: 높음)

커밋 포인터 위치를 과거로 이동시킵니다. 로컬 이력을 되돌리는 강력한 명령어지만, 이미 푸시된 커밋에 사용하면 협업자들에게 문제를 일으킬 수 있습니다 (이력 손실 가능).



### git revert (위험도: 낮음)

이전 커밋의 변경 사항을 취소하는 새로운 커밋을 생성합니다. 기존 이력을 건드리지 않아 협업 시 가장 안전한 되돌리기 방식입니다.

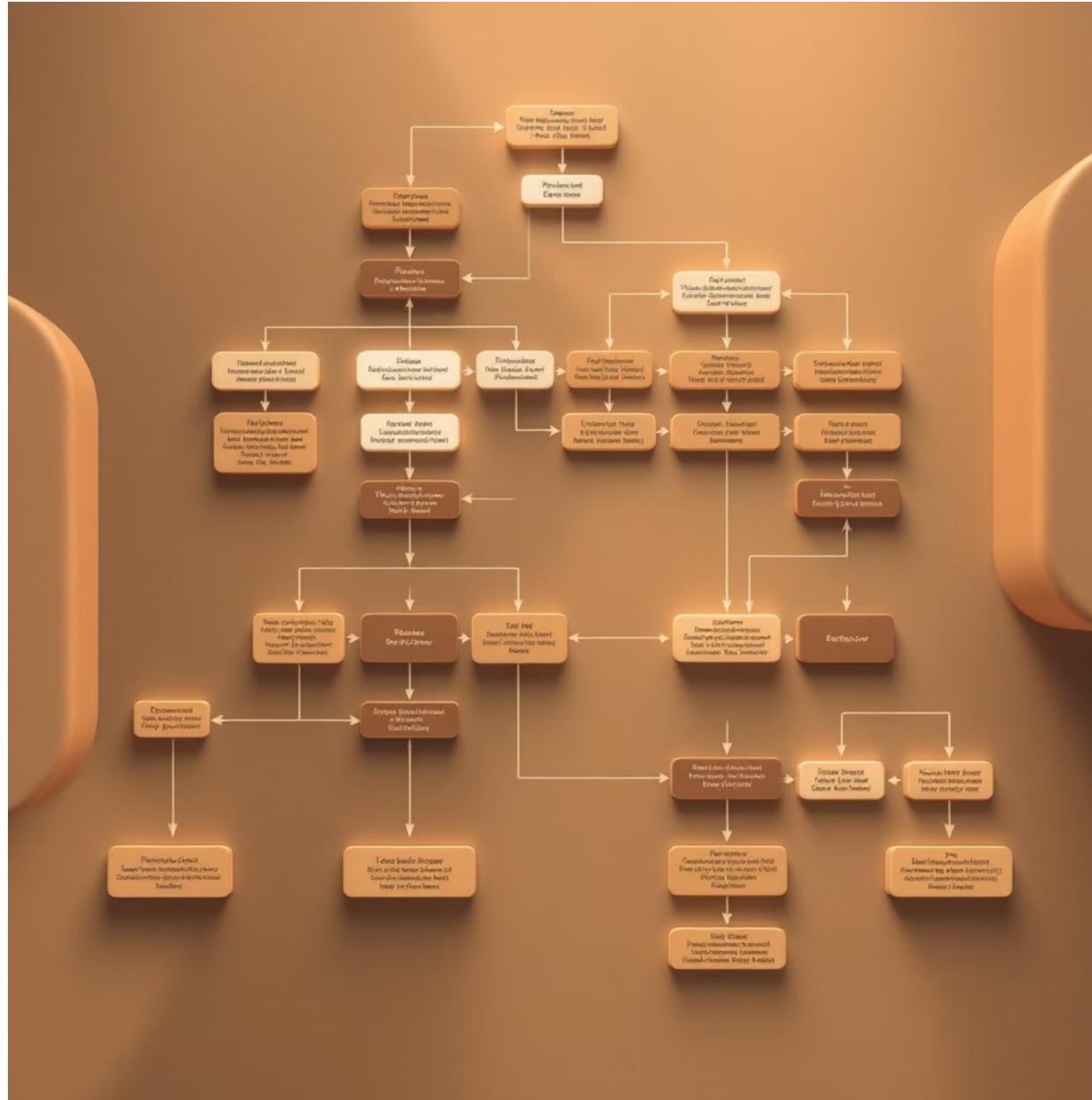


### git restore (범위: 파일)

작업 디렉토리나 **Staging Area**에 있는 개별 파일의 상태를 복원합니다. 커밋 전체가 아닌 특정 파일만 되돌릴 때 유용합니다.

# git rebase: 깔끔한 커밋 히스토리의 비밀

**Rebase**는 커밋들을 새로운 기반(**Base**) 위로 "재배치"하여 히스토리를 선형적이고 깔끔하게 유지하는 데 도움을 줍니다.



## → 기반 변경 (Rebasing)

현재 브랜치의 커밋들을 다른 브랜치의 가장 최신 커밋 위로 옮겨 마치 처음부터 그 브랜치에서 작업한 것처럼 보이게 합니다.

## → 대화형 리베이스 (Interactive Rebase)




`git rebase -i` 명령을 사용하면 커밋들을 합치거나(**squash**), 순서를 바꾸거나, 메시지를 수정하는 등 이력을 정리할 수 있습니다.




📌 ⚠️ **중요 주의사항:** 이미 원격 저장소에 **push**한 커밋에 대해 **rebase**를 수행하는 것은 피해야 합니다. 이력을 덮어쓸 경우 협업자들의 저장소에 문제가 발생할 수 있습니다. 꼭 필요하다면 **--force-with-lease** 옵션을 사용하세요.



# Git 명령 흐름 요약

새로운 기능을 개발하고 협업자들과 코드를 공유하는 일반적인 **Git** 워크플로우를 단계별로 정리합니다.

-  시작  
새 저장소 생성 (`git init`) 또는 기존 저장소 복제 (`git clone`).
-  수정  
**Working Directory**에서 코드를 자유롭게 수정하고 개발합니다.
-  준비  
`git add` 명령으로 변경된 파일을 **Staging Area**에 등록합니다.

-  기록  
`git commit -m` 명령으로 변경 스냅샷을 로컬 저장소에 영구 기록합니다.
-  공유  
`git push` 명령으로 로컬 커밋을 원격 저장소로 업로드합니다.
-  동기화  
다른 사람의 변경 사항을 반영하기 위해 `git pull` 명령을 수행합니다.



# 마무리: Git의 핵심 정리

**Git**은 프로젝트의 생명주기 동안 안정적인 협업과 이력 관리를 위한 가장 강력한 도구입니다.

## 스냅샷 기반

파일의 차이(**diff**)가 아닌 전체 파일 시스템의 스냅샷을 저장합니다.



## 분산형

모든 개발자가 완전한 저장소를 가지며 중앙 서버 없이도 작업할 수 있습니다.

## 고급 이력 관리

**rebase, reset, revert**를 통해 이력을 깔끔하고 안전하게 유지할 수 있습니다.



## 핵심 협업 명령

**add, commit, push, pull**만 숙달해도 대부분의 협업이 가능합니다.

**Git** 사용을 통해 더 효율적이고 체계적인 개발 환경을 구축하세요.

# 참고

- <https://aprilamb.com/%EC%86%8C%EC%8A%A4-%EA%B4%80%EB%A6%AC%EB%A5%BC-%EC%9C%84%ED%95%9C-git/>
- <https://blog.naver.com/since201109/222432491012>
- GPT
- GAMMA