T-Patch 메뉴얼

이 문서는 `tpatch` 실행 파일로 Tibero 패치를 수행하는 운영자와 패치 담당자를 위한 메뉴얼입니다.

> 이 문서의 모든 주소, 계정, 비밀번호 및 경로는 설명을 위한 예시입니다. 실제 운영 환경의 값으로 사용하지 마십시오.
> **필수 확인:** T-Patch 수행 전에 기존 `TB_HOME` 백업이 필요하면 운영자가 직접 백업해야 합니다. T-Patch가 기존 `TB_HOME` 전체를 자동 백업하지 않습니다.
> **필수 확인:** T-Patch 수행 후에는 각 대상 인스턴스의 `slog`를 확인하여 오류 발생 여부를 반드시 점검해야 합니다.

1. T-Patch는 무엇인가

T-Patch는 Tibero 패치 과정에서 반복적으로 필요한 작업을 정해진 절차에 따라 실행하는 패치 자동화 도구입니다.

Tibero 패치는 단순히 바이너리 파일만 교체하는 작업이 아닙니다. 패치 전후로 DB/TAC, TAS, CM, SSVR 상태를 확인하고, 필요한 시점에 SQL 또는 shell task를 실행하고, 경우에 따라 엔진 홈을 교체하고, 여러 노드의 순서를 맞춰야 합니다. 이 작업을 사람이 매번 순서대로 수행하면 누락, 순서 오류, 노드별 편차가 생기기 쉽습니다.

T-Patch는 이 문제를 줄이기 위해 다음 작업을 한 흐름으로 묶어 실행합니다.

- 패치를 적용할 서버와 인스턴스를 정합니다.
- 패치 전후에 필요한 SQL, shell, 수동 확인 작업을 단계에 맞춰 실행합니다.
- 작업별 실행 여부와 사용자 확인을 처리합니다.
- 바이너리 교체 흐름을 관리합니다.
- 여러 노드 작업을 한 곳에서 제어하고 로그와 입력을 모읍니다.

2. 왜 필요한가

T-Patch가 필요한 이유는 세 가지입니다.

첫째, 패치 절차를 실행 가능한 형태로 고정합니다. 패치마다 필요한 SQL, shell, 수동 확인 task를 `tasks.json`으로 받아 정해진 단계에 실행하므로, 작업자가 문서를 보며 순서를 맞추는 부담이 줄어듭니다.

둘째, 패치 대상의 상태 전환을 일관되게 처리합니다. 패치 전에는 DB/TAC, TAS, CM, SSVR을 필요한 순서로 내리고, 바이너리 교체 후에는 반대 순서로 올리면서 각 단계의 task를 실행합니다.

셋째, 단일 서버뿐 아니라 여러 노드의 원격 패치를 한 곳에서 제어할 수 있습니다. control/agent 모드를 사용하면 Control Plane 서버에서 각 대상 노드의 agent와 통신하며 패치 흐름, 로그, 사용자 입력을 모읍니다.

3. 실행 방식 선택

T-Patch는 빌드된 실행 파일 하나를 세 가지 모드로 실행합니다.

모드	실행 명령	용도
standalone	`./tpatch --mode standalone`	`tpatch`를 실행한 로컬 서버에서 직접 패치
control	`./tpatch --mode control`	Control Plane에서 여러 대상 노드의 agent를 제어
agent	`./tpatch --mode agent --port 8081`	대상 노드에서 실제 패치 작업을 수행하는 agent 서버

`--mode`를 생략하면 기본값은 `standalone`입니다.

standalone을 쓰는 경우

대상 DB가 `tpatch`를 실행하는 서버에 있고, 로컬에서 바로 패치해도 되는 경우에 사용합니다. 가장 단순한 방식입니다.

```
./tpatch --patch
./tpatch --rollback
./tpatch --patch --rolling
```

control/agent를 쓰는 경우

패치 대상이 여러 서버에 흩어져 있거나, 한 서버에서 전체 노드의 진행 상황과 사용자 입력을 모아 처리하고 싶을 때 사용합니다.

Control Plane에서 실행:

```
./tpatch --patch --mode control
./tpatch --rollback --mode control
./tpatch --patch --mode control --rolling
```

대상 노드에서 agent를 미리 실행:

```
./tpatch --mode agent --port 8081
```

agent가 미리 떠 있지 않으면 control 모드는 `AGENT_INSTALL_*` 설정을 사용해 원격 서버에 현재 `tpatch` 바이너리를 업로드하고 agent를 기동하려고 시도합니다.

4. T-Patch가 제공하는 주요 기능

패치와 롤백 실행

`--patch`는 `tasks.json`의 패치 task를 기준으로 패치 절차를 수행합니다. `--rollback`은 패치 절차가 완료된 시점을 기준으로 `rollback_tasks`에 정의된 롤백 절차를 수행합니다.

> **롤백 기준:** 이 문서에서 롤백은 패치 절차가 완료된 시점을 기준으로 수행하는 롤백을 의미합니다.

현재 CLI 기준으로 `--patch`와 `--rollback`은 파일 경로 인자를 받지 않습니다. TOBE 바이너리 경로는 `setting.yaml`의 `BINARY_FILE_PATH`에 작성합니다.

```
./tpatch --patch
./tpatch --rollback
```

non-rolling과 rolling

non-rolling은 전체 노드에 대해 패치 전 작업과 바이너리 교체를 묶어서 진행한 뒤, 모든 노드가 바이너리 교체를 완료한 상태에서 패치 후 작업을 수행하는 방식입니다. 노드 간 패치 작업이 비동기로 수행되므로 전체 패치 시간은 짧아질 수 있지만 서비스 중단 범위가 큽니다.

rolling은 노드 하나의 `pre_patch -> apply_patch -> post_patch`를 완료한 뒤 다음 노드로 넘어가는 방식입니다. 노드 간 패치 작업이 순차적으로 수행되므로 전체 시간은 길어질 수 있지만, 다른 노드는 운영 상태를 유지할 수 있습니다.

```
# non-rolling 패치
./tpatch --patch

# rolling 패치
./tpatch --patch --rolling
```

> **확인:** rolling과 non-rolling은 노드별 수행 task가 다를 수 있으므로 적용할 `tasks.json`과 패치 절차를 사전에 확인합니다.

task 실행

T-Patch는 `tasks.json`에 정의된 task를 단계별로 실행합니다.

task type	의미

`SQL`	`tbsql`로 SQL 또는 SQL 파일 실행
`SHELL`	shell script 또는 shell 명령 실행
`MANUAL`	사용자가 직접 확인해야 하는 작업으로 처리

task는 `order`, `order_add`, `type`, `user`, `task`, `taskIdx`, `is_conditional` 같은 값으로 실행 위치와 조건이 결정됩니다.

SHELL task의 노드 실행 범위

다중 노드 환경에서 SHELL task의 일부 명령을 전체 노드, 첫 번째 노드 또는 마지막 노드에만 실행하도록 지정할 수 있습니다.

```
tpatch_task_execute_node ALL
echo "모든 노드에서 실행"
tpatch_task_execute_node_end

tpatch_task_execute_node FIRST
echo "첫 번째 노드에서만 실행"
tpatch_task_execute_node_end

tpatch_task_execute_node LAST
echo "마지막 노드에서만 실행"
tpatch_task_execute_node_end
```

- `ALL` : 모든 노드에서 실행
- `FIRST` : 첫 번째 노드에서만 실행
- `LAST` : 마지막 노드에서만 실행
- 모든 `order_add` 단계에서 사용 가능
- 블록 중첩은 허용하지 않음
- 시작과 종료 표식을 반드시 함께 작성
- 블록 밖 명령은 기존 `order` / `order_add` 노드 배치 규칙을 따름

대상 노드에 실행할 명령이 없으면 해당 task는 실행하지 않습니다.

조건부 task

모든 환경에서 항상 필요한 task가 아니라면 조건부 task로 둘 수 있습니다.

`CONDITIONAL_TASK_EXECUTION`은 `taskIdx`별 실행 여부를 지정합니다.

```
OPTION:
  -
    CONDITIONAL_TASK_EXECUTION:
      884: true
      759: false
```

이 설정은 조건부 task에만 적용됩니다.

task 변수 치환

task에 적용된 변수를 `setting.yaml`에서 미리 변환할 수 있습니다. (T-Patch 수행시 프롬프트에서 입력 받는 방식으로도 변경할 수 있습니다.)

task 예시:

```
{
  "taskIdx": 1092,
  "task": "SELECT * FROM {{table_name:t1}} WHERE owner = '{{schema_name:SYS}}';"
}
```

`setting.yaml` 예시:

```yaml
OPTION:
  -
    TASK_VARIABLES:
      1092:
        table_name: T_PATCH_TEST
        schema_name: SYS
```

변수 형식은 `{{변수명:기본값}}` 입니다. 기본값 없는 `{{변수명}}` 형식은 사용하지 않습니다.

바이너리 교체 자동화

`BINARY_REPLACE_AUTO: true` 를 사용하면 T-Patch가 교체할 바이너리의 압축을 풀고 AS-IS 홈의 기본 설정 파일을 TO-BE 홈으로 반영하는 흐름을 수행합니다.

필수 조건:

- 각 대상 노드에 `BINARY_FILE_PATH` 가 있어야 합니다.
- `ASIS_TB_HOME` 과 `TOBE_TB_HOME` 이 모두 있어야 합니다.
- `ASIS_TB_HOME` 과 `TOBE_TB_HOME` 은 서로 달라야 합니다.

기본적으로 바이너리 교체 자동화는 TO-BE 홈에 새 바이너리를 준비한 뒤, AS-IS 홈에서 운영 환경에 필요한 기본 파일을 TO-BE 홈으로 반영합니다.

기본 파일과 `BINARY_REPLACE_EXTRA_TARGETS` 는 AS-IS 홈에서 제거하지 않고 `cp -p` 방식으로 TO-BE 홈에 복사합니다. 따라서 AS-IS 홈에 원본이 남으며 파일 권한과 수정 시각이 보존됩니다.

기본 반영 대상:

AS-IS 홈 기준 원본	TO-BE 홈 기준 반영 위치	설명
`config/*.tip`	`config`	Tibero 인스턴스 설정 파일
`config/lsnr_invited.lst`	`config/lsnr_invited.lst`	listener invited list
`client/config/tbdsn.tbr`	`client/config/tbdsn.tbr`	client 접속 설정
`client/config/*.cfg`	`client/config`	client 설정 파일
`license/*.xml`	`license`	라이선스 파일

위 파일들은 존재하면 반영하고, 없는 파일은 기본적으로 건너뜁니다. 환경별로 반드시 필요한 파일이 기본 대상에 없다면 아래의 `BINARY_REPLACE_EXTRA_TARGETS` 로 추가합니다.

예시:

```yaml
DBSVR:
  -
    ASIS_TB_HOME: /app/tibero
    TOBE_TB_HOME: /app/tibero_new
    BINARY_FILE_PATH: /app/binary/tibero_patch.tar.gz
    TAC_NAME: tibero
    TAC_USER: "<DB_USER>"
    TAC_PW: "<DB_PASSWORD>"

OPTION:
  -
    TASK_JSON: tasks.json
    CHECK_TASKS: false
    BINARY_REPLACE_AUTO: true
```

추가로 복사해야 하는 파일이 있으면 `BINARY_REPLACE_EXTRA_TARGETS` 를 사용합니다. `source` 는 `ASIS_TB_HOME` 기준 상대 경로이고, `target` 은 `TOBE_TB_HOME` 기준 상대 경로입니다.

`required` 는 원본 파일이 없을 때 바이너리 교체를 실패 처리할지 결정합니다.

- `required: false` : 원본 파일이 없으면 로그만 남기고 건너뜁니다.

- `required: true` : 원본 파일이 없으면 오류로 처리하고 바이너리 교체를 중단합니다.

대부분의 환경별 선택 파일은 `false` 로 두고, 반드시 TO-BE 홈에 반영되어야 하는 파일만 `true` 로 지정합니다.

```
OPTION:
  -
    BINARY_REPLACE_EXTRA_TARGETS:
      - source: "client/epa/java/config/epa.cfg"
        target: "client/epa/java/config/epa.cfg"
        required: false
```

자동 기동 옵션

`TBBOOT_AUTO` 는 패치 흐름 중 인스턴스 기동/종료 자동화 여부를 제어하기 위한 옵션입니다.

```
OPTION:
  -
    TBBOOT_AUTO: true
```

운영 절차상 기동/종료를 직접 통제해야 하는 환경에서는 이 값을 명시적으로 검토한 뒤 사용합니다.

root 실행 환경에서 OS 사용자 전환

패치 대상 인스턴스가 root 계정으로 실행되고 있다면 T-Patch 또는 agent도 반드시 root 계정으로 실행해야 합니다. root 권한이 아닌 계정으로 T-Patch를 실행하면 해당 인스턴스의 task, 상태 확인, 기동 및 종료 작업을 정상적으로 수행할 수 없습니다.

root로 실행한 T-Patch 또는 agent에서 TAC(DB)와 TAS 관련 작업을 별도의 OS 운영 계정으로 수행해야 할 때는 `USER_SWITCH` 를 사용합니다. `USER_SWITCH` 에 계정을 지정하면 T-Patch 또는 agent의 실행 권한은 root로 유지하면서 TAC(DB)/TAS의 task, 상태 확인, `tbboot` / `tbdown` 명령만 지정한 OS 계정으로 분리하여 실행합니다. 모든 작업을 root로 수행해야 한다면 `USER_SWITCH` 를 설정하지 않습니다.

```
OPTION:
  -
    USER_SWITCH: "<OS_USER>"
```

내부적으로 `su - <OS_USER> -c '<COMMAND>'` 형태의 login shell을 사용합니다. 그렇기 때문에, T-Patch 프로세스가 실행되는 계정에서 다른 계정으로 user switching이 허용되어야합니다.

`<OS_USER>` 와 `<COMMAND>` 는 설명용 자리표시자이며, 실제 환경에 맞는 값으로 바꿔야 합니다. 사용자 전환은 TAC(DB)와 TAS에만 적용되며, CM, SSVR, 바이너리 교체 작업은 root 실행 컨텍스트를 유지합니다.

모드별 필수 조건은 다음과 같습니다.

- root 계정으로 실행되는 인스턴스를 standalone 모드로 패치할 때는 `tpatch` 를 root로 실행해야 합니다.
- root 계정으로 실행되는 인스턴스를 control 모드로 패치할 때는 대상 노드의 agent를 root로 실행해야 합니다.
- control 모드에서 해당 agent를 자동 설치한다면 `AGENT_INSTALL_USER: root` 가 필수입니다.
- `USER_SWITCH` 를 사용한다면 전환 대상 계정에서 TB_HOME 접근과 Tibero 명령 실행이 가능해야 합니다.

`USER_SWITCH` 가 비어 있거나 T-Patch/agent가 root가 아니면 사용자 전환이 적용되지 않습니다.

Wallet 및 HSM 자동 기동

Wallet 적용 DB를 자동 기동할 때 다음 옵션을 사용할 수 있습니다.

```
OPTION:
  -
    TBBOOT_AUTO: true
    WALLET_AUTO: true
    WALLET_PASSWORD: "<WALLET_PASSWORD>"
    USE_HSM: false
    HSM_PIN: ""
```

`WALLET_AUTO: true`이면 TAC(DB)의 normal boot 단계에서 `tbboot -w`를 사용하고, mount 단계에서는 wallet open SQL을 실행합니다. Wallet 자동 기동은 TAC(DB)에만 적용됩니다.

로컬 대화형 실행에서는 `WALLET_PASSWORD`나 `HSM_PIN`이 비어 있으면 실행 중 입력할 수 있습니다. control/agent, SSH, CI/CD처럼 프롬프트 입력이 불가능한 모드에서는 `WALLET_AUTO: true`일 때 `WALLET_PASSWORD`가 필수이며, `USE_HSM: true`이면 `HSM_PIN`도 작성해야 합니다.

HSM을 사용하면 각 노드의 `$TB_HOME/config/$TB_SID.tip`에서 `HSM_LIB_FILE`을 읽어 HSM 장비를 판별합니다.

`HSM_LIB_FILE`	장비
`libsgkms_crytoki.so`	DAMO
`libvorpkcs11.so`	VORMETRIC
`libpkcs11KSignKMS.so`	KSIGN

TIP 파일을 읽을 수 없거나 지원하지 않는 `HSM_LIB_FILE`이면 패치 절차에 들어가기 전 사전 검증에서 중단합니다.

SSVR 지원

`SSVR` 섹션을 작성하면 DB 서버 외 SSVR 대상도 패치 흐름에 포함할 수 있습니다.

```yaml
SSVR:
  -
    ASIS_TB_HOME: /app/tibero_ssvr
    TOBE_TB_HOME: /app/tibero_ssvr_new
    BINARY_FILE_PATH: /app/binary/tibero_patch.tar.gz
    SSVR_NAME: tibero_ssvr
    SSVR_USER: "<SSVR_USER>"
    SSVR_PW: "<SSVR_PASSWORD>"
```

SSVR 단계는 DB/TAC, TAS, CM과 별도 단계로 처리됩니다.

DBSVR와 SSVR가 같은 물리 호스트에 있으면 `ASIS_TB_HOME` 및 `TOBE_TB_HOME`을 서로 다르게 설정해야 합니다. 서로 다른 호스트라면 동일한 TB_HOME 경로 문자열을 사용할 수 있습니다.

5. 필요한 파일 및 권한, 확인이 필요한 DB 옵션

기본 작업 디렉터리에는 다음 파일이 필요합니다.

```
작업 디렉터리/
├── tpatch
├── setting.yaml
└── tasks.json
```

`BINARY_REPLACE_AUTO` 또는 CI 방식의 바이너리 교체를 사용한다면 각 노드의 `BINARY_FILE_PATH`가 실제 교체할 바이너리 압축 파일을 가리켜야 합니다.

```
/app/binary/tibero_patch.tar.gz
```

control 모드에서 agent를 원격 bootstrap하려면 Control Plane 서버에 실행 중인 `tpatch` 바이너리와 대상 서버 SSH 접속 정보가 필요합니다.

보안 주의사항

- 외부 배포본과 예제 파일에는 실제 DB 계정, SSH 비밀번호, Wallet 비밀번호, HSM PIN, 서버 주소 또는 고객 식별정보를 작성하지 않습니다.
- 실제 값이 포함된 `setting.yaml`은 소스 저장소, 메신저, 이메일 및 공용 파일 저장소에 업로드하지 않습니다.
- `setting.yaml`은 패치 담당자만 읽을 수 있도록 최소 권한을 적용합니다. Linux에서는 `chmod 600 setting.yaml`을 권장합니다.
- 비밀정보가 노출되었거나 외부로 전달된 경우 해당 비밀번호와 PIN을 즉시 변경하고 접근 기록을 점검합니다.

- agent 포트는 인터넷이나 불필요한 네트워크에 공개하지 않습니다. Control Plane의 주소만 접근할 수 있도록 방화벽과 네트워크 접근 제어를 적용합니다.
- SSH bootstrap에 사용하는 계정과 비밀번호는 승인된 보안 채널로만 전달하고, 패치 종료 후 불필요한 사본을 폐기합니다.
- `tpatch.log`와 agent 로그는 운영 경로, SID 및 작업 결과를 포함할 수 있으므로 설정 파일과 동일하게 접근 권한과 반출 여부를 관리합니다.

권한 확인

> ❗ 패치 실행 전에는 실행 파일, 설정 파일, task 파일, 로그 디렉터리, 바이너리 압축 파일에 대한 권한을 확인합니다.

```
# tpatch 실행 파일 확인
ls -la ./tpatch
chmod +x ./tpatch

# 설정 파일과 task 파일 읽기 권한 확인
ls -la ./setting.yaml
chmod 600 ./setting.yaml
ls -la ./tasks.json

# 로그 디렉터리 쓰기 권한 확인
ls -ld /app/logs

# 교체할 바이너리 압축 파일 읽기 권한 확인
ls -la /app/binary/tibero_patch.tar.gz
```

`LOG_DIR`을 별도로 지정했다면 해당 디렉터리에 쓰기 권한이 있어야 합니다. 바이너리 교체 자동화를 사용하는 경우에는 `ASIS_TB_HOME`의 기본 반영 대상 파일을 읽고 복사할 수 있어야 하며, `TOBE_TB_HOME`에는 압축 해제와 파일 반영이 가능해야 합니다.

root/user switching 사전 점검

`USER_SWITCH`를 사용할 때는 대상 노드에서 다음 항목을 확인합니다.

아래의 `<OS_USER>`는 설명용 자리표시자입니다. 명령을 그대로 복사하지 말고 승인된 전환 대상 OS 계정으로 바꿔 실행합니다.

```
# T-Patch 또는 agent가 root로 실행되는지 확인
id -u

# 대상 계정으로 전환할 수 있는지 확인
su - <OS_USER> -c 'id && command -v tbboot && command -v tbsql'

# 대상 계정의 TB_HOME 접근 권한 확인
su - <OS_USER> -c 'test -r /app/tibero/config/tb.tip && test -x /app/tibero/bin/tbboot'
```

> ❗ `USER_SWITCH`를 사용하려면 T-Patch 또는 agent를 root로 실행해야 합니다.
> root 계정 사용이 금지되어 있거나 `su`, PAM 또는 보안 정책으로 root에서 대상 OS 계정으로의 user switching이 차단된 환경은 지원하지 않습니다.
> CM, SSVR, 바이너리 교체 작업에는 사용자 전환이 적용되지 않습니다.

네트워크 조건

standalone 모드는 로컬 서버에서 직접 실행하므로 별도 agent 통신 포트가 필요하지 않습니다.

> ❗ control 모드를 사용할 때는 Control Plane 서버에서 각 대상 노드의 agent 주소로 접속할 수 있어야 합니다.

- Control Plane 서버에서 대상 노드의 `AGENT_HOST:AGENT_PORT`로 접속 가능해야 합니다.
- `AGENT_PORT`를 비워두면 기본값 `8081`을 사용합니다.
- 대상 노드 OS 방화벽과 외부 방화벽에서 agent TCP 포트가 허용되어 있어야 합니다.

- `AGENT_HOST`를 비워두면 `AGENT_INSTALL_IP`를 agent 접속 주소로 사용합니다.

> agent가 미리 실행되어 있지 않아 Control Plane이 원격 bootstrap을 수행해야 하는 경우에는 SSH 접속 조건도 필요합니다.

- Control Plane 서버에서 대상 노드의 `AGENT_INSTALL_IP:AGENT_INSTALL_PORT`로 접속 가능해야 합니다.
- `AGENT_INSTALL_PORT`를 비워두면 기본값 `22`를 사용합니다.
- `AGENT_INSTALL_USER`, `AGENT_INSTALL_PASSWORD`로 SSH 로그인이 가능해야 합니다.

`AGENT_INSTALL_DIR`에 파일 업로드, 실행 권한 부여, agent 실행이 가능해야 합니다.

CM Tip 확인

> T-Patch는 자동으로 DB 기동상태를 변경하고, 이에 맞춰 태스크를 수행하는 역할입니다.
> CM Param중 tas나 db를 auto booting 하는 옵션이 꺼져있는지 **반드시 확인이 필요합니다.**

- Param명 : CM_SVC_AUTO_START_ON_BOOT
- 위 Param이 Y면 반드시 N 상태로 변경 후 패치를 진행해주세요.
- **(추후) T-Patch 기능으로 추가 예정.**

6. setting.yaml 작성

기본 standalone 예시

```yaml
DBSVR:
  -
    ASIS_TB_HOME: /app/tibero
    TOBE_TB_HOME: /app/tibero_new
    BINARY_FILE_PATH: /app/binary/tibero_patch.tar.gz
    TAC_NAME: tibero
    TAC_USER: "<DB_USER>"
    TAC_PW: "<DB_PASSWORD>"
    TAS_NAME:
    TAS_USER:
    TAS_PW:
    CM_NAME:
    CM_HOME:

OPTION:
  -
    TASK_JSON: tasks.json
    CHECK_TASKS: false
    LOG_DIR: /app/logs
    SYSLOG_ERROR_DETAIL: true
    ROLLBACK_OP: false
    TBBOOT_AUTO: false
    USER_SWITCH:
    WALLET_AUTO: false
    WALLET_PASSWORD:
    USE_HSM: false
    HSM_PIN:
    BINARY_REPLACE_AUTO: false
    TASK_VARIABLES:
    CONDITIONAL_TASK_EXECUTION:
```

control 모드 예시

```yaml
DBSVR:
  -
    ASIS_TB_HOME: /app/tibero1
```

```
    TOBE_TB_HOME: /app/tibero1_new
    BINARY_FILE_PATH: /app/binary/tibero_patch.tar.gz
    TAC_NAME: tibero1
    TAC_USER: "<DB_USER>"
    TAC_PW: "<DB_PASSWORD>"
    # Agent가 떠있다면 아래와 같이 AGENT_* 작성
    AGENT_HOST: 192.0.2.10
    AGENT_PORT: 8081
    # Agent를 실행되어있지 않다면 아래와 같이 AGENT_INSTALL_* 작성
    AGENT_INSTALL_IP: 192.0.2.10
    AGENT_INSTALL_PORT: 22
    AGENT_INSTALL_USER: root
    AGENT_INSTALL_PASSWORD: "<SSH_PASSWORD>"
    AGENT_INSTALL_DIR: /tmp/tpatch-agent
  -
    ASIS_TB_HOME: /app/tibero2
    TOBE_TB_HOME: /app/tibero2_new
    BINARY_FILE_PATH: /app/binary/tibero_patch.tar.gz
    TAC_NAME: tibero2
    TAC_USER: "<DB_USER>"
    TAC_PW: "<DB_PASSWORD>"
    # Agent가 떠있다면 아래와 같이 AGENT_* 작성
    AGENT_HOST: 192.0.2.11
    AGENT_PORT: 8081
    # Agent를 실행되어있지 않다면 아래와 같이 AGENT_INSTALL_* 작성
    AGENT_INSTALL_IP: 192.0.2.11
    AGENT_INSTALL_PORT: 22
    AGENT_INSTALL_USER: root
    AGENT_INSTALL_PASSWORD: "<SSH_PASSWORD>"
    AGENT_INSTALL_DIR: /tmp/tpatch-agent

OPTION:
  -
    TASK_JSON: tasks.json
    CHECK_TASKS: false
    LOG_DIR: /app/logs
    SYSLOG_ERROR_DETAIL: true
    ROLLBACK_OP: false
    TBBOOT_AUTO: false
    # USER_SWITCH를 사용하면 agent가 root로 실행되어야 함
    USER_SWITCH: "<OS_USER>"
    WALLET_AUTO: false
    WALLET_PASSWORD:
    USE_HSM: false
    HSM_PIN:
    BINARY_REPLACE_AUTO: false
    TASK_VARIABLES:
    CONDITIONAL_TASK_EXECUTION:
```

주요 설정 항목

항목	위치	필수 여부	설명	예시
`ASIS_TB_HOME`	`DBSVR`, `SSVR`	항상 필수	현재 사용 중인 Tibero 홈	`/app/tibero`
`TOBE_TB_HOME`	`DBSVR`, `SSVR`	바이너리 교체 자동화 사용 시 필수	교체 후 사용할 Tibero 홈	`/app/tibero_new`
`BINARY_FILE_PATH`	`DBSVR`, `SSVR`	바이너리 교체 자동화 사용 시 필수	교체할 바이너리 압축 파일 전체 경로	`/app/binary/tibero_patch.t`

`TAC_NAME`, `TAS_NAME`, `CM_NAME`	`DBSVR`	`TAC_NAME`은 항상 필수, `TAS_NAME` / `CM_NAME`은 해당 구성 사용 시 필요	DB/TAC, TAS, CM 대상 SID	`tibero`, `tibero_tas`, `tiber`
`TAC_USER`, `TAC_PW`	`DBSVR`	항상 필수	DB/TAC SQL task 실행 계정과 비밀번호	`<DB_USER>` / `<DB_PASSWORD>`
`TAS_USER`, `TAS_PW`	`DBSVR`	TAS 구성 사용 시 필요	TAS SQL task 실행 계정과 비밀번호	`<DB_USER>` / `<DB_PASSWORD>`
`CM_HOME`	`DBSVR`	CM 구성 사용 시 필요	CM 사용 시 CM 홈 경로	`/app/tibero_cm`
`SSVR_NAME`, `SSVR_USER`, `SSVR_PW`	`SSVR`	`SSVR_NAME`, `SSVR_USER` / `SSVR_PW`은 SSVR 섹션 작성 시 필수	SSVR 사용 시 접속 정보, NAME은 SSVR SID를 의미	`<SSVR_SID>`, `<SSVR_USER>`, `<SSVR_PASSWORD>`
`TASK_JSON`	`OPTION`	항상 필수	패치 task JSON 파일 경로	`tasks.json`
`CHECK_TASKS`	`OPTION`	항상 필수	task별 실행 전 사용자 확인 여부	`false`
`LOG_DIR`	`OPTION`	선택	`tpatch.log` 저장 디렉터리	`/app/logs`
`SYSLOG_ERROR_DETAIL`	`OPTION`	선택	syslog 에러 전문을 콘솔/프롬프트에 표시할지 여부	`false`
`TASK_VARIABLES`	`OPTION`	task 변수를 사용할 때 필요	task 변수 값	`{1092: {table_name: t1}}`
`CONDITIONAL_TASK_EXECUTION`	`OPTION`	조건부 task 실행 여부를 지정할 때 필요	조건부 task 실행 여부	`{884: true, 759: false}`
`TBBOOT_AUTO`	`OPTION`	선택	기동/종료 자동화 사용 여부	`true`
`USER_SWITCH`	`OPTION`	선택	root 실행 상태에서 TAC/TAS 명령을 수행할 OS 계정	`<OS_USER>`
`WALLET_AUTO`	`OPTION`	선택	TAC(DB) 자동 기동 시 wallet open 수행 여부	`true`

`WALLET_PASSWORD`	`OPTION`	비대화형 Wallet 자동 기동 시 필수	Wallet 비밀번호	`<SECRET>`
`USE_HSM`	`OPTION`	선택	HSM 연동 wallet open 여부	`true`
`HSM_PIN`	`OPTION`	비대화형 HSM 사용 시 필수	HSM PIN	`<SECRET>`
`BINARY_REPLACE_AUTO`	`OPTION`	바이너리 교체를 자동화할 때 사용	바이너리 교체 자동화 사용 여부	`true`
`BINARY_REPLACE_EXTRA_TARGETS`	`OPTION`	기본 반영 대상 외 추가 파일이 있을 때 사용	추가 반영 파일 목록	`[{source: ".cube", target: ".cube", required: false}]`

7. 실행 절차

단일 노드 패치

```
cd /opt/tpatch
./tpatch --version
./tpatch --patch
```

단일 노드 롤백

```
cd /opt/tpatch
./tpatch --rollback
```

다중 노드 rolling 패치

```
cd /opt/tpatch
./tpatch --patch --rolling
```

control 모드 패치

agent를 미리 띄우는 방식:

```
# 각 대상 노드
./tpatch --mode agent --port 8081

# Control Plane 서버
./tpatch --patch --mode control
```

agent bootstrap을 맡기는 방식:

```
# Control Plane 서버
./tpatch --patch --mode control
```

이 경우 `setting.yaml`의 `AGENT_INSTALL_IP`, `AGENT_INSTALL_PORT`, `AGENT_INSTALL_USER`, `AGENT_INSTALL_PASSWORD`, `AGENT_INSTALL_DIR`가 정확해야 합니다.

8. 패치 흐름

T-Patch의 전체 흐름은 크게 세 단계입니다.

1. `pre_patch`: 패치 전 task 실행, 인스턴스 down 처리
2. `apply_patch`: 바이너리 교체, `TB_HOME` 전환

3. `post_patch` : 인스턴스 boot 처리, 패치 후 task 실행

non-rolling에서는 여러 노드에 대해 각 단계를 묶어서 진행합니다. rolling에서는 한 노드의 세 단계를 완료한 뒤 다음 노드로 넘어갑니다.

down 방향의 기본 순서는 `TAC -> TAS -> CM -> SSVR` 입니다. boot 방향의 기본 순서는 `SSVR -> CM -> TAS -> TAC` 입니다.

9. 로그와 결과 확인

기본 로그 파일명은 `tpatch.log` 입니다. `LOG_DIR` 을 지정하면 해당 디렉터리에 저장됩니다.

```
tail -f /app/logs/tpatch.log
grep ERROR /app/logs/tpatch.log
grep -E "ERROR|WARNING" /app/logs/tpatch.log
```

control 모드에서는 Control Plane 로그에 전체 흐름, 노드별 stage 결과, 사용자 입력 프롬프트가 모입니다. agent는 노드별 실행 로그를 Control Plane으로 relay합니다.

조건부 SHELL/SQL task의 수행 조건이 충족되지 않으면 실패가 아니라 `SKIPPED` 상태로 처리됩니다. control 모드에서는 `Execution condition not met` 사유가 함께 전달됩니다. 운영 결과를 확인할 때 `FAILED` 와 `SKIPPED` 를 구분합니다.

원격 bootstrap으로 agent를 기동한 경우 대상 노드의 `AGENT_INSTALL_DIR` 아래 로그도 확인할 수 있습니다.

```
cat /tmp/tpatch-agent/agent_bootstrap.log
cat /tmp/tpatch-agent/tpatch_agent.log
```

10. 자주 발생하는 문제

설정 파일을 읽지 못함

확인할 항목:

- `setting.yaml` 이 `tpatch` 실행 디렉터리에 있는지
- YAML 들여쓰기와 콜론이 올바른지
- `DBSVR` , `OPTION` 이 목록 형태인지
- 필수 키가 비어 있지 않은지

대표 필수 키:

- `ASIS_TB_HOME`
- `TAC_NAME`
- `TAC_USER`
- `TAC_PW`
- `TASK_JSON`
- `CHECK_TASKS`

바이너리 교체 검증 실패

`BINARY_REPLACE_AUTO: true` 를 사용한다면 다음을 확인합니다.

- `BINARY_FILE_PATH` 가 실제 `.tar.gz` 파일을 가리키는지
- `ASIS_TB_HOME` 과 `TOBE_TB_HOME` 이 모두 있는지
- 두 홈 경로가 서로 다른지
- `BINARY_REPLACE_EXTRA_TARGETS` 의 `source` , `target` 이 상대 경로인지

control 모드에서 agent 연결 실패

확인할 항목:

- Control Plane 서버에서 `AGENT_HOST:AGENT_PORT` 로 접속 가능한지

- 대상 노드 방화벽에서 `AGENT_PORT`가 열려 있는지
- agent를 미리 띄운 경우 `./tpatch --mode agent --port 8081`가 실행 중인지
- 원격 bootstrap을 사용할 경우 `AGENT_INSTALL_*` SSH 정보가 맞는지
- 같은 물리 agent 대상에 서로 다른 `AGENT_PORT`를 설정하지 않았는지

agent protocol 또는 binary mismatch

control 모드는 agent의 protocol version과 binary version을 확인합니다. protocol이 맞지 않으면 기존 agent 종료 후 현재 바이너리로 bootstrap을 시도합니다. 이미 떠 있는 agent를 닫지 못하면 덮어쓰지 않고 중단합니다.

task가 예상과 다르게 실행됨

확인할 항목:

- `tasks.json`의 `taskIdx`와 `TASK_VARIABLES` / `CONDITIONAL_TASK_EXECUTION`의 키가 일치하는지
- 조건부 task인지
- `CHECK_TASKS: true`로 인해 사용자 입력에서 취소되지 않았는지
- rolling 전용 task가 아닌지
- `order`, `order_add`가 의도한 단계인지

USER_SWITCH 또는 root 권한 오류

확인할 항목:

- T-Patch 또는 agent가 실제로 root인지
- 자동 agent 설치 시 `AGENT_INSTALL_USER: root`인지
- `USER_SWITCH` 대상 OS 계정이 존재하는지
- root에서 `su - <OS_USER> -c 'id'`가 성공하는지
- 전환 계정이 TAC/TAS의 TB_HOME과 Tibero 명령에 접근할 수 있는지

root 사용 또는 user switching이 차단된 환경은 현재 지원하지 않습니다. `sudo` 등으로 임의 변경하여 진행하지 말고 패치를 중단합니다.

Wallet/HSM 사전 검증 실패

확인할 항목:

- 비대화형 모드에서 `WALLET_PASSWORD`가 설정되어 있는지
- `USE_HSM: true`이면 `HSM_PIN`이 설정되어 있는지
- `$TB_HOME/config/$TB_SID.tip`을 읽을 수 있는지
- TIP의 `HSM_LIB_FILE`이 지원 목록과 일치하는지

SHELL task 노드 블록 오류

`tpatch_task_execute_node`에는 `ALL`, `FIRST`, `LAST`만 사용할 수 있습니다. 블록 중첩, 종료 표식 누락, 시작 표식 없는 종료 표식은 오류로 처리됩니다.

DBSVR/SSVR TB_HOME 중복 오류

DBSVR와 SSVR가 같은 물리 호스트에 있으면 `ASIS_TB_HOME`과 `TOBE_TB_HOME`을 서로 다르게 설정해야 합니다.

11. 운영 권장 사항

운영 전에는 항상 `setting.yaml`과 `tasks.json`을 같은 기준으로 검토합니다. 특히 `BINARY_FILE_PATH`, `ASIS_TB_HOME`, `TOBE_TB_HOME`, `TASK_JSON`은 실제 패치 당일 경로와 일치해야 합니다.

자동화 수준을 높이고 싶으면 `CHECK_TASKS: false`를 사용하되, 수동 판단이 필요한 작업은 `MANUAL` task 또는 조건부 task로 분리합니다.

control 모드를 반복적으로 사용할 서버라면 agent 포트를 노드별로 고정하고, 방화벽 정책과 SSH 계정을 미리 검증합니다.

12. 빠른 참조

```
# 버전 확인
./tpatch --version

# standalone 패치
./tpatch --patch

# standalone rolling 패치
./tpatch --patch --rolling

# standalone 롤백
./tpatch --rollback

# control 패치
./tpatch --patch --mode control

# control rolling 패치
./tpatch --patch --mode control --rolling

# agent 실행
./tpatch --mode agent --port 8081

# 로그 확인
tail -f /app/logs/tpatch.log
```