<깃 설계 목적>

* 빠른 속도
* 단순한 디자인
* 비선형적 개발 지원(수천 개의 브랜치를 병행할 수 있음)
* 완전 분산형 시스템
* 리눅스와 같은 거대한 프로젝트도 속도 저하의 문제없이 관리할 수 있는 시스템

<Repository 만들기>

1. Git init | 비어 있는 레포지토리(프로젝트 디렉토리의 각 버전이 담기는 저장소)를 생성
2. ls -al | .git 🡪 레포지토리

<Commit 주의사항>

* 처음으로 커밋을 하기 전, 사용자 이름과 이메일 주소 설정
* 커밋 메시지 남기기(옵션 -m)
* 커밋할 파일을 git add로 지정

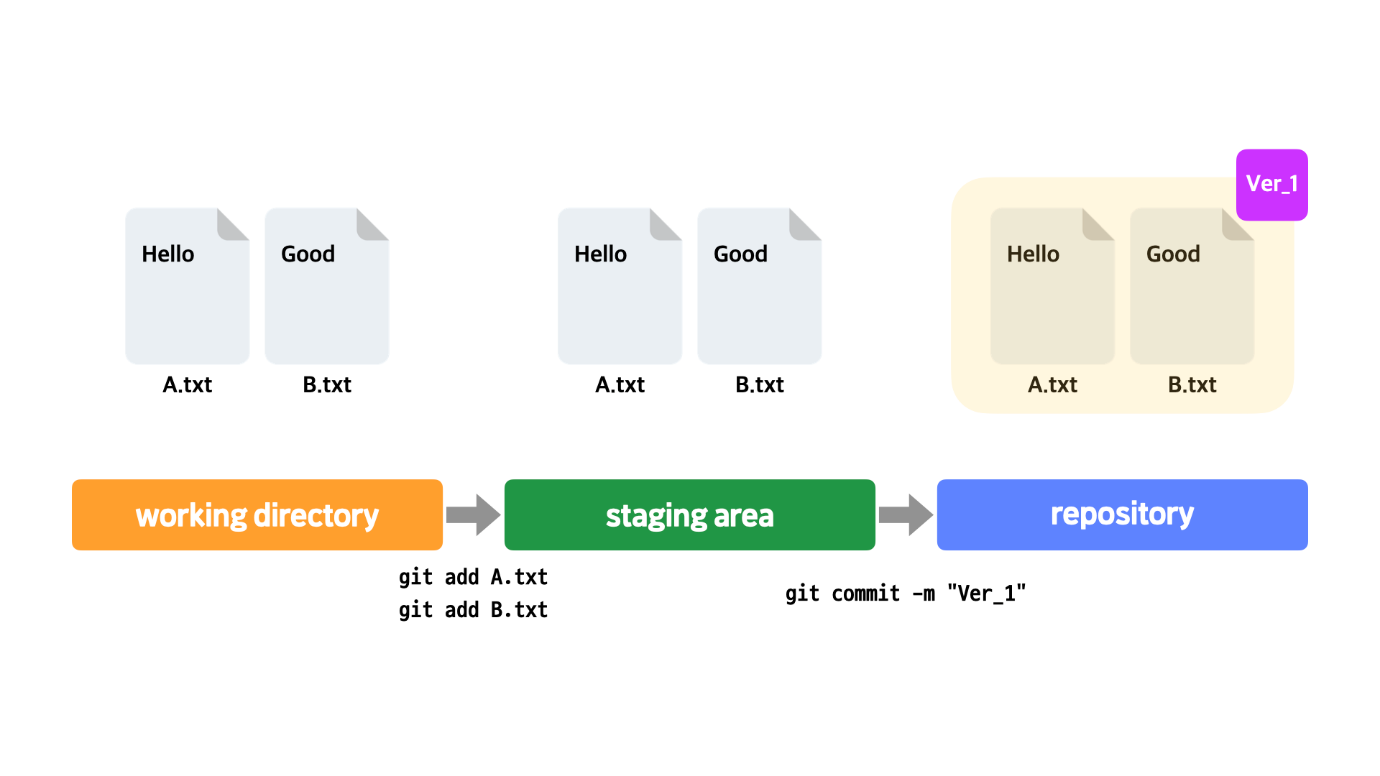
<Git 작업 영역>

1. working directory(working tree): 작업하는 프로젝트 디렉토리
   * ex. MathTool 디렉토리
2. staging area(index): git add를 한 파일들이 존재하는 영역. 커밋을 하게 되면 staging area에 있는 파일들만 커밋에 반영됨
3. repository: working directory의 변경 이력들이 저장되어 있는 영역. 즉, 커밋들이 저장되는 영역

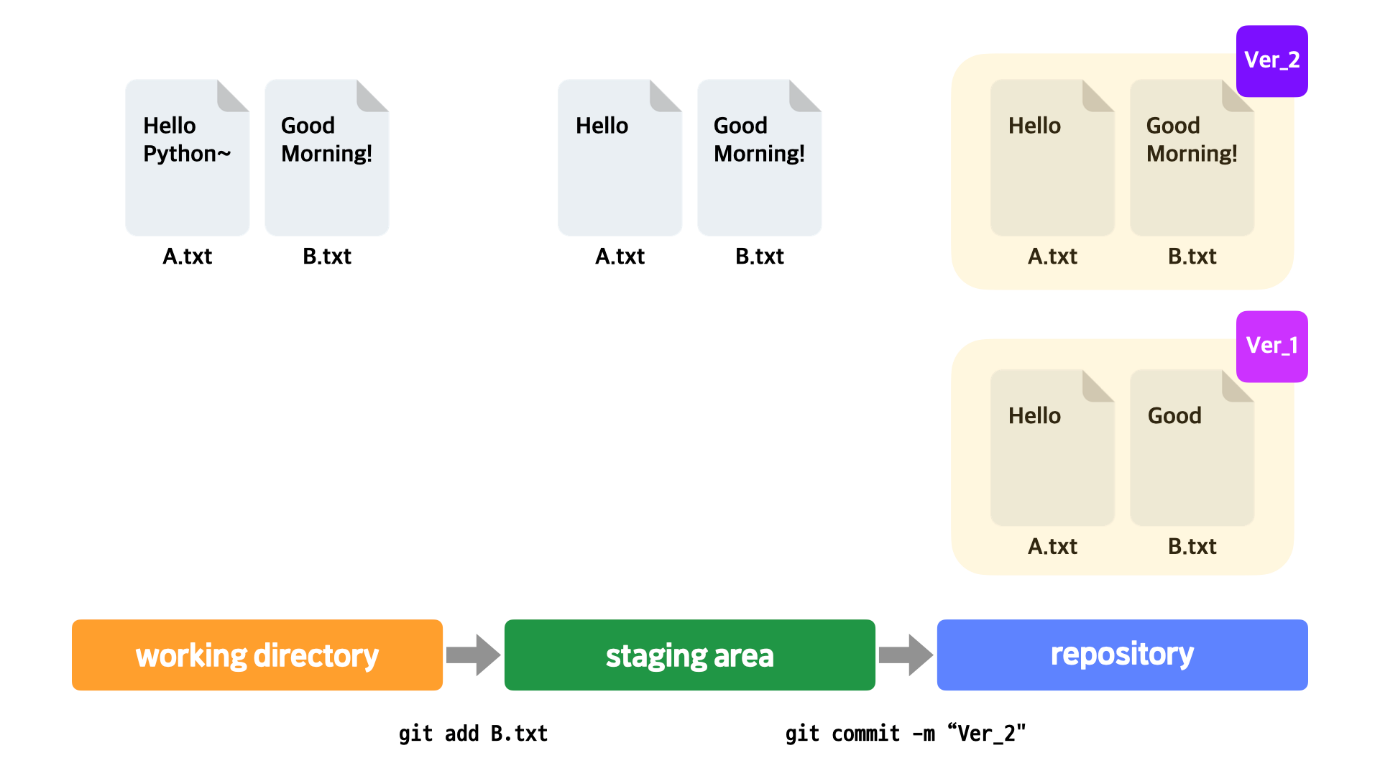
정리하면,

* working directory에서 뭔가 작업을 하고,
* 작업한 파일들을 git add 해주고,
* 커밋을 하면 staging area에 있던 파일들의 모습이 마치 영화의 한 장면, 스냅샷(snapshot)처럼 이 repository에 저장되는 것.

<Git 작업영역 예제>



1. working directory에서 A.txt 파일과 B.txt 파일을 작성하고
2. git add A.txt와 git add B.txt를 실행해서 A.txt, B.txt 둘 다 staging area에 올렸습니다.
3. 그 다음 git commit -m "Ver\_1"를 실행해서 staging area에 있는 파일들을 가져와 커밋으로 남겼습니다.



1. working directory에서 A.txt 파일 내용에 Python~이라는 단어를 추가, B.txt 파일 내용에 “Morning!”이라는 단어를 추가했습니다.
2. 그런데 이번에는 git add B.txt만 실행해서 B.txt 파일만 staging area에 올렸습니다.
3. 그 다음 git commit -m "Ver\_2"로 두 번째 커밋을 했습니다.

* 결론: working directory에서 수정했다는 사실은 동일하지만, staging area에 올렸는지 여부에 따라 그 최신 모습이 커밋에 반영되는지가 결정된다.