

Git 실습 Hands-on

Eclipse · EGit · GitLab 수강생용 실습 안내

2026-10-16 / Notion 접속이 어려울 때 사용하는 PDF 배포본

[첫 Commit용 스타터 ZIP 다운로드](#)

코드블록은 터미널 명령이 아니라 파일 내용·브랜치 이름·Commit 메시지를 복사하기 위한 자료입니다. 슬라이드 번호는 Notion 원문 기준이며 추가 안내 슬라이드에 따라 차이가 있을 수 있습니다.

완성본 슬라이드의 실습 01~06과 환경 승격 종합 실습을 따라가는 수강생용 안내입니다. Git 작업은 Eclipse/EGit와 GitLab Web UI에서 진행합니다. 아래 코드블록은 터미널 명령이 아니라 파일 내용·브랜치 이름·Commit 메시지·MR 설명을 복사하기 위한 자료입니다.

[참고 페이지](#)

[교육용 GitLab](#)

[참고용 그룹](#)

0. 시작 전 — 내 저장소와 번호 확인

01은 예시 수강생 번호입니다. 강사가 배부한 번호로 브랜치 이름의 끝 번호를 바꿉니다. 이름·팀번호·GitLab 사용자명도 본인 정보로 바꿉니다. 비밀번호나 토큰은 이 페이지·README·MR에 입력하지 않습니다.

- git-local-lab: 내 PC에만 있는 저장소. 첫 Commit·로컬 Merge·Conflict·Reset 실습.
- git-collab-번호: 개인 Fork. Clone·Push·Revert·MR 실습.
- git-promotion-팀번호: 팀 Fork. dev → stage → prod 실습.

각 실습 전에 프로젝트 이름, 현재 브랜치, Working Tree 상태를 확인합니다. C0는 처음 Commit을 부르는 교안용 이름입니다. 로컬 실습의 C0와 개인 Fork의 초기 Commit은 서로 다른 저장소의 이력입니다. 다른 사람의 Commit SHA를 복사하지 않습니다.

개인 Fork 준비

원본 저장소 주소를 브라우저에서 엽니다.

<https://gitlab.musjung.com/git-training/git-collab-01>

GitLab에서 Fork를 선택하고 자신의 namespace에 생성합니다. 가능하면 프로젝트 이름을 강사가 배부한 [git-collab-번호](#)로 지정합니다. 생성 후 자신의 Fork의 Code → Clone with HTTPS에서 URL을 복사합니다. 원본 그룹 URL을 Clone 주소로 쓰지 않습니다.

Eclipse에서는 File → Import → Git → Projects from Git → Clone URI로 가져오고 Existing Eclipse Project를 Import합니다. GitLab 사용자명과 본인의 [write_repository](#) 범위 토큰으로 인증합니다. 토큰은 비밀번호 입력란에서만 사용합니다.

Fork를 만들면 MR target이 원본으로 선택될 수 있습니다. 이번 교육은 자신의 Fork 내부에서 feature → main으로 병합합니다. MR 생성 시 target project를 반드시 확인합니다.

강사는 짝 리뷰어 접근권한, main 보호, 병합 담당자를 수업 전에 설정합니다. 보호된 main에는 직접 Push하지 않습니다. Fork의 보호 설정과 역할은 원본에서 그대로 적용됐다고 가정하지 말고 확인합니다.

로컬 첫 Commit용 프로젝트

첫 Commit 실습은 GitLab에서 Clone한 저장소가 아니라, Git 이력 없는 아래 스타터 ZIP으로 시작합니다. 파일을 내려받으면 강사에게 별도로 받을 필요 없이 직접 준비할 수 있습니다.

스타터 ZIP 다운로드

1. 위 스타터 ZIP 다운로드 링크에서 [git-local-starter.zip](#)을 다운로드합니다.
2. 압축을 해제해 [git-local-lab](#) 폴더를 확인합니다. 다운로드한 ZIP 자체를 Import하지 않습니다.
3. Eclipse의 File → Import → General → Existing Projects into Workspace를 선택합니다.
4. Select root directory에서 압축 해제한 [git-local-lab](#) 폴더를 선택하고 Projects의 [git-local-lab](#)을 확인한 뒤 Finish합니다. 원래 폴더를 그대로 사용할 때는 Copy projects into workspace를 해제합니다.
5. 아래 초기 실행 결과를 확인한 뒤 실습 01의 Team → Share Project → Git에서 직접 저장소를 만들고 첫 Commit C0를 기록합니다.

ZIP에는 [src/training/App.java](#), [config/application.yml](#), [README.md](#), [.project](#), [.classpath](#), [.gitignore](#) 초기 파일 6개가 들어 있습니다. Git 이력·빌드 결과는 포함하지 않습니다. Eclipse Workspace는 별도 폴더를 사용합니다.

이미 Clone한 개인 Fork나 강사 참고 레포는 첫 Commit 실습에 사용하지 않습니다. 같은 이름의 Eclipse 프로젝트가 이미 있으면 기존 작업과 구분해 가져옵니다. 직접 파일을 만드는 경우 필요한 초기 파일은 페이지 마지막의 “초기 파일”에도 있습니다.

실행 확인: [src/training/App.java](#)에서 Run As → Java Application. 초기 Console 출력:

Git 실습 시작

설정 파일 초기 값은 [timeoutSeconds: 30](#)입니다. 이 프로그램은 YAML을 읽지 않습니다.

1. 실습 01 — 초기 Commit과 선택적 Stage

슬라이드 29~36 / 저장소 git-local-lab.

1-A. 초기 Commit C0 만들기

1. 프로젝트 우클릭 → Team → Share Project → Git에서 저장소를 생성합니다.
2. 기본 브랜치 이름은 `main`으로 통일합니다.
3. Git Staging에서 소스·문서·프로젝트 설정 파일을 모두 Stage합니다. `bin`과 빌드 산출물은 제외합니다.
4. 아래 메시지로 Commit하고 History에서 Commit ID를 기록합니다.

chore: 실습 프로젝트 초기화

- ☐ 현재 브랜치 `main`
- ☐ History에 초기 Commit C0
- ☐ Working Tree clean
- ☐ 초기 문구 실행 확인

1-B. App과 README를 수정하고 App만 Commit

App.java 전체 교체:

```
package training;
```

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("로컬 변경 확인");  
    }  
}
```

README.md 끝에 한 줄 추가:

메모: 로컬 변경을 확인했습니다.

App.java와 README.md를 저장합니다. Git Staging에서 App.java만 Stage하고 아래 메시지로 Commit합니다.

feat: 로컬 변경 확인 문구 변경

History에서 C0와 새 Commit의 Diff를 비교합니다. README 수정은 아직 Unstaged여야 합니다. README의 남아 있는 수정만 Discard하고 clean 상태로 정리합니다. App.java의 Commit은 그대로 유지합니다.

- ☐ Commit 2개
- ☐ 두 번째 Commit에는 App.java만 포함
- ☐ README 메모는 Commit되지 않았고 Discard됨

2. 실습 02 — 첫 Push, Fetch와 Pull

슬라이드 47~48 / 저장소 개인 Fork git-collab-번호. 로컬 실습 저장소와 별개입니다.

2-A. 작업 브랜치에서 첫 Push

main의 초기 상태에서 아래 예시 이름으로 브랜치를 만들고 Checkout합니다. 01을 본인 번호로 바꿉니다.

feature/intro-01

README의 기존 이름: **교육생** 줄만 본인 이름으로 수정합니다.

이름: 홍길동

README만 Stage하고 Commit합니다.

docs: 실습 참여자 이름 기록

Eclipse의 Push Branch 등 설치본의 메뉴로 Push합니다. origin이 자신의 Fork인지, 원격 브랜치가 feature/intro-번호인지 확인하고 첫 Push에서 upstream을 연결합니다. 여기서 upstream은 현재 로컬 브랜치가 추적할 원격 브랜치이며 원본 레포를 뜻하는 remote 이름과 구분합니다.

[] GitLab 자신의 Fork에 작업 브랜치 생성

[] 로컬·원격 Commit ID 일치

[] main은 초기 상태 유지

2-B. Fetch와 Pull 비교

GitLab에서 같은 feature/intro-번호 브랜치의 README 끝에 아래 내용을 추가하고 Commit합니다. 새 브랜치를 만들지 않습니다.

원격 메모: GitLab에서 추가한 내용을 Pull로 가져옵니다.

GitLab Commit 메시지:

docs: 원격 동기화 확인 메모 추가

1. Eclipse에서 현재 feature/intro-번호와 clean 상태를 확인합니다.

2. Fetch: 원격 추적 브랜치의 History는 갱신되지만 로컬 README에는 메모가 없어야 합니다.

3. Pull: 같은 브랜치에 원격 변경을 통합합니다. 수업은 Merge 방식입니다.

4. README에 메모가 생겼는지 확인합니다.

[] Fetch 후 파일 유지

[] Pull 후 원격 메모 반영

[] 현재 브랜치와 origin 설명 가능

3. 실습 03 — Feature Branch에서 두 번 Commit

슬라이드 55~57 / 저장소 개인 Fork git-collab-번호.

main을 Checkout하고 동기화한 뒤 clean 상태에서 다음 브랜치를 만듭니다. 실습 02의 intro 브랜치에서 만들지 않습니다.

feature/welcome-01

첫 번째 Commit — 출력 문구

App.java 전체를 다음으로 교체하고 저장·실행합니다.

```
package training;
```

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("환영합니다");  
    }  
}
```

Console의 **환영합니다**를 확인합니다. App.java만 Stage합니다.

feat: 실습 환영 문구 변경

두 번째 Commit — 실행 결과 기록

README의 기존 “실행 결과” 아래 **Git 실습 시작** 한 줄을 다음으로 교체합니다. README 전체를 덮어쓰지 않습니다.

환영합니다

README만 Stage합니다.

docs: 환영 문구 실행 결과 기록

작업 브랜치를 Push하고 main과 Diff를 비교합니다. 이 브랜치는 실습 06 MR에서 사용하므로 아직 main에 병합하지 않습니다.

[] 기능 변경·문서 변경 Commit 각 1개

[] main에는 변경 없음

[] GitLab feature 브랜치 확인

[] App 실행 결과와 README 일치

3-추가. 로컬 Fast-forward Merge

슬라이드 59 / 저장소 git-local-lab, 시작점 그 저장소의 C0.

History의 C0를 기준으로 아래 두 브랜치를 모두 생성합니다.

merge-lab

merge-feature

merge-feature를 Checkout하고 README 끝에 추가합니다.

메모: Fast-forward 병합을 확인합니다.

Commit 메시지:

docs: Fast-forward 확인 메모 추가

merge-lab을 Checkout한 뒤 merge-feature를 Merge합니다. Fast-forward 허용을 선택합니다.

[] 두 브랜치가 같은 새 Commit M1을 가리킴

[] 별도 Merge Commit 없음

[] 개인 Fork의 main에는 영향 없음

4. 실습 04 — 충돌 재현과 해결

슬라이드 64~66 / 저장소 git-local-lab / 시작점 C0. GitLab의 충돌 MR 촬영 예시와 별개로, 학생은 로컬에서 아래 과정을 수행합니다.

History에서 같은 C0를 기준으로 세 브랜치를 먼저 생성합니다.

conflict-a

conflict-b

conflict-target

4-A. 개발팀 변경

conflict-a를 Checkout하고 App.java 전체를 교체합니다.

```
package training;
```

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("개발팀 변경");  
    }  
}
```

Stage-Commit:

feat: 개발팀 문구 변경

4-B. 운영팀 변경

conflict-b를 Checkout합니다. 처음 문구가 [Git 실습 시작](#)인지 확인합니다. 개발팀 문구가 보이면 시작 브랜치를 잘못 만들었으므로 진행을 멈추고 C0를 확인합니다.

App.java 전체를 교체합니다.

```
package training;
```

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("운영팀 변경");  
    }  
}
```

Stage-Commit:

feat: 운영팀 문구 변경

4-C. 같은 대상에 순서대로 Merge

1. conflict-target을 Checkout합니다.

2. conflict-a를 먼저 Merge합니다.

3. conflict-b를 Merge하면 App.java에서 충돌이 발생합니다.

4. 비교 화면에서 양쪽 변경을 확인하고 최종 App.java를 다음으로 정리합니다. 충돌 표시 문자는 남기지 않습니다.

```
package training;
```

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("개발·운영 공동 변경");  
    }  
}
```

저장·실행 후 App.java를 Stage하고 병합 Commit을 작성합니다.

fix: 개발팀과 운영팀 문구 충돌 해결

- [] Console에 개발·운영 공동 변경
- [] 충돌 표시 해제, Working Tree clean
- [] History에서 해결 Commit의 부모가 2개
- [] 현재 브랜치 conflict-target

한 사람이 a·b 역할을 순서대로 수행해도 됩니다. 브랜치의 기준이 같아야 충돌이 재현됩니다.

5. 실습 05-A — Discard와 Mixed Reset

슬라이드 70 / 저장소 git-local-lab. 원격에 올리지 않는 reset-lab에서만 수행합니다.

5-A-1. Commit 전 수정 취소

App.java의 문구를 잠시 **취소할 변경**으로 수정·저장합니다. Stage·Commit하지 않고 해당 파일만 Discard합니다. 파일이 수정 전 내용으로 돌아오고 History는 그대로인지 확인합니다.

5-A-2. 미공유 Commit의 Mixed Reset

History에서 C0를 기준으로 브랜치를 생성합니다.

reset-lab

App.java 전체를 교체하고 Stage·Commit합니다.

package training;

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("Reset 연습 변경");  
    }  
}
```

feat: Reset 연습 문구 변경

이 새 Commit을 R1로 기록합니다. History에서 R1이 아닌 C0를 선택 → Reset → Mixed합니다.

[] reset-lab은 C0를 가리킴

[] App.java의 Reset 연습 변경 내용은 파일에 남음

[] App.java는 Unstaged 상태

파일을 다시 Stage하고 새 Commit을 작성합니다.

feat: Reset 후 문구 변경 다시 기록

새 Commit ID가 R1과 다른지 확인합니다. Hard Reset은 이 실습에서 실행하지 않습니다.

6. 실습 05-B — 공유한 Commit Revert

슬라이드 71~73 / 저장소 개인 Fork git-collab-번호.

최신 main에서 새 브랜치를 만들고 Checkout합니다. welcome 브랜치와 별개입니다.

feature/revert-01

README 끝에 연습용 잘못된 안내를 추가합니다.

잘못된 안내: main에 직접 Push하세요.

Commit 후 작업 브랜치를 Push합니다.

docs: 잘못된 안내 추가 (복구 연습)

이 일반 Commit을 V1로 기록합니다. History에서 V1을 선택하고 Revert Commit합니다. 만들어진 반대 변경 Commit을 V2로 기록하고 Push합니다.

[] V1과 V2 모두 History에 남음

[] 잘못된 안내 문구 삭제

[] README는 V1 이전 내용으로 복구

[] GitLab에서도 V2 확인

V2 자체의 Diff를 열면 삭제한 문구를 볼 수 있습니다. 초기 상태와 최종 상태만 비교하면 차이가 없습니다. 이번 실습은 부모가 하나인 일반 Commit을 되돌리며 Merge Commit Revert는 다루지 않습니다.

7. 실습 06 — MR, 짝 리뷰, 추가 Commit, 병합

슬라이드 80~85 / 저장소 개인 Fork git-collab-번호 / source 실습 03의 feature/welcome-번호 / target 같은 Fork의 main.

슬라이드 캡처는 App 변경 후 리뷰를 받고 README를 수정한 시연입니다. 학생은 실습 03에서 App과 README를 이미 두 Commit으로 기록했으므로, 이번에는 리뷰 의견을 반영하는 세 번째 Commit을 추가합니다.

7-A. MR 생성

GitLab에서 source-target project와 브랜치를 확인합니다. 제목 예시:

feat: 실습 환영 문구 변경

다음 MR 설명을 복사하고 실행 결과를 실제 확인한 내용으로 채웁니다.

변경 목적

교육용 프로그램의 환영 문구를 변경합니다.

주요 변경

- App.java 출력 문구: Git 실습 시작 → 환영합니다
- README 실행 결과를 변경된 출력과 일치시켰습니다.

검증 방법

- Eclipse에서 Run As → Java Application 실행
- 실제 Console 출력: [확인한 내용을 입력]
- 코드와 README 실행 결과 일치 여부: [확인 결과 입력]

리뷰 포인트

- 변경 파일이 App.java와 README.md인지 확인
- 출력 문구와 문서 내용이 일치하는지 확인

7-B. 리뷰와 수정 Commit

짝 리뷰어가 Changes를 보고 의견 1개를 남깁니다. 이번 순서에 맞는 의견 예시:

README에 실행 확인 방법과 확인 결과를 한 줄 더 남겨주세요.

작성자 같은 feature/welcome-번호에서 README 끝에 실제 확인 결과를 추가합니다. 다음 예시는 App을 실행하여 해당 출력이 나온 경우에 사용합니다.

검증: Eclipse에서 Run As > Java Application으로 실행하여 Console의 환영합니다를 확인했습니다.

README만 Stage·Commit·Push합니다.

docs: 리뷰 의견에 따라 실행 검증 기록 추가

리뷰어가 추가 Commit을 확인하고 스레드를 해결합니다. 댓글 해소와 강제 승인 규칙은 서로 다릅니다.

7-C. 병합과 로컬 동기화

권한 있는 담당자가 같은 Fork의 main에 MR를 Merge합니다. Eclipse에서 main을 Checkout하고 Pull합니다.

[] MR URL 기록

[] 리뷰 의견 1개와 수정 Commit

[] 병합 완료

[] 로컬 main에 환영 문구·문서 반영

작성자·리뷰어 접근권한과 병합 역할은 강사가 사전 배정합니다. Merge 버튼을 사용할 수 없으면 담당자에게 요청하고 main 직접 Push로 우회하지 않습니다.

8. 종합 실습 — feature → dev → stage → prod

슬라이드 88~98 / 저장소 팀 Fork git-promotion- 팀번호 / 예정 시간 42분.

팀별로 원본을 Fork하거나 강사가 배부한 팀 저장소를 사용합니다. 한 팀에서 작성자 한 명이 설정을 변경하고 나머지는 리뷰·병합 역할을 맡습니다. 각자 개인 Fork로 따로 승격하지 않습니다.

원본 주소:

<https://gitlab.musjung.com/git-training/git-promotion-01>

시작 전 dev:stage:prod가 같은 초기 Commit이고 설정 값이 모두 30인지 확인합니다. 강사는 환경 브랜치 직접 Push를 막고 팀원·병합 담당자 권한을 준비합니다. 이 교육 서버에서는 보호 브랜치 병합을 Maintainer에게 맡깁니다.

8-A. dev에서 feature 생성

팀 저장소를 Clone·Import합니다. origin/dev에서 로컬 dev를 생성·Checkout하고 동기화합니다. 최신 dev에서 다음 브랜치를 생성합니다.

feature/config-timeout

[config/application.yml](#)의 값 한 줄만 변경합니다. 아래는 변경 후 전체 내용입니다. 기존 주석을 유지합니다.

```
# Git 형상관리용 예제입니다. App.java가 읽는 설정은 아닙니다.  
timeoutSeconds: 60
```

Commit·Push:

chore: 실습 타임아웃을 60초로 변경

이 변경 Commit을 F1로 기록합니다.

8-B. MR 1 — feature/config-timeout → dev

MR 제목:

chore: 타임아웃 설정 변경을 dev에 반영

MR 설명:

```
## 변경 목적  
설정 파일의 timeoutSeconds를 30에서 60으로 변경합니다.
```

```
## 변경 범위  
config/application.yml의 설정 값 한 줄
```

```
## 검증  
- 전체 Diff에 30 → 60 변경만 있는지 확인  
- 병합 후 dev 브랜치 파일 값 확인  
- 실제 서비스 동작이나 배포 검증은 포함하지 않습니다.
```

리뷰·병합 후 dev의 파일 값을 확인하고 MR에 검증 댓글을 남깁니다.

검증: dev의 config/application.yml에서 timeoutSeconds: 60을 확인했습니다. stage와 prod는 아직 30입니다.

8-C. MR 2 — dev → stage

새 MR의 source는 dev, target은 stage입니다. 전체 Diff를 확인하고 병합합니다.

chore: dev 설정 변경을 stage에 승격

병합 후 실제 값을 확인하고 댓글을 남깁니다.

검증: stage의 config/application.yml에서 timeoutSeconds: 60을 확인했습니다. dev도 60이며 prod는 아직 30입니다.

8-D. MR 3 — stage → prod

새 MR의 source는 stage, target은 prod입니다. 담당자가 전체 Diff를 검토하고 병합합니다.

chore: stage 설정 변경을 prod에 승격

실제 확인 후 댓글:

검증: prod의 config/application.yml에서 timeoutSeconds: 60을 확인했습니다. dev, stage, prod 모두 60입니다.

세 MR 모두 Squash를 해제하고 source branch 삭제를 해제합니다. dev:stage:prod를 삭제하지 않습니다. 각 브랜치의 끝 Commit ID는 Merge Commit 때문에 달라질 수 있지만 원래 변경 F1은 포함되어야 합니다.

완료 시점	dev	stage	prod
초기	30	30	30
MR 1 병합	60	30	30
MR 2 병합	60	60	30
MR 3 병합	60	60	60

[] MR 3개 URL와 검증 댓글 기록

[] 대상 브랜치 파일 값 확인

[] 각 History에 F1 포함

[] prod 직접 Push 없음

[] 환경 브랜치 유지

YAML은 형상관리용 예제입니다. Java 프로그램이 이 파일을 읽지 않으므로 timeout 동작이 바뀐다고 설명하지 않습니다. prod 병합은 서버 배포 완료가 아닙니다.

9. 완료 기록과 막혔을 때 확인

완료 기록 양식을 복사해 자신의 메모에 작성합니다. 토큰은 기록하지 않습니다.

수강생 번호:

개인 Fork URL:

로컬 C0 Commit ID:

첫 Push Commit ID:

충돌 해결 Commit ID:

Revert 전 V1:

Revert 후 V2:

리뷰·병합 MR URL:

팀 저장소 URL:

설정 변경 F1:

feature → dev MR URL:

dev → stage MR URL:

stage → prod MR URL:

- 인증 실패: 개인 Fork HTTPS URL, 계정, 토큰 만료·write_repository 범위를 확인합니다.
- Push rejected: 현재 브랜치, 원격 변경, 보호 규칙을 확인합니다. Force Push로 우회하지 않습니다.
- 브랜치 전환 실패: 남아 있는 파일 변경을 확인하고 강사에게 현재 상태를 보여줍니다.
- 충돌이 안 생김: conflict-a·b·target이 모두 같은 C0에서 시작했는지 확인합니다.
- MR에 예상 밖의 Diff: source·target project와 브랜치, 기준 Commit을 확인합니다.
- Merge 버튼 사용 불가: 병합 담당자와 권한을 확인합니다.
- Eclipse Java 오류: package training, 파일 경로 src/training/App.java, 프로젝트 JRE·Build Path를 확인합니다.
- GitLab 접속 실패: 교육장 공인 IP가 서버 접근 허용 범위에 포함됐는지 강사에게 확인합니다.

부록. 직접 프로젝트를 만들 때 사용할 초기 파일

강사가 배부한 starter에 아래 파일이 이미 있으면 다시 덮어쓰지 않습니다. 초기화는 첫 실습 전에만 수행합니다.

폴더 구조:

```
git-local-lab/  
├── src/training/App.java  
├── config/application.yml  
├── README.md  
├── .project  
├── .classpath  
└── .gitignore
```

src/training/App.java

```
package training;  
  
public class App {  
    public static void main(String[] args) {  
        System.out.println("Git 실습 시작");  
    }  
}
```

config/application.yml

```
# Git 형상관리용 예제입니다. App.java가 읽는 설정은 아닙니다.  
timeoutSeconds: 30
```

README.md

Git 형상관리 실습

이름: 교육생

Eclipse에서 App.java를 열고 Run As > Java Application을 선택합니다.
외부 라이브러리는 없습니다.

실행 결과

Git 실습 시작

config/application.yml은 설정 변경 이력 연습용이며 프로그램 동작에 연결되지 않습니다.

.project

파일명 앞의 점을 포함합니다. 이 로컬 프로젝트 이름은 git-local-lab입니다.

```

<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
  <name>git-local-lab</name>
  <comment>Git version control practice</comment>
  <projects></projects>
  <buildSpec>
    <buildCommand>
      <name>org.eclipse.jdt.core.javabuilder</name>
      <arguments></arguments>
    </buildCommand>
  </buildSpec>
  <natures>
    <nature>org.eclipse.jdt.core.javanature</nature>
  </natures>
</projectDescription>

```

.classpath

```

<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="output" path="bin"/>
</classpath>

```

.gitignore

```

/bin/
/.metadata/
*.class
*.log
.DS_Store
.env
*.pem
*.key
*accessKeys.csv

```