M06-03 · GIBALJA VISUAL MANUAL

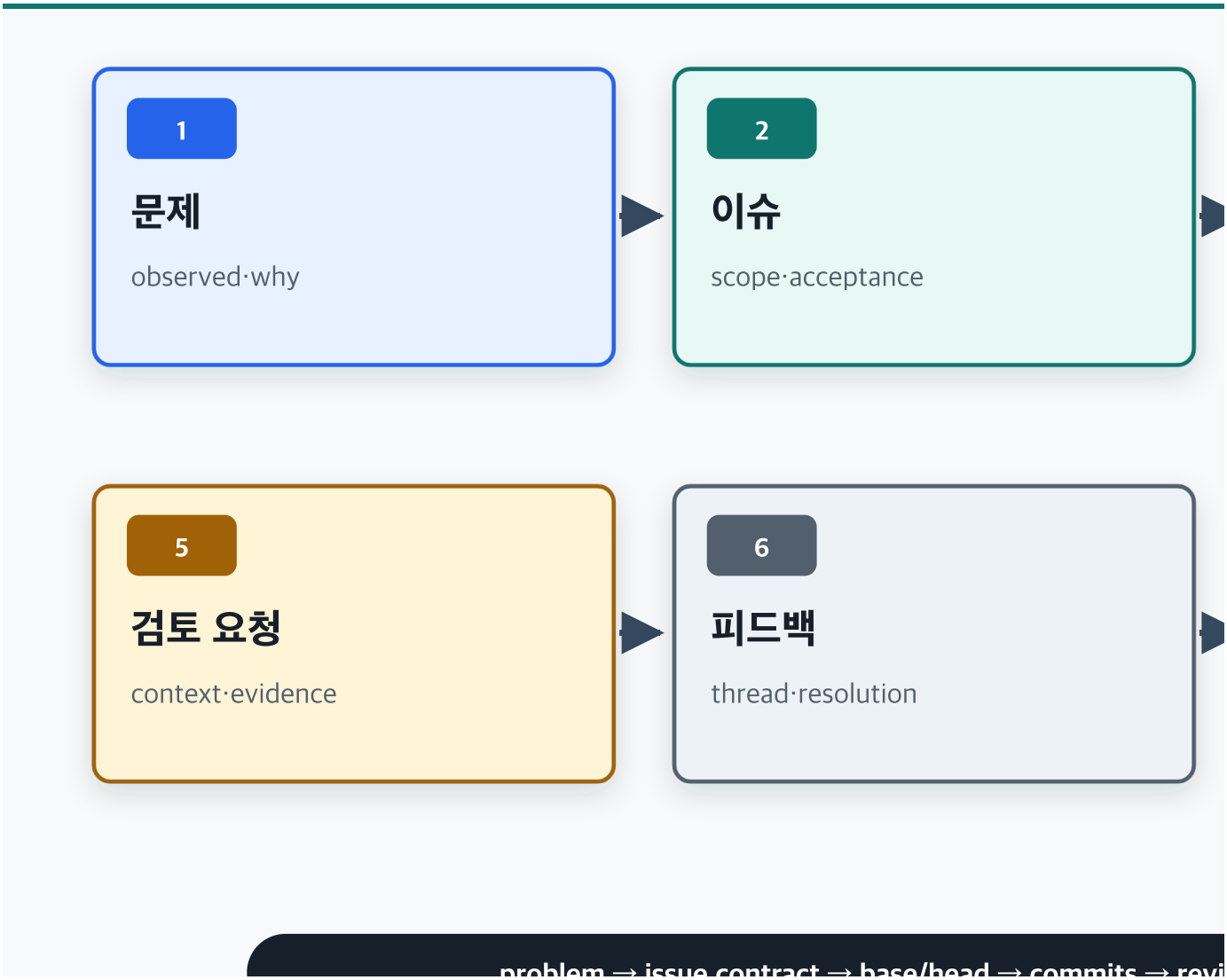
이슈·브랜치·검토 요청으로 협업하기

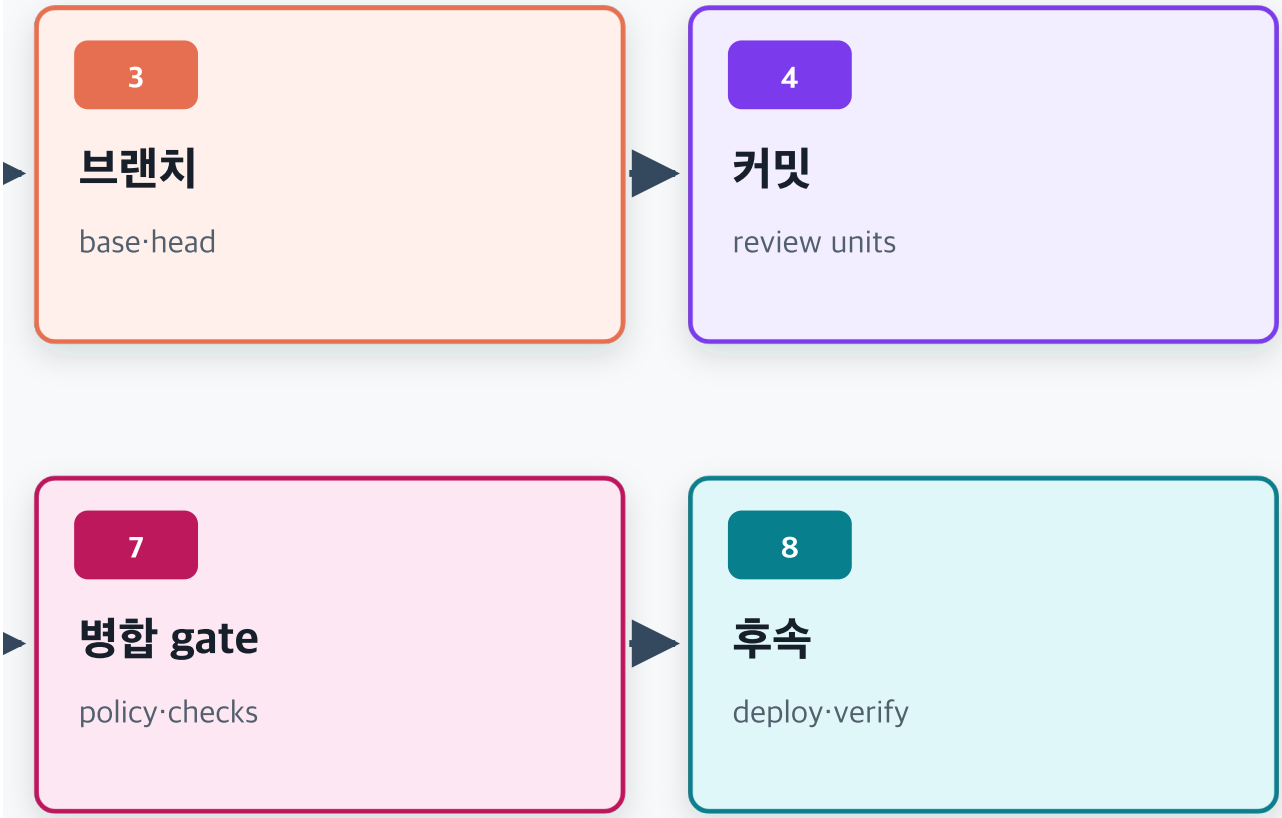
> **한 문장 목표:** `문제 증거 → 이슈 계약 → base·head → 검토 가능한 commit → 검토 요청 → feedback 해결 → merge gate → merge 결과 → deploy 증거` 를 한 흐름으로 연결하고, "누가 봐도 지금 무엇을 결정할 수 있는가"를 재현 가능한 기록으로 남깁니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	60분	15분	협업 작업 기록, 이슈 정의서, 브랜치·커밋 계획, 검토 요청서, 피드백 해결 기록

좋은 협업은 댓글 수가 많은 상태가 아닙니다. 관찰한 문제와 완료 기준이 한 이슈에 있고, 변경 범위가 base와 head로 고정되며, 피드백이 해결 commit과 test에 연결되고, 현재 head가 repository policy를 통과했는지 판정할 수 있는 상태입니다. **"고쳤습니다"보다 "어느 OID에서 어떤 기준을 어떤 증거로 충족했습니다"가 강합니다.**

그림 1. 협업의 산출물은 서로 떨어진 문서가 아닙니다. 앞 단계의 decision과 evidence가 다음 단계의 입력이 됩니다.


1
문제
observed·why
2
이슈
scope·acceptance
5
검토 요청
context·evidence
6
피드백
thread·resolution
problem → issue contract → base/head → commits → revi



ew request → feedback → merge gate → deploy evidence

0. 이 PDF를 공부하는 방법

1회차 · 그림만 읽기 · 20분

그림 1부터 그림 14까지 제목과 아래 결론 띠만 읽습니다. 다음 문장을 소리 내어 말합니다.

```
issue는 solution memo가 아니라 problem contract다.
한 review request는 한 decision을 요청한다.
base는 받을 쪽이고 head는 제안할 쪽이다.
branch 이름은 움직이므로 OID와 관찰 시각을 남긴다.
merge base는 두 history가 갈라진 공통 조상이다.
commit은 index snapshot과 parent와 message다.
test pass는 작성된 test만 통과했다는 뜻이다.
feedback은 resolution commit과 current-head verification으로 닫는다.
approval과 required check와 thread와 freshness는 서로 다른 gate다.
merged와 deployed와 verified는 서로 다른 상태다.
```

2회차 · 실습 저장소 만들기 · 10분

협업 실습 생성기를 실행합니다. 기존 target folder가 있으면 exit code 2로 멈추며 덮어쓰지 않습니다.

```
./02_Labs/G06_Git/L06-03_create-collaboration-practice.sh
```

생성기는 외부 package나 internet 없이 local bare remote, author clone, reviewer clone, evidence folder를 만듭니다.

3회차 · 12개 검토 장면 판독 · 40분

협업 검토 데스크를 엽니다. 해설을 보기 전에 author, reviewer, maintainer 역할을 바꾸며 다섯 값을 말합니다.

```
현재 decision =
base·head range =
확인한 evidence =
남은 gap =
다음 owner·action =
```

4회차 · 내 repository 기록 · 65분

협업 작업 기록 양식을 채웁니다. 낯선 말은 이슈·브랜치·검토 협업 용어집에서 확인합니다.

1. 먼저 협업의 기준점을 고정합니다

1.1. repository와 revision

M06-01과 M06-02의 기준을 이어받습니다.

```
git rev-parse --show-toplevel
git remote -v
git rev-parse --verify HEAD
git status --short --branch
git --version
```

항목	기록값	이 값이 막는 혼동
repository root	absolute path	다른 repository에서 실행
remote identity	fetch·push URL	fork와 upstream 혼동
current full OID	40-hex object name	움직이는 branch만 기록
working tree	clean 또는 변경 목록	local 미반영 변경 누락
Git version	`git version 2.54.0`	command behavior 차이
capture time	ISO 8601 + timezone	오래된 관찰을 현재 상태로 오해

실습은 local Git 2.54.0에서 검증했습니다. 공식 Git 문서는 2026-07-15 기준 2.55.0 문서를 확인했습니다. version이 다르면 output 형식과 option 동작을 다시 확인합니다.

1.2. 읽기와 쓰기의 경계

처음에는 관찰 command로 시작합니다.

```
git status --short --branch
git branch --all --verbose --verbose
git log --graph --oneline --decorate --all
git remote show origin
```

다음 command는 repository·remote state를 바꿀 수 있습니다.

```
git switch -c
git commit
git push
git merge
git rebase
git branch -d
```

특히 강제 push는 다른 사람이 기반으로 삼은 history를 바꿀 수 있습니다. `--force-with-lease`도 remote-tracking ref가 오래되었거나 자동 fetch가 개입한 환경에서는 기대한 보호가 되지 않을 수 있으므로 repository policy와 team 절차를 먼저 확인합니다.

실행 전 멈춤: production repository, shared branch, protected branch, signed commit, regulated evidence가 관련되면 이 PDF의 예시 command를 그대로 실행하지 않습니다. 먼저 권한·정책·rollback·audit 보존 조건을 확인합니다.

2. 협업을 evidence chain으로 봅니다

한 변경은 다음 질문을 통과합니다.

단계	decision	최소 evidence
problem	실제 mismatch인가	관찰값·재현 조건·영향
issue	무엇을 완료할 것인가	expected outcome·scope·acceptance
branch	어느 history에서 갈라졌는가	base·head·merge base OID
commit	어떤 검토 순서인가	snapshot·parent·message·test
review request	무엇을 결정해 달라는가	diff·risk·check·open question
feedback	어떤 gap을 어떻게 닫았는가	thread·resolution commit·verification
merge	지금 policy를 통과했는가	approvals·checks·threads·freshness
post-merge	사용자가 실제로 받았는가	merge result·artifact·deploy·verification

CORE MODEL 협업 기록의 목적은 활동을 자랑하는 것이 아니라 다음 사람이 같은 decision을 다시 내릴 수 있게 하는 것입니다.

decision + evidence + gap + owner + next action

3. issue를 problem contract로 씁니다

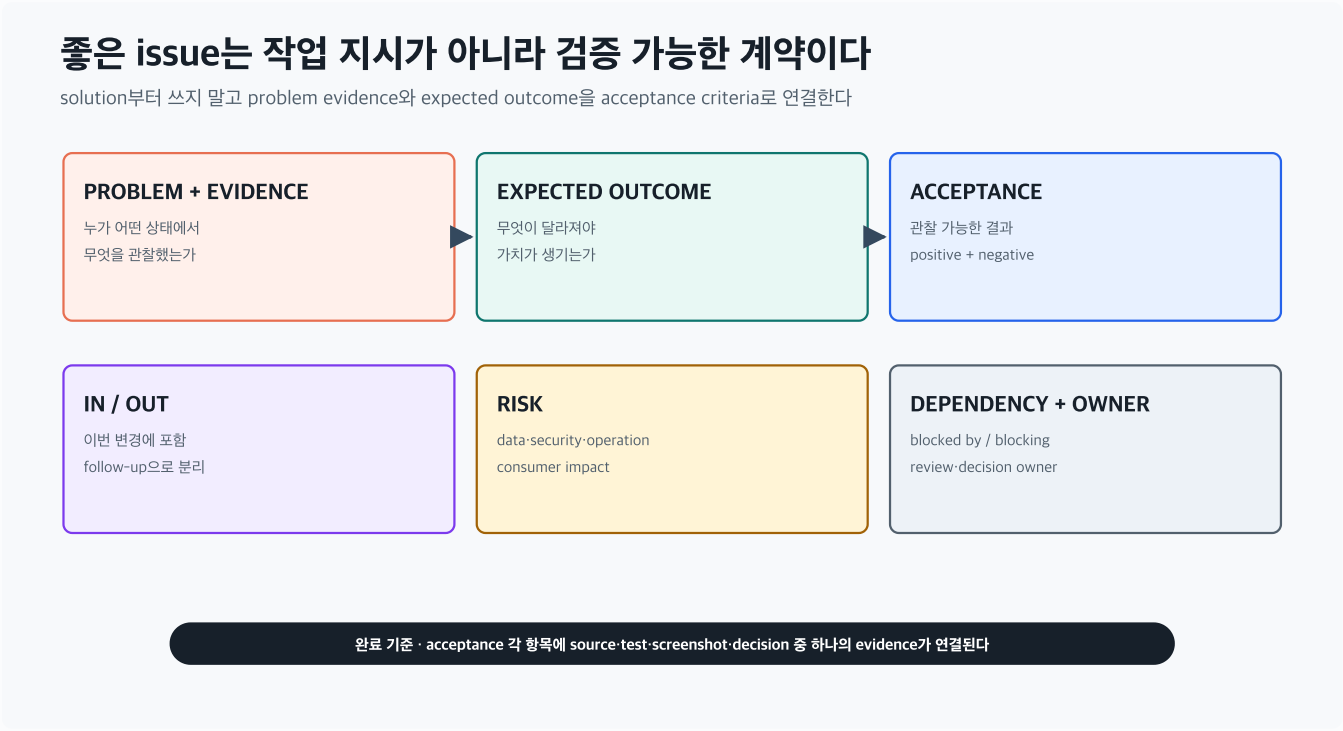
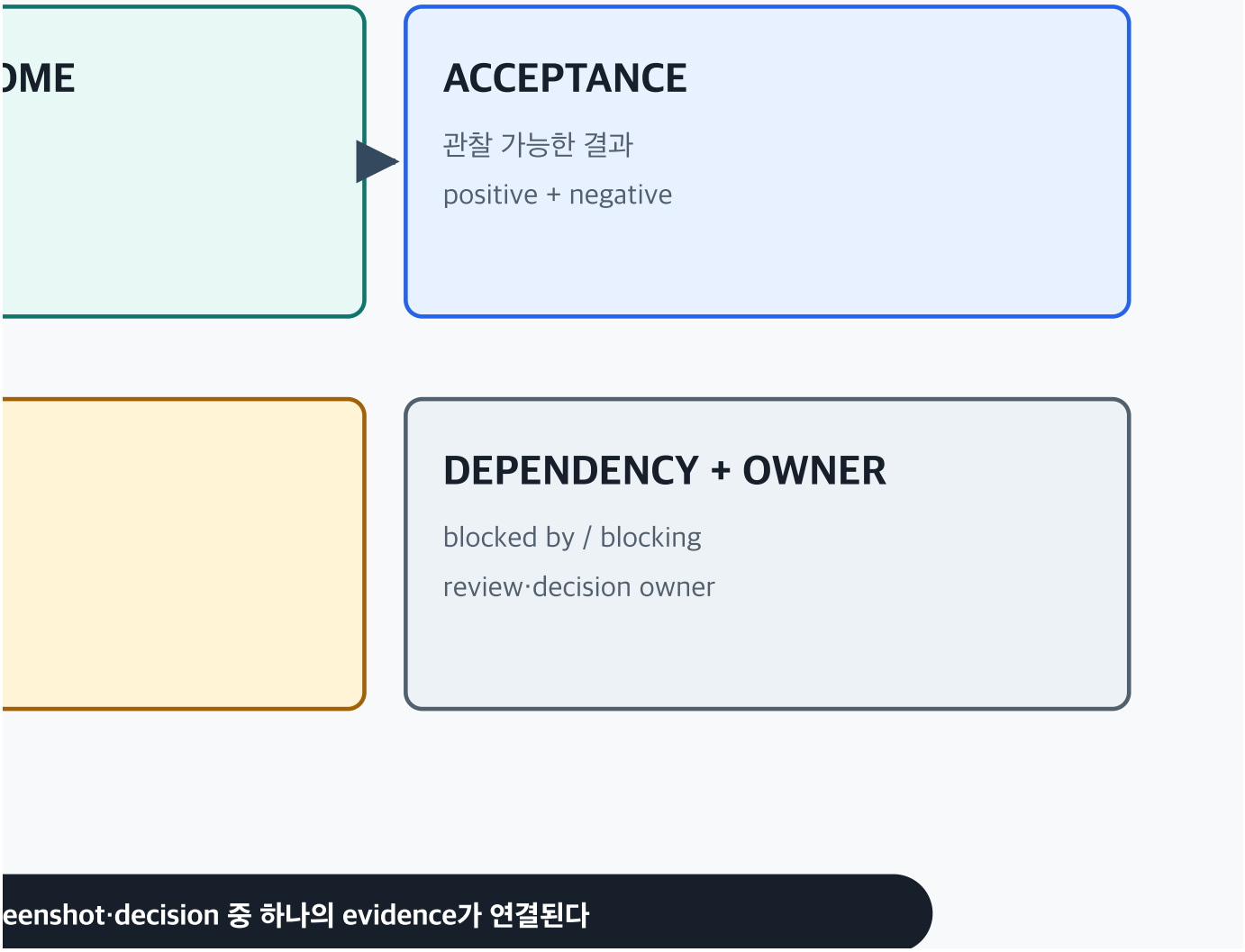


그림 2. 좋은 issue는 해결책 이름보다 관찰된 mismatch와 검증 가능한 완료 상태를 먼저 고정합니다.


PROBLEM + EVIDENCE
누가 어떤 상태에서
무엇을 관찰했는가
EXPECTED OUTCO
무엇이 달라져야
가치가 생기는가
IN / OUT
이번 변경에 포함
follow-up으로 분리
RISK
data·security·operation
consumer impact
완료 기준 · acceptance 각 항목에 source·test·scr


ACCEPTANCE
관찰 가능한 결과
positive + negative
DEPENDENCY + OWNER
blocked by / blocking
review·decision owner
eenshot·decision 중 하나의 evidence가 연결된다

3.1. 제목은 outcome을 가리킵니다

나쁜 제목:

```
badge 수정
테스트 보강
코드 정리
```

좋은 제목:

```
수료 조건 결과에 completion badge contract를 추가한다
```

제목만으로 모든 세부사항을 담을 필요는 없지만, reviewer가 “무엇이 달라져야 하는가”를 짐작할 수 있어야 합니다.

3.2. problem과 evidence

실습의 관찰:

```
화면 adapter가 badge visible과 label을 다시 추론한다.
미수료 learner에서 hidden badge의 label이 남으면
accessibility tree와 analytics가 상태를 잘못 읽을 수 있다.
```

다음 요소를 분리합니다.

요소	예
actor	learner progress를 표시하는 화면 adapter
current behavior	각 화면이 visible·label을 재추론
expected behavior	policy가 하나의 badge contract 반환
impact	접근성·분석 결과 불일치
reproduction	progress 100%, score 74
observed evidence	output·screenshot·log·test gap

3.3. acceptance criteria는 관찰 가능해야 합니다

```
[ ] progress와 score가 모두 threshold 이상이면
    badge.visible = true, badge.label = "수료 가능"

[ ] 하나라도 미달이면
    badge.visible = false, badge.label = null

[ ] 기존 outcome과 invalid-input validation은 유지된다.
[ ] runtime dependency는 0개를 유지한다.
```

"잘 동작한다", "깔끔하다", "성능이 좋다"는 그대로는 판정할 수 없습니다. 입력, 상태, 관찰할 출력, 허용 범위를 적습니다.

3.4. in scope와 out of scope

In scope	Out of scope
badge output shape	UI·CSS
visible·hidden decision	localization
positive·negative regression test	analytics schema
기존 validation 보존	deployment

out of scope는 "안 한다"는 변명이 아닙니다. 이번 decision의 경계를 만들고, 필요한 후속 작업을 잃지 않게 합니다.

3.5. dependency와 owner

GitHub의 issue dependency와 sub-issue, GitLab의 linked issue·dependency 기능처럼 platform은 관계 표현을 제공할 수 있습니다. 다만 실제 지원 범위는 plan·version·repository configuration에 따라 달라집니다. 최소한 다음을 text로도 남깁니다.

```
blocked by =
blocks =
related =
follow-up =
decision owner =
review owner =
release owner =
```

30초 확인 1

“영어 label도 넣어 주세요”라는 제안은 이번 issue와 같은 outcome·owner·release·rollback을 공유합니까? 아니면 follow-up issue로 분리해야 합니까?

4. 한 review request는 한 decision을 요청합니다

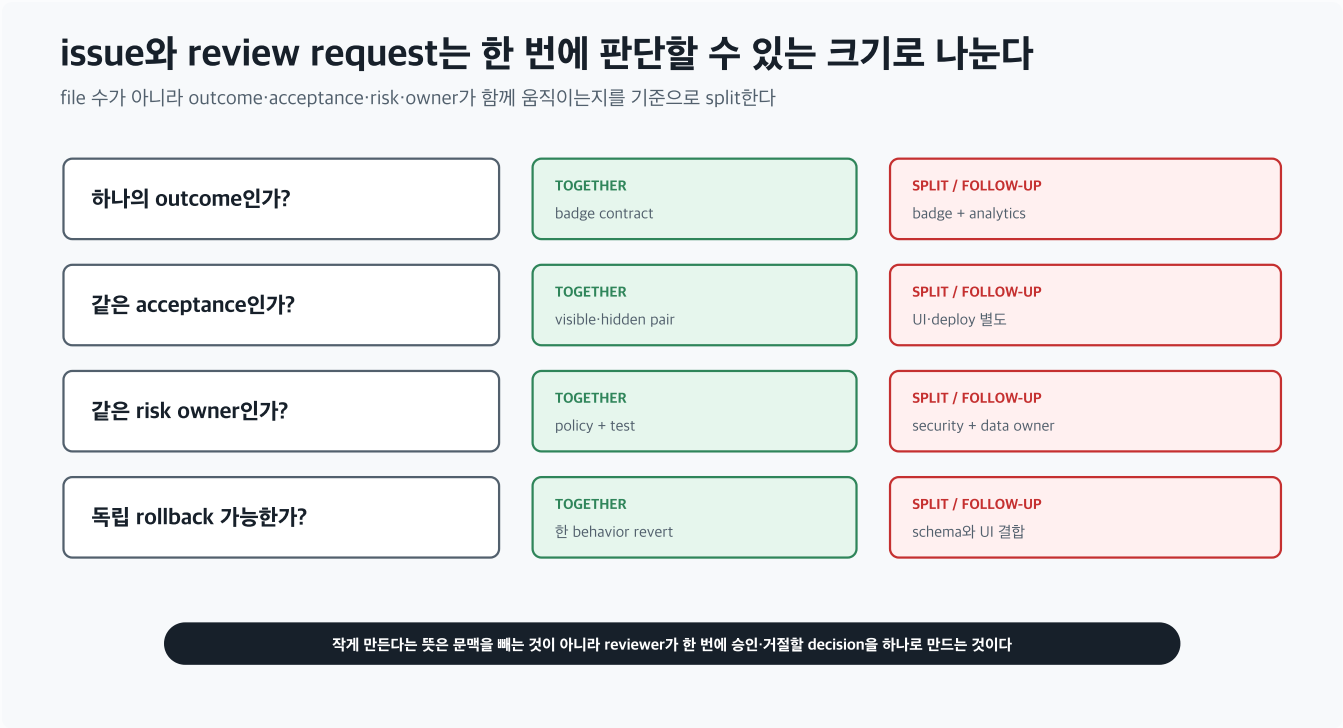


그림 3. 같은 파일을 바꾼다고 같은 decision은 아닙니다. outcome·owner·risk·rollback이 다르면 분리를 검토합니다.

하나의 outcome인가?	**TOGETHER** badge contract
같은 acceptance인가?	**TOGETHER** visible·hidden pai
같은 risk owner인가?	**TOGETHER** policy + test
독립 rollback 가능한가?	**TOGETHER** 한 behavior revert

작게 만든다는 뜻은 문맥을 빼는 것이 아니라 reviewer가

	SPLIT / FOLLOW-UP badge + analytics
	SPLIT / FOLLOW-UP UI·deploy 별도
	SPLIT / FOLLOW-UP security + data owner
	SPLIT / FOLLOW-UP schema와 UI 결합

가 한 번에 승인·거절할 decision을 하나로 만드는 것이다

4.1. 범위를 묶는 네 기준

다음 네 질문에 대부분 같은 답이 나와야 한 review request에 묶기 쉽습니다.

```
같은 user outcome인가?
같은 acceptance로 검증하는가?
같은 risk owner가 승인하는가?
같은 rollback 단위인가?
```

4.2. 큰 변경을 줄이는 방법

분리 축	예
preparation / behavior	함수 이동 후 정책 변경
contract / consumer	output shape 후 UI adapter
data migration / read path	schema 추가 후 read 전환
feature / localization	badge contract 후 번역
source / rollout	코드 merge 후 feature flag 활성화

분리는 무조건 commit 수를 늘리는 일이 아닙니다. reviewer가 한 번에 내릴 decision을 작게 만드는 일입니다.

4.3. 분리하면 안 되는 경우

다음은 함께 있어야 의미가 분명할 수 있습니다.

- source change와 그 behavior를 증명하는 targeted test
- schema change와 호환되는 최소 read·write path
- security check를 통과하기 위한 정책과 enforcement
- rollback이 불가능한 중간 state를 만드는 반쪽 변경

> 파일 수나 line 수만으로 scope를 기계적으로 나누지 않습니다. 작은 한 줄도 authorization contract를 바꿀 수 있고, generated snapshot 수백 줄은 한 decision일 수 있습니다.

5. base·head·merge base를 방향 있게 읽습니다

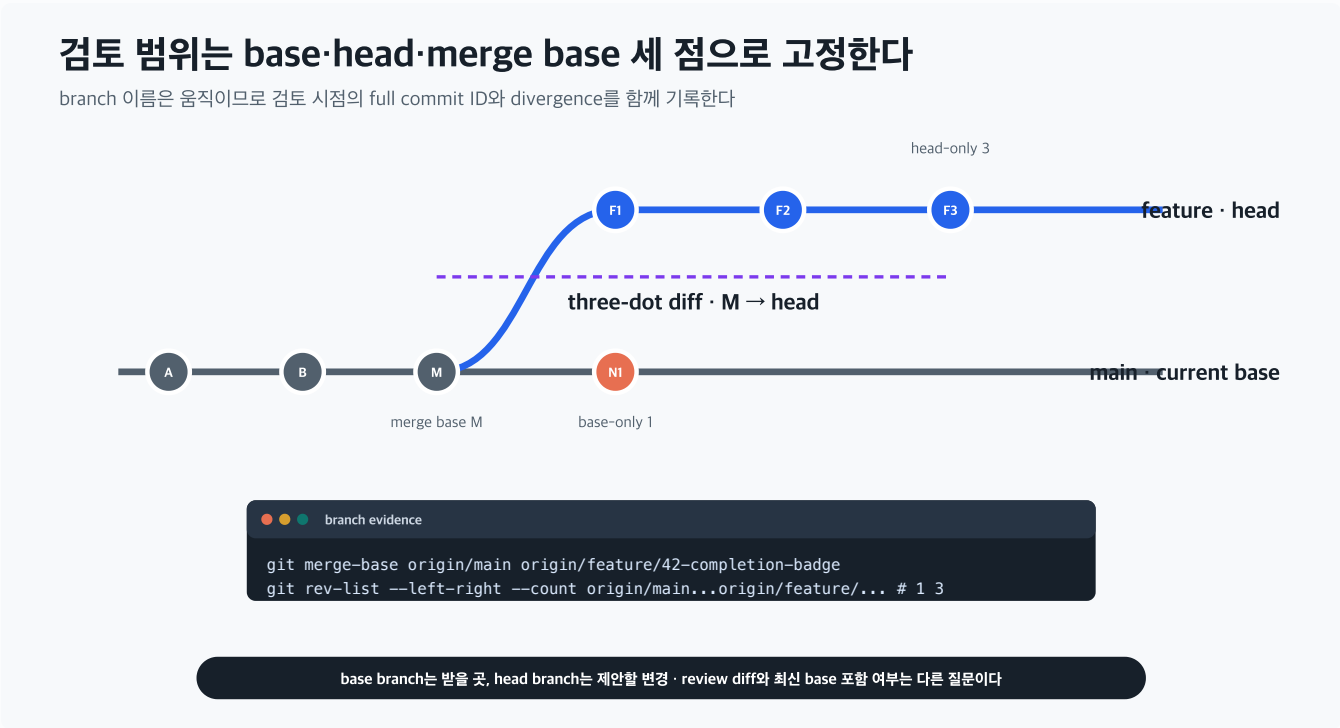
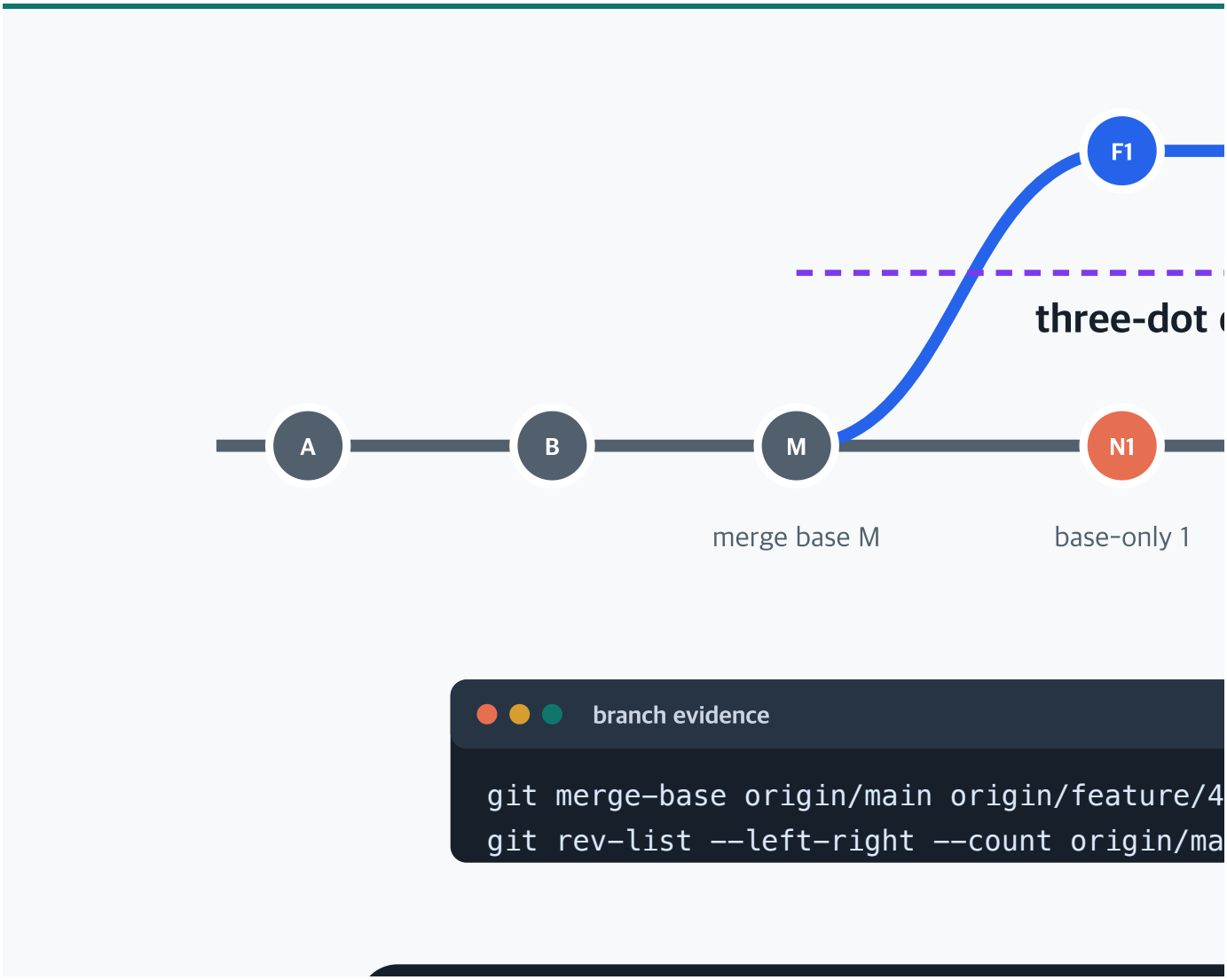
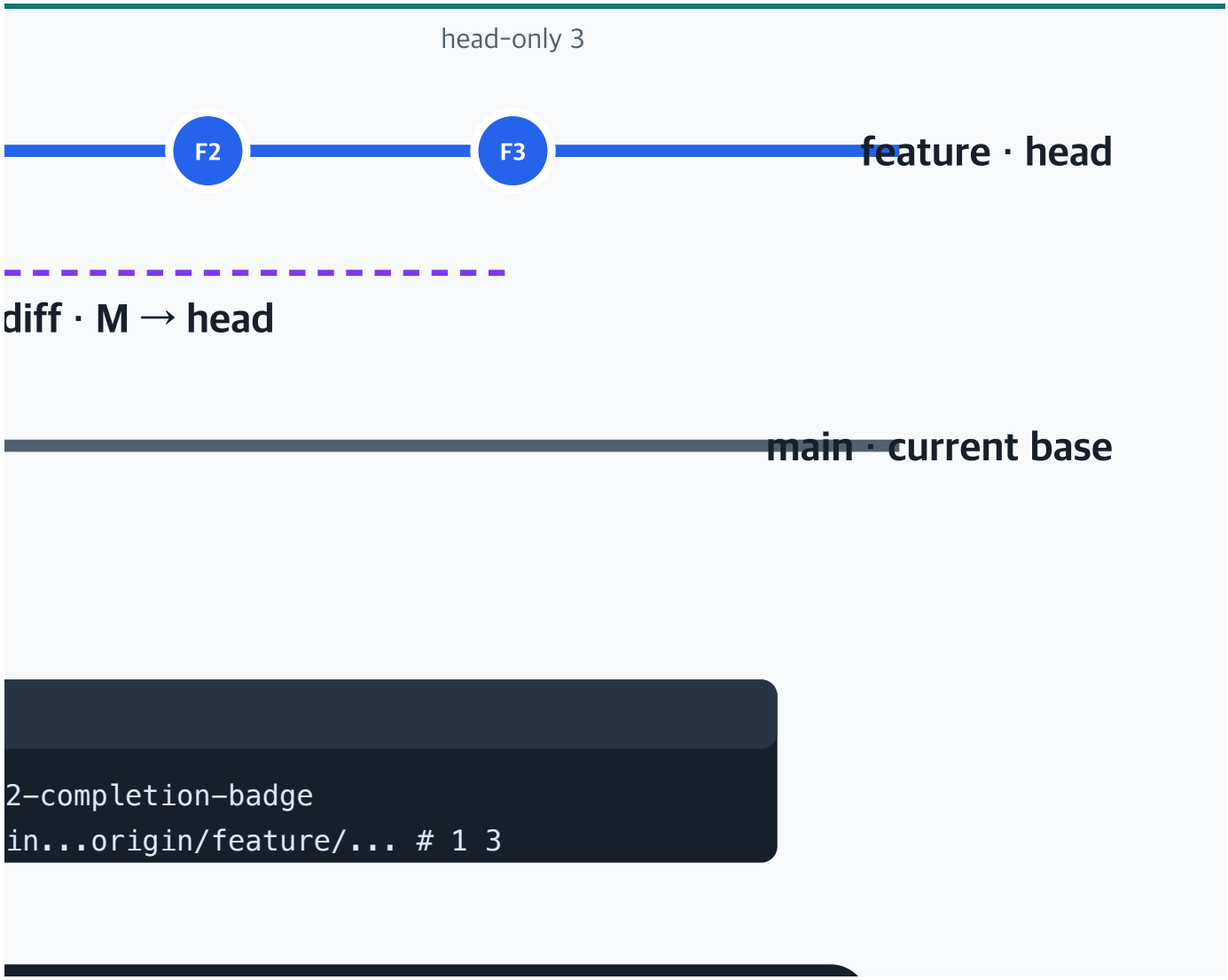


그림 4. base branch와 head branch는 역할이 다릅니다. merge base는 두 history가 갈라진 공통 조상입니다.


F1
three-dot
A
B
M
N1
merge base M
base-only 1
branch evidence
git merge-base origin/main origin/feature/4
git rev-list --left-right --count origin/ma




```
2-completion-badge
in...origin/feature/... # 1 3
```

5.1. 세 기준

이름	질문	실습
base	변경을 받을 target은	`origin/main`
head	제안하는 source는	`origin/feature/42-completion-badge`
merge base	두 history의 공통 기준은	`90640e5...`

```
git rev-parse origin/main
git rev-parse origin/feature/42-completion-badge
git merge-base origin/main origin/feature/42-completion-badge
```

실습 결과:

```
base current  3619f57  docs: publish next release window
head current  32700a6  fix: hide label when completion badge is hidden
merge base    90640e5  docs: record issue 42 context
```

5.2. branch 이름은 ref입니다

branch는 commit 자체가 아니라 commit을 가리키는 ref입니다. 새 commit이 생기면 ref가 움직입니다. 그래서 review request에는 이름과 함께 다음을 기록합니다.

```
base repository·branch·full OID =
head repository·branch·full OID =
merge base full OID =
observed at =
```

5.3. three-dot 표기의 두 얼굴

```
git diff origin/main...origin/feature/42-completion-badge
git log origin/main...origin/feature/42-completion-badge
```

두 command 모두 `...`를 쓰지만 같은 질문이 아닙니다.

command	질문
`git diff A...B`	merge base(A,B)의 tree와 B의 tree 차이
`git log A...B`	A 또는 B 중 한쪽에만 있는 commit의 symmetric difference

기호 모양만 외우지 말고 command의 공식 정의와 output을 함께 읽습니다.

6. branch lifecycle과 freshness를 관리합니다

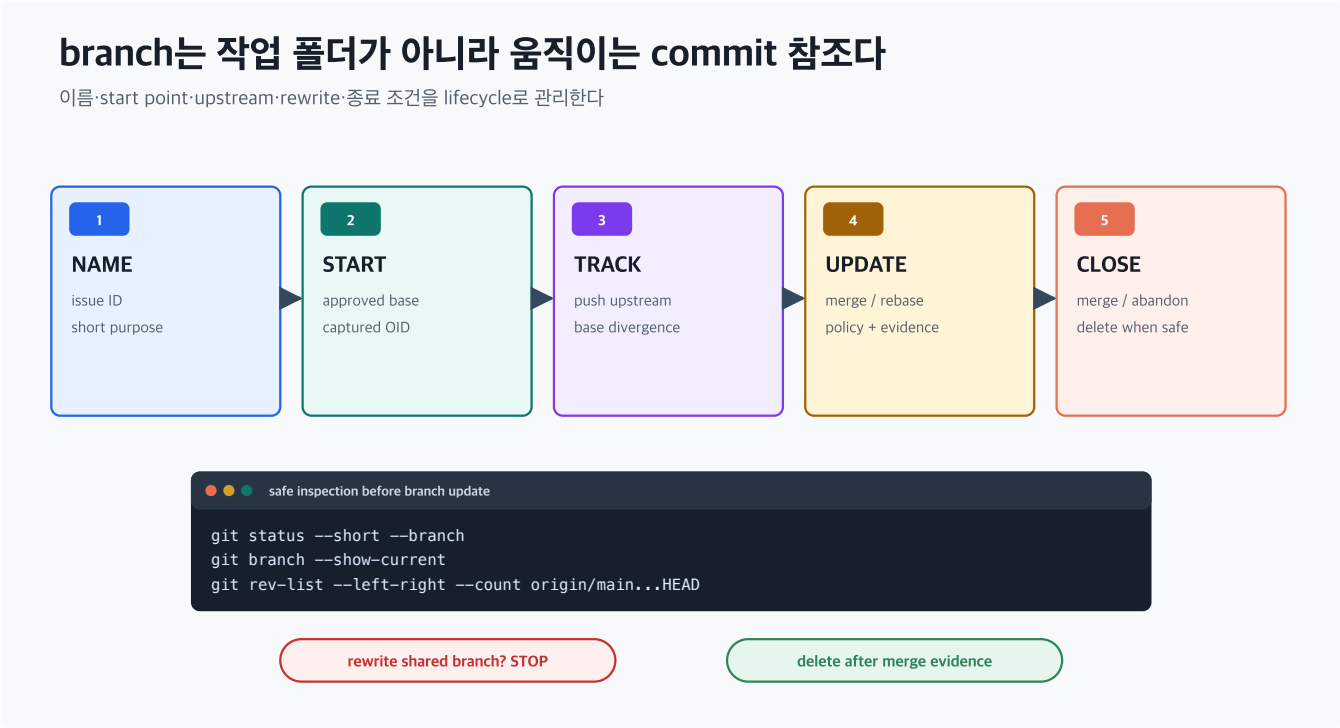
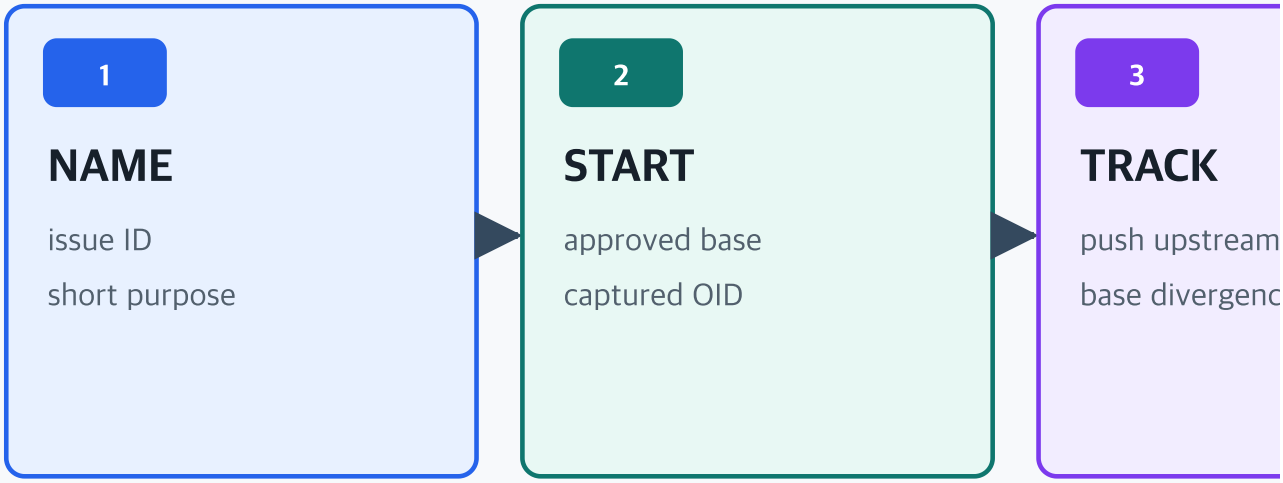


그림 5. branch는 만드는 순간보다 기준·owner·upstream·freshness를 계속 기록하는 것이 중요합니다.



safe inspection before branch update

```
git status --short --branch
git branch --show-current
git rev-list --left-right --count origin/main...H
```

rewrite shared branch? STOP


4
UPDATE
merge / rebase
policy + evidence
5
CLOSE
merge / abandon
delete when safe
EAD
delete after merge evidence

6.1. branch를 만들기 전

```
git status --short --branch
git fetch --prune origin
git switch main
git branch --show-current
git rev-parse origin/main
```

`fetch`는 remote-tracking ref를 갱신하지만 working branch를 자동으로 merge하지 않습니다. fetch 시각과 remote를 기록합니다.

6.2. branch 이름

repository convention이 있으면 그대로 따릅니다.

```
feature/42-completion-badge
fix/417-null-label
docs/88-review-guide
```

이름은 검색 단서일 뿐 issue 관계를 자동 보장하지 않습니다. issue link를 별도로 남깁니다.

6.3. divergence는 두 숫자입니다

```
git rev-list --left-right --count \
  origin/main...origin/feature/42-completion-badge
```

실습 결과:

```
1	3
```

값	뜻
left 1	current base에만 있는 commit 1
right 3	head에만 있는 commit 3

head-only가 많다는 사실은 최신이라는 뜻이 아닙니다. base-only가 1이므로 current base가 head에 포함되지 않았습니다. up-to-date가 required gate라면 blocking입니다.

6.4. update 전략

merge, rebase, branch recreation은 history와 OID를 다르게 만듭니다. 팀 정책을 확인합니다.

```
allowed update method =
force push allowed =
approval dismissal on new push =
required checks rerun =
signed commits required =
review thread behavior =
```

30초 확인 2

`head-only 3`만 보고 최신 branch라고 말할 수 있습니까? `left 1`은 어떤 질문을 새로 만듭니까?

7. commit을 review unit으로 만듭니다

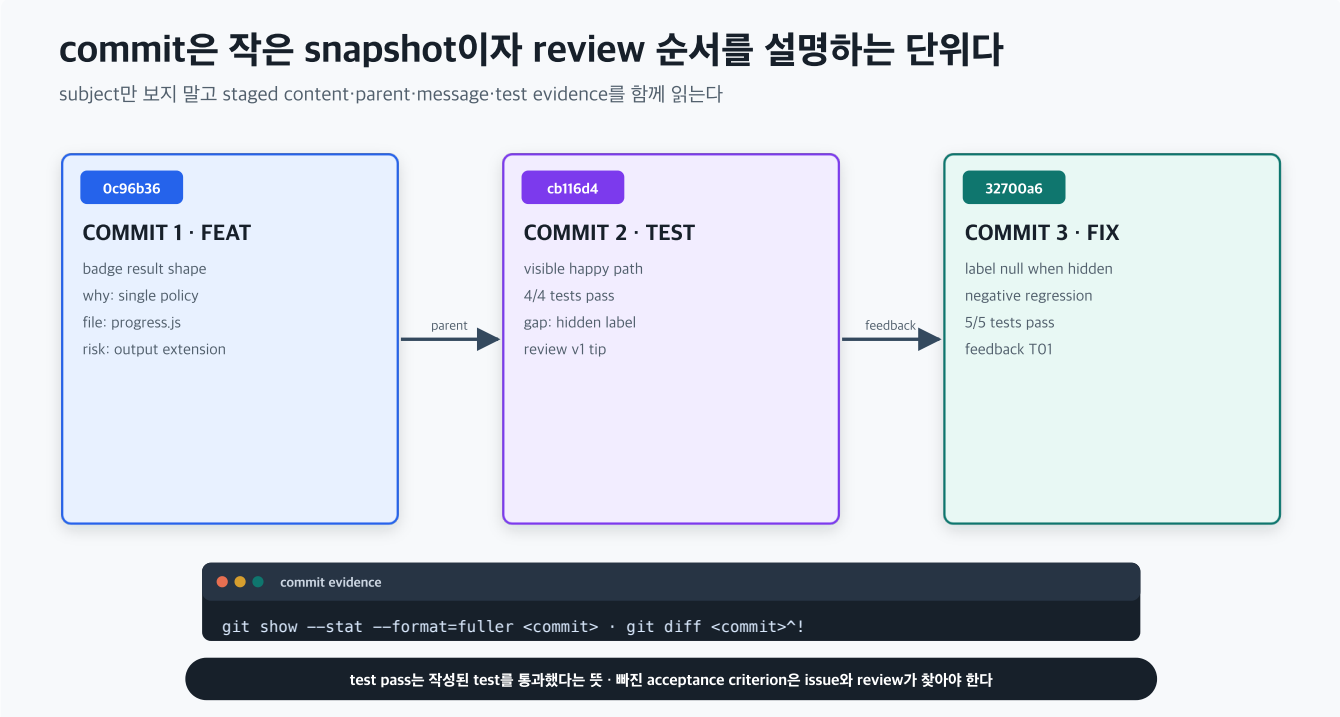
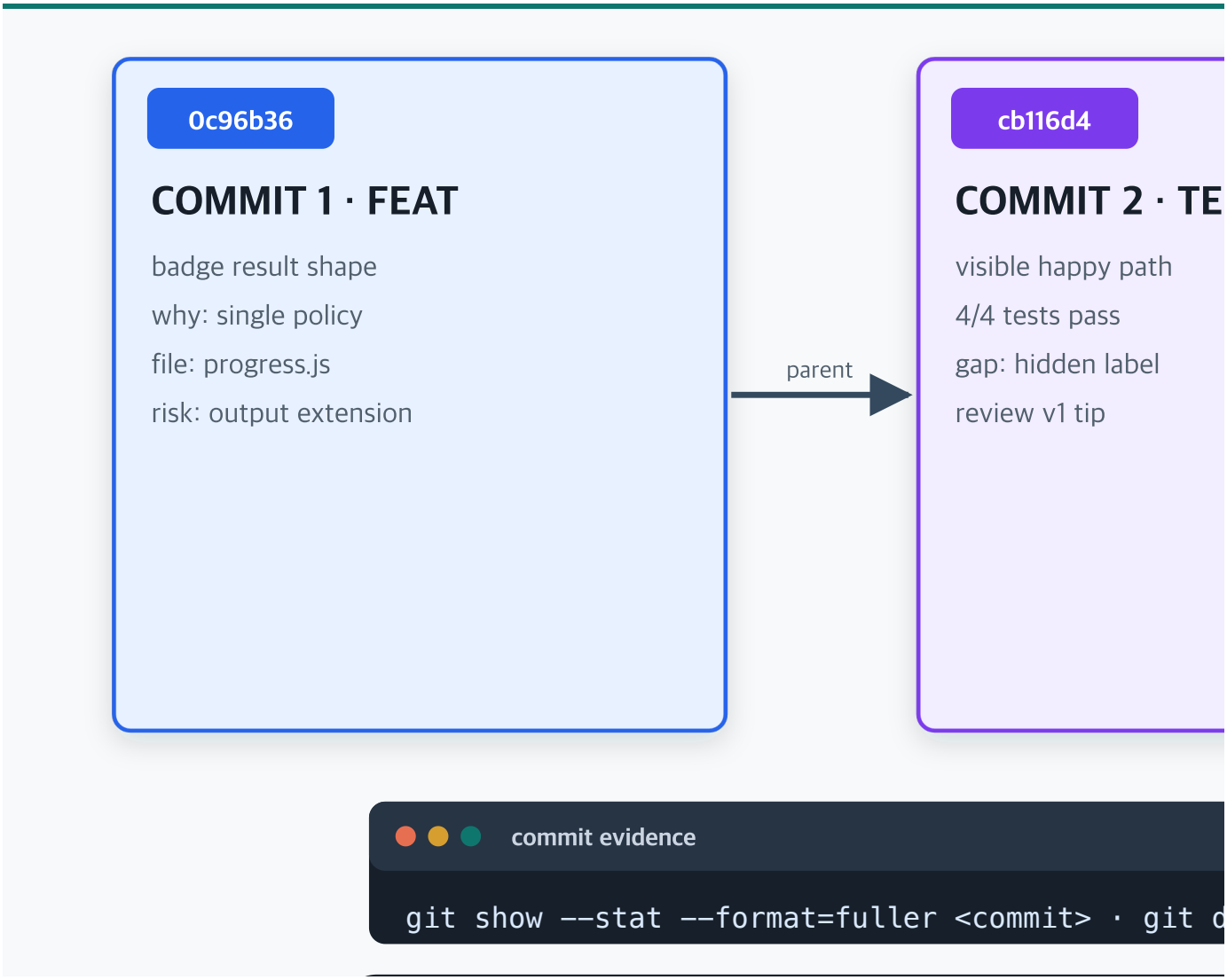
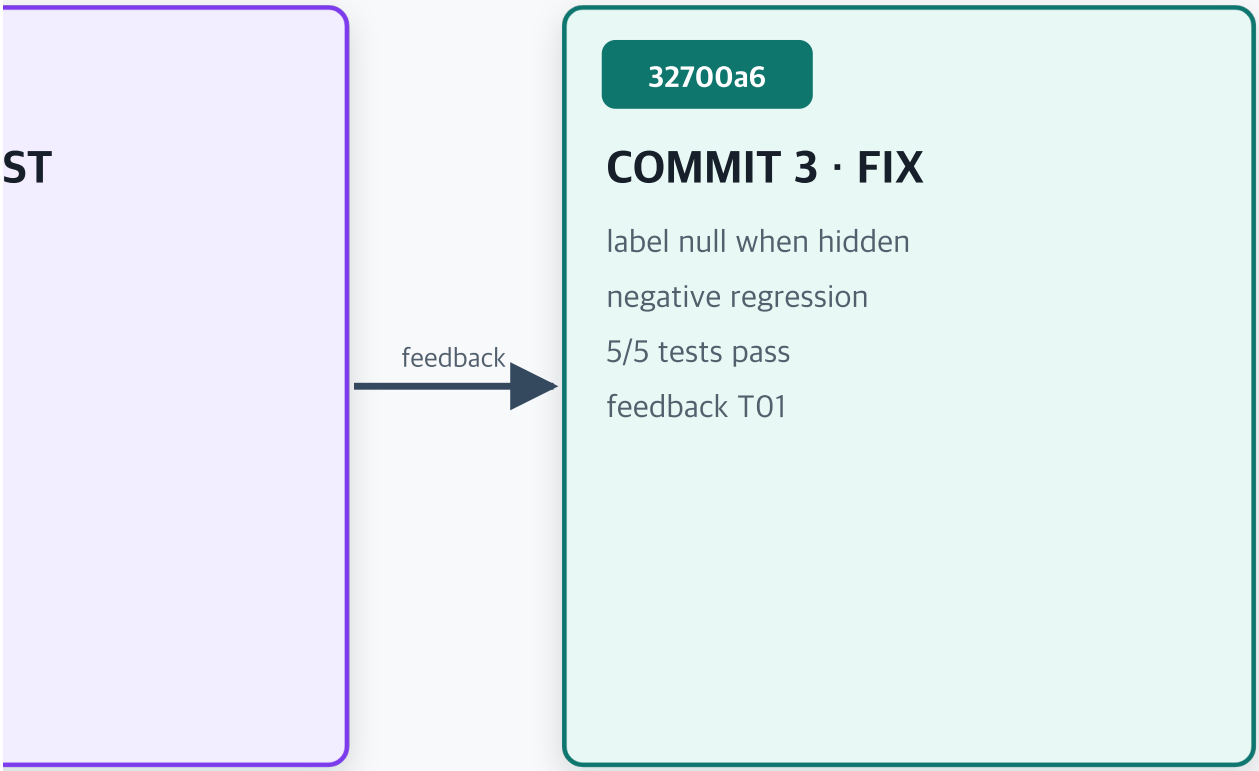


그림 6. commit은 저장 버튼이 아니라 staged index의 snapshot, parent, message와 metadata를 가진 object입니다.


0c96b36
COMMIT 1 · FEAT
badge result shape
why: single policy
file: progress.js
risk: output extension
parent
cb116d4
COMMIT 2 · TE
visible happy path
4/4 tests pass
gap: hidden label
review v1 tip
commit evidence
git show --stat --format=fuller <commit> · git d


ST
feedback
32700a6
COMMIT 3 · FIX
label null when hidden
negative regression
5/5 tests pass
feedback T01

```
iff <commit>^!
```

7.1. 무엇이 commit되는가

```
git status --short
git diff
git diff --staged
git commit
```

관찰	질문
working tree diff	수정했지만 아직 stage하지 않은 것은
staged diff	다음 commit snapshot에 들어갈 것은
untracked	아직 Git이 추적하지 않는 것은
ignored	policy로 제외된 것은

commit은 working folder 전체를 막연히 저장하는 행위가 아닙니다. index에 준비된 snapshot을 parent와 message에 연결합니다.

7.2. 실습의 commit sequence

```
0c96b36 feat: add completion badge result
cb116d4 test: cover visible completion badge
32700a6 fix: hide label when completion badge is hidden
```

이 sequence는 검토 이야기를 만듭니다.

```
contract 추가
→ happy path test
→ review가 negative gap 발견
→ source fix + regression test
```

7.3. 제목보다 content

```
git show --stat 32700a6
git show 32700a6^!
git show --format=fuller --no-patch 32700a6
```

제목만 보고 변경을 단정하지 않습니다. tree diff, parent, author·committer, timestamp, message body, trailer를 함께 봅니다.

7.4. message는 why를 남깁니다

```
fix: hide label when completion badge is hidden

Return null for the hidden state so accessibility and analytics
do not interpret a stale completion label.

Refs: #42
Review-Thread: T01
```

trailer는 repository automation과 정책이 실제로 해석하는 key인지 확인합니다. 보기 좋은 text가 자동 연결을 보장하지 않습니다.

8. 검토 요청은 decision context입니다

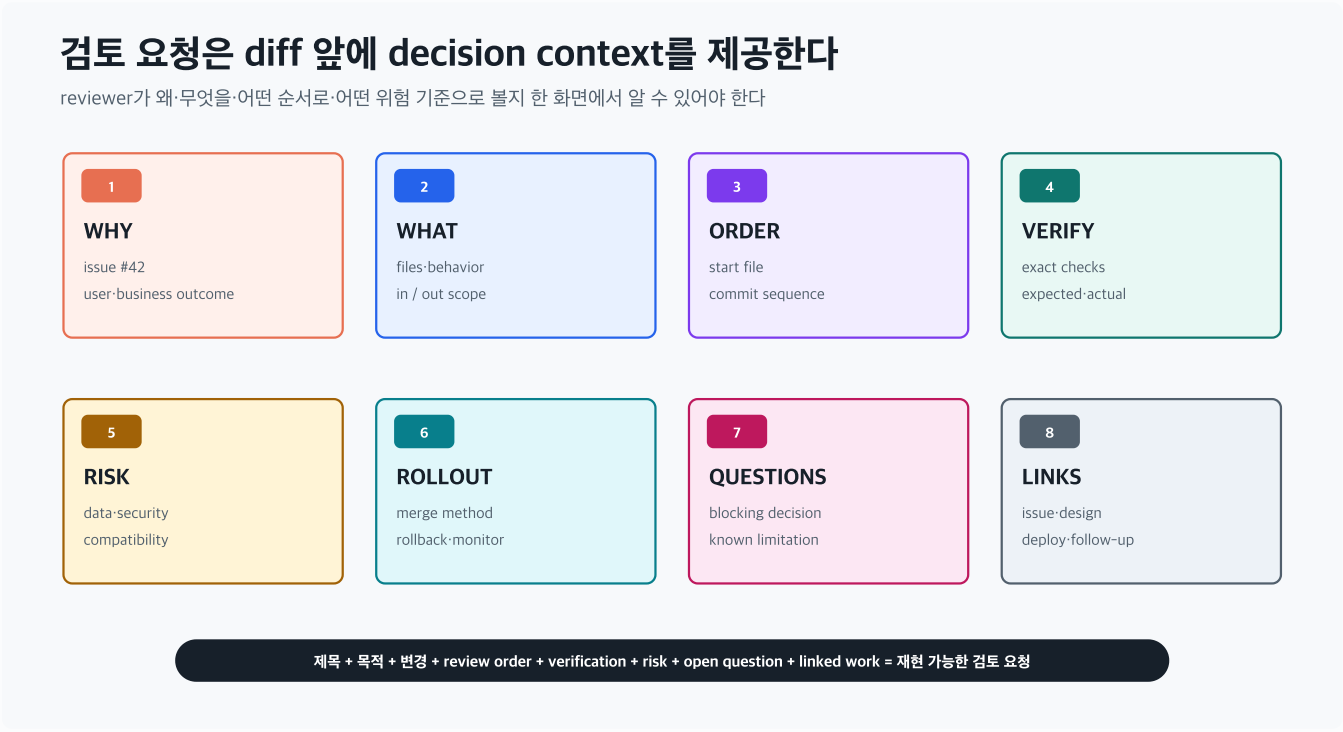


그림 7. reviewer가 diff만 보고 issue, risk, test, open question을 다시 추론하게 하지 않습니다.

1

WHY

issue #42

user·business outcome

2

WHAT

files·behavior

in / out scope

5

RISK

data·security

compatibility

6

ROLLOUT

merge method

rollback·monitor

제목 + 목적 + 변경 + review order + verification + risk

3

ORDER

start file

commit sequence

4

VERIFY

exact checks

expected·actual

7

QUESTIONS

blocking decision

known limitation

8

LINKS

issue·design

deploy·follow-up

+ open question + linked work = 재현 가능한 검토 요청

GitHub는 Pull Request(PR), GitLab은 Merge Request(MR)라는 이름을 주로 씁니다. 이 매뉴얼에서는 두 platform을 함께 가리킬 때 “검토 요청”이라고 부릅니다.

8.1. 검토 요청의 여덟 칸

```
1. linked issue와 requested decision
2. problem과 expected outcome
3. in scope / out of scope
4. base·head·merge base·current head OID
5. change sequence와 중요한 diff
6. risk·security·data·compatibility 영향
7. exact checks·result·head OID
8. open question·blocking point·follow-up
```

8.2. v1 검토 요청

```
Decision requested
  issue #42 badge contract가 acceptance를 충족하는지 검토

Range
  base 90640e5 → head cb116d4

Change
  badge visible·label output 추가
  positive test 추가

Checks
  node --test
  tests 4, pass 4, fail 0

Known risk
  hidden label behavior를 별도로 확인해야 함
```

4/4 test pass는 작성된 네 test가 통과했다는 뜻입니다. issue의 negative criterion을 증명하지는 않습니다.

8.3. diff를 읽기 쉽게 안내합니다

```
Start here:
  src/progress.js output contract

Then:
  test/badge.test.js positive·negative cases

Generated:
  none

Out of scope:
  UI·CSS·localization·deployment
```

reviewer의 인지 부하를 줄이는 설명은 review를 대신하지 않습니다. 중요한 file과 decision 순서를 알려 줍니다.

9. draft·ready·re-review 상태를 구분합니다

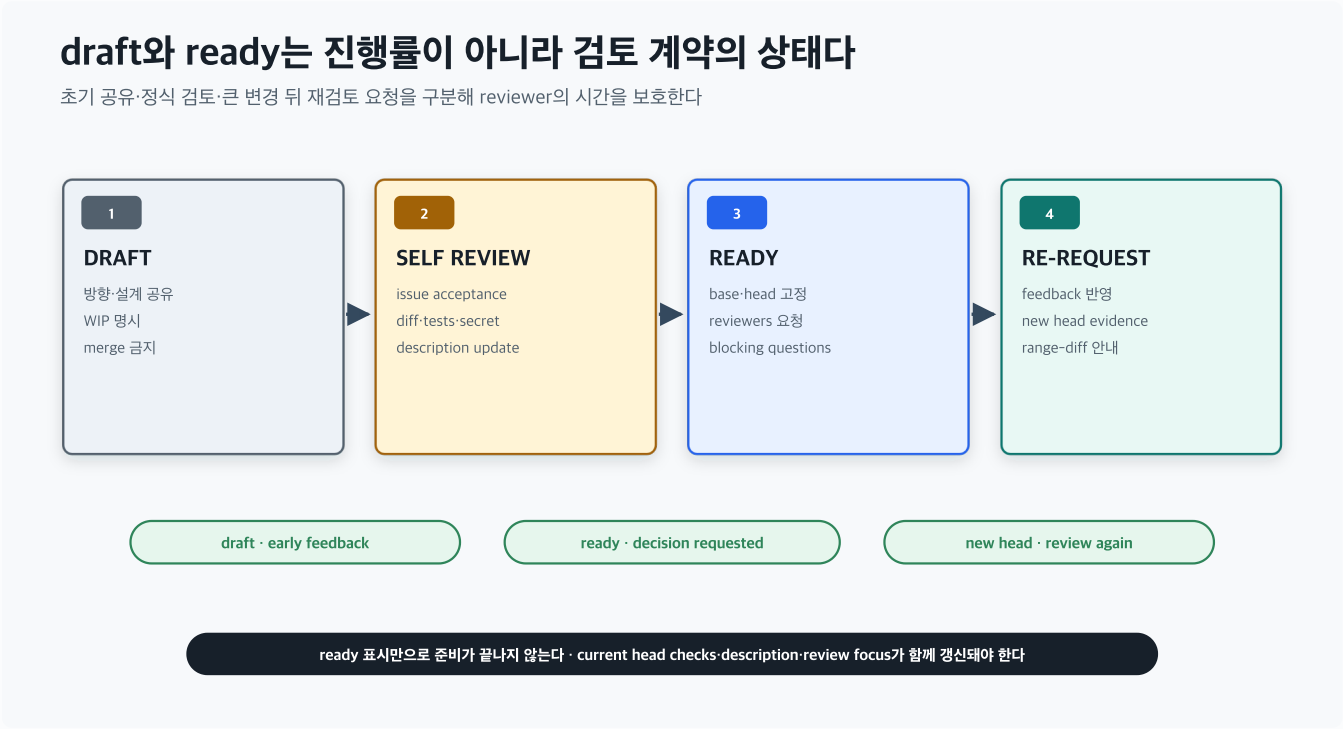
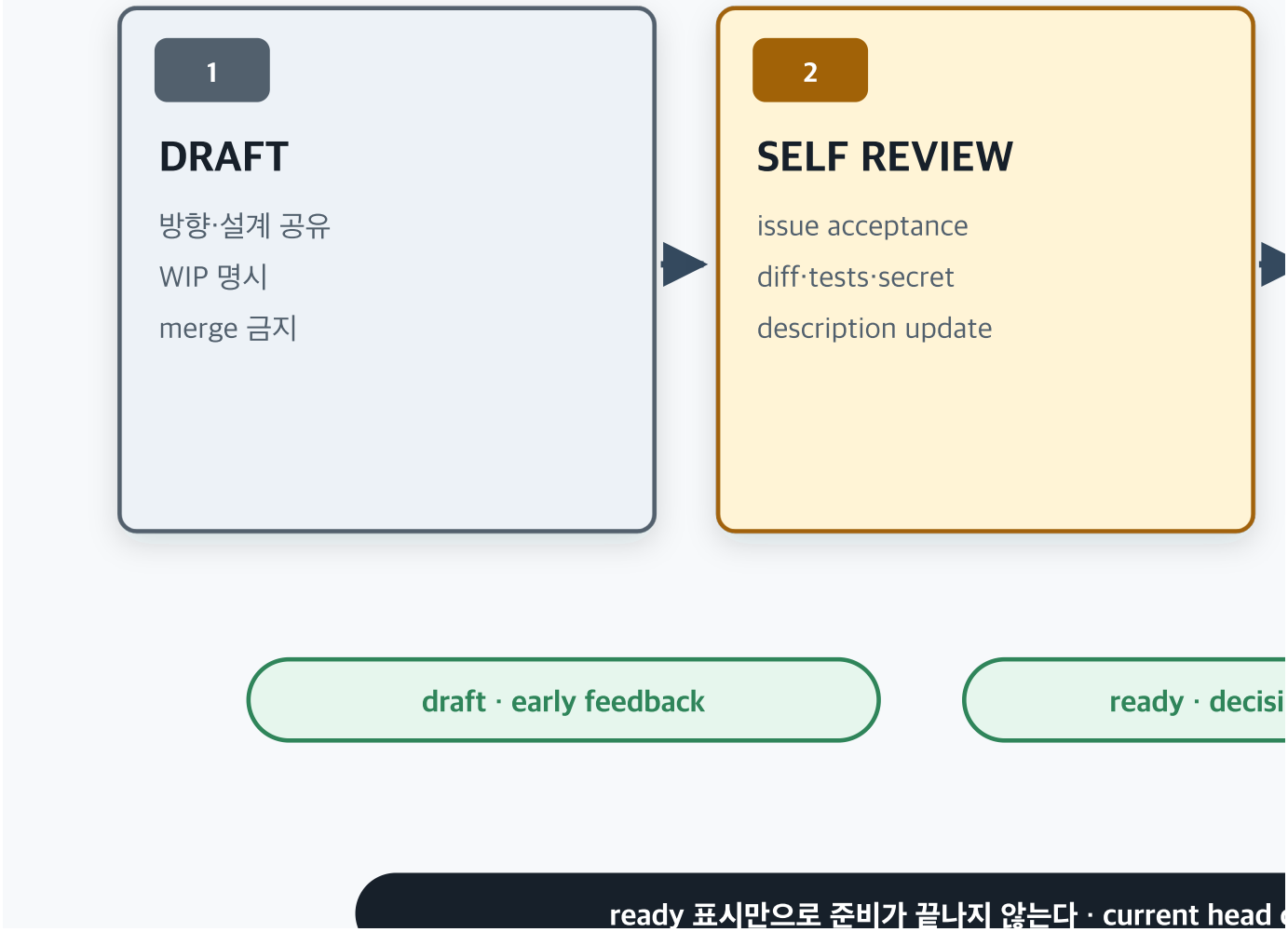
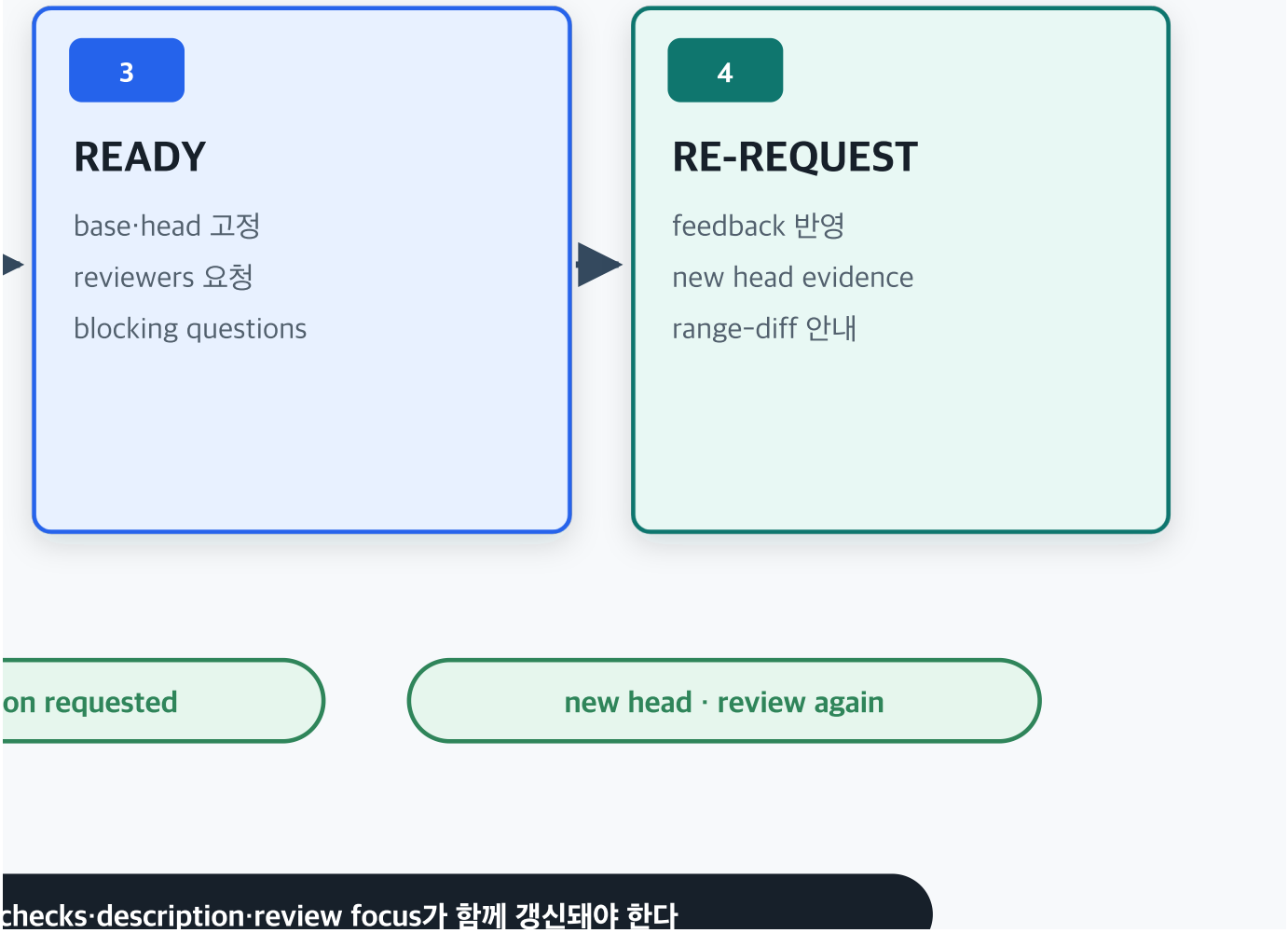


그림 8. draft는 “아무것도 보지 말라”가 아니라 decision 전 조기 feedback을 받을 수 있는 상태입니다.


1
DRAFT
방향·설계 공유
WIP 명시
merge 금지
2
SELF REVIEW
issue acceptance
diff·tests·secret
description update
draft · early feedback
ready · decisi
ready 표시만으로 준비가 끝나지 않는다 · current head


3
READY
base·head 고정
reviewers 요청
blocking questions
4
RE-REQUEST
feedback 반영
new head evidence
range-diff 안내
on requested
new head · review again
checks·description·review focus가 함께 갱신돼야 한다

9.1. draft

draft 단계에서 유용한 질문:

- contract 방향이 맞는가
- scope를 나눠야 하는가
- migration·security owner를 일찍 불러야 하는가
- spike 결과를 production change와 분리해야 하는가

draft는 incomplete 상태를 숨기는 label이 아닙니다. 아직 결정하지 말아야 할 부분과 원하는 feedback을 씁니다.

9.2. ready

ready 전 self-review:

```
[ ] issue acceptance와 diff를 한 줄씩 대조했다.
[ ] unintended file·secret·generated artifact가 없다.
[ ] current head OID를 기록했다.
[ ] exact checks가 current head에서 실행됐다.
[ ] risk·rollback·open question을 적었다.
[ ] reviewer와 required owner가 정해졌다.
```

9.3. re-review

새 push가 작고 comment typo만 고쳤는지, contract·data·security를 바꿨는지에 따라 re-review 범위가 다릅니다. 다음을 적습니다.

```
previous reviewed head =
current head =
resolution commits =
which threads addressed =
checks rerun =
substantial change =
```

platform과 repository rule에 따라 new push가 approval을 dismiss하거나 check를 다시 요구할 수 있습니다.

10. reviewer는 여덟 lens로 읽습니다

그림 9. diff line만 보지 않고 outcome·state·boundary·operation까지 여러 lens로 읽습니다.

1

행동

acceptance·edge

2

데이터

shape·migration

5

운영

timeout·rollback

6

테스트

positive·negative

review comment = 관찰 원칙 + 영향 + 기대

3

보안

trust·secret

4

권한

actor·deny

7

문서

contract·owner

8

접근성

hidden·label

contract + 재현 evidence + blocking 여부

lens	핵심 질문
intent	issue가 요구한 outcome인가
correctness	정상·오류·경계 입력에서 맞는가
state	loading·retry·partial·concurrent state는
data	schema·null·migration·retention 영향은
security	authn·authz·secret·injection boundary는
compatibility	client·API·config·version 호환성은
operation	log·metric·alert·rollback·deploy 영향은
test	acceptance와 risk를 실제로 증명하는가

comment를 evidence로 씁니다

약한 comment:

```
이 코드 이상합니다.
```

강한 comment:

```
BLOCKING · hidden label contract

score=74이면 badge.visible=false지만 label="수료 가능"이 남습니다.
accessibility tree와 analytics가 완료 상태로 해석할 수 있습니다.
issue #42 criterion 2에 따라 hidden state의 label은 null이어야 합니다.
negative regression test도 함께 필요합니다.
```

severity와 blocking

```
BLOCKING
NON-BLOCKING
QUESTION
SUGGESTION
PRAISE
```

label 자체가 platform gate가 되는지는 repository rule로 확인합니다. GitHub의 “Request changes”도 branch protection·rule에 연결되어야 merge blocking으로 작동하며, GitLab approval·thread policy도 configuration에 따라 다릅니다.

30초 확인 3

같은 comment라도 “blocking behavior bug”와 “후속 refactor 제안”을 왜 분리해야 합니까?

11. feedback은 verified resolution으로 닫습니다

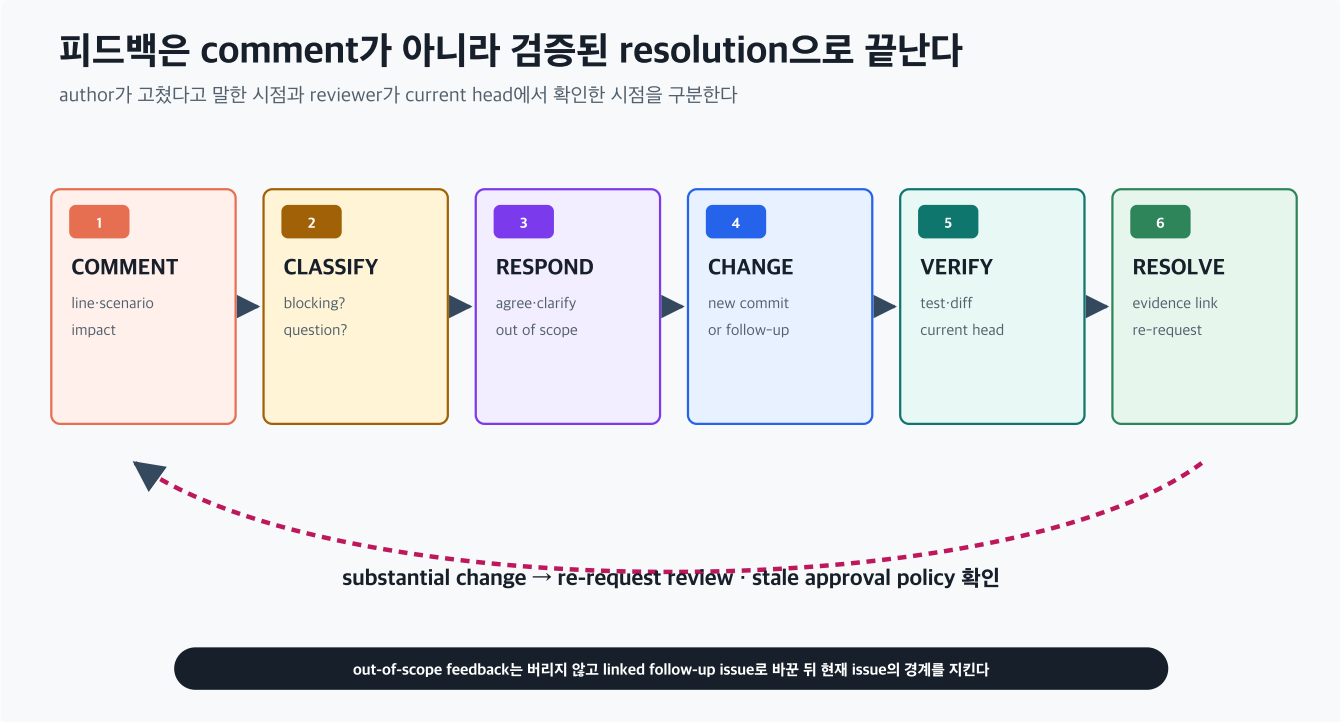
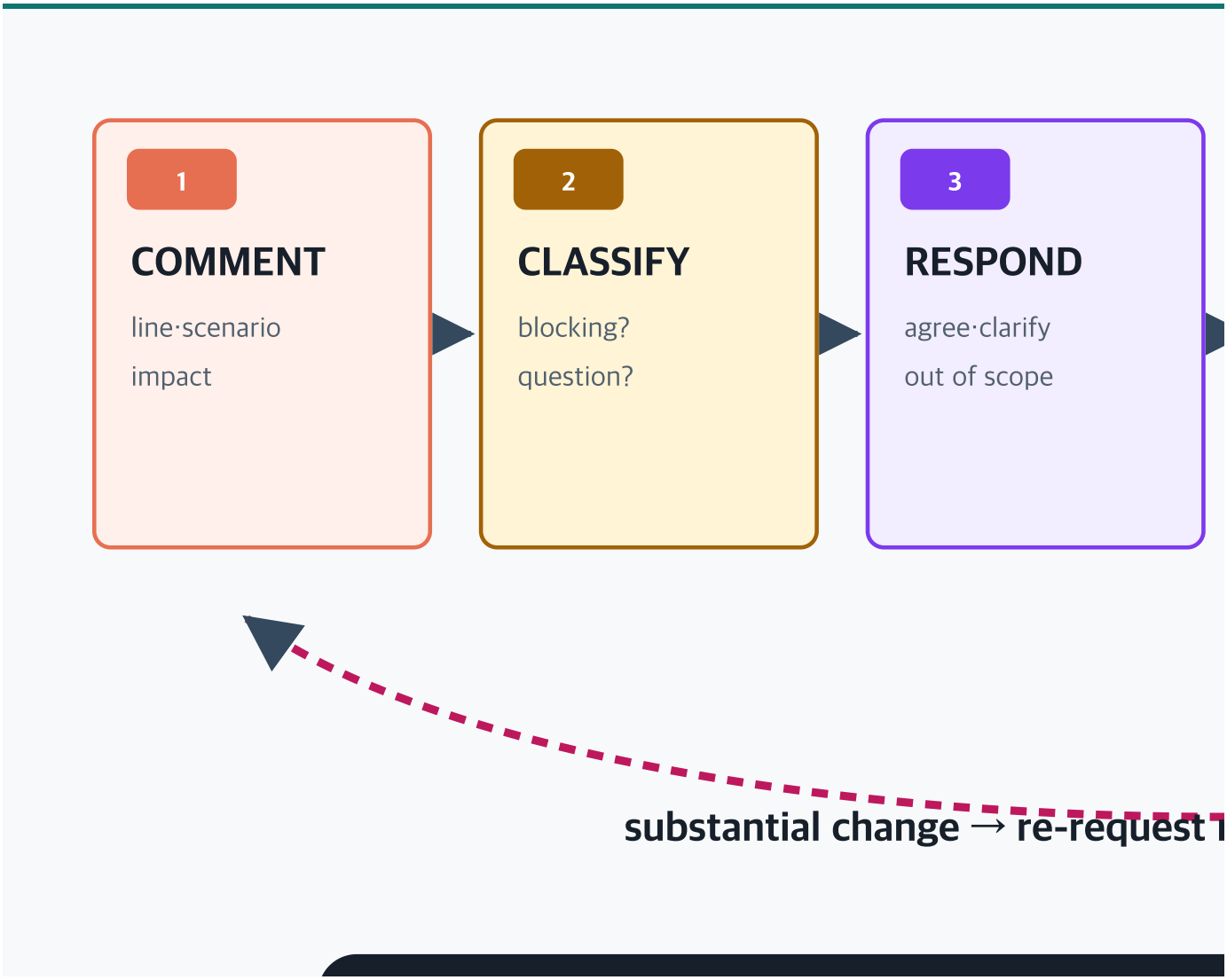
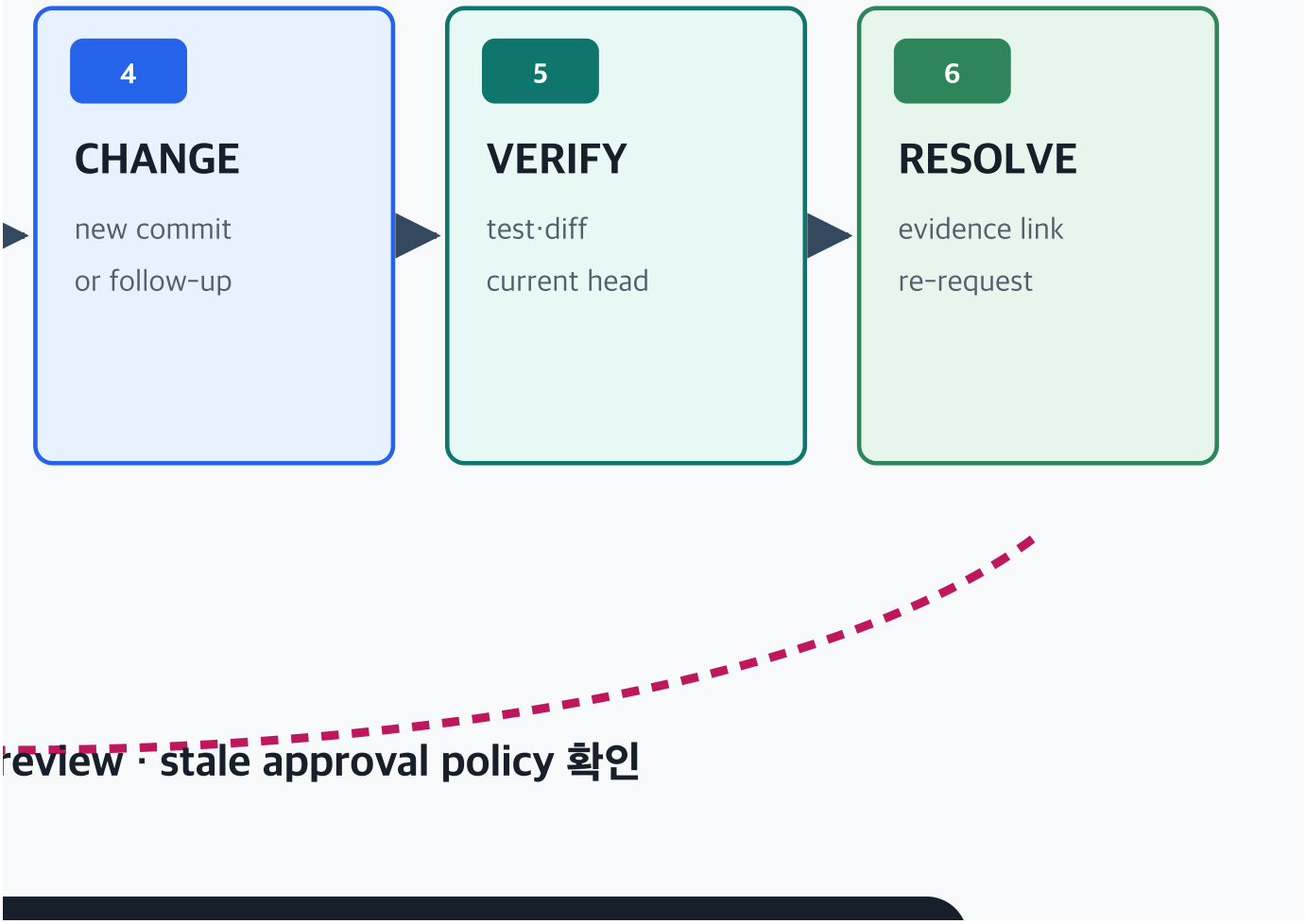


그림 10. “수정했습니다”는 중간 응답입니다. thread가 가리킨 gap이 current head에서 사라졌다는 verification이 필요합니다.


1
COMMENT
line·scenario
impact
2
CLASSIFY
blocking?
question?
3
RESPOND
agree·clarify
out of scope
substantial change → re-request


4
CHANGE
new commit
or follow-up
5
VERIFY
test·diff
current head
6
RESOLVE
evidence link
re-request
review · stale approval policy 확인

11.1. feedback thread의 여섯 요소

요소	실습 T01
observation	hidden인데 label이 남음
scenario	progress 100, score 74
impact	accessibility·analytics 오해
expected	label null
blocking	yes
evidence anchor	v1 line·OID

11.2. author response

```
동의합니다.
32700a6에서 hidden label을 null로 바꾸고
negative regression test를 추가했습니다.

Verification
  node --test
  tests 5 · pass 5 · fail 0

Head
  32700a6...
```

“반영 완료”만 쓰지 않고 decision, resolution commit, verification을 연결합니다.

11.3. resolve의 책임

repository policy에 따라 author, reviewer, maintainer 중 누가 thread를 resolve할 수 있는지 다릅니다. 최소 판단:

```
comment가 가리킨 source를 확인했는가?
expected behavior와 일치하는가?
regression evidence가 있는가?
current head에서 확인했는가?
new risk가 생기지 않았는가?
```

11.4. force push 후 anchor

history rewrite로 comment가 가리킨 old commit이 사라질 수 있습니다. old head, new head, resolution patch의 대응을 기록합니다.

```
git range-diff --no-color \
  90640e5..origin/review/42-v1 \
  90640e5..origin/feature/42-completion-badge
```

실습:

```
1: 0c96b36 = 1: 0c96b36 feat: add completion badge result
2: cb116d4 = 2: cb116d4 test: cover visible completion badge
-: ------- > 3: 32700a6 fix: hide label when completion badge is hidden
```

`range-diff`는 두 commit series의 patch 대응을 사람이 읽게 돕습니다. output 형식은 안정된 machine-readable interface로 약속되지 않으므로 automation parser로 가정하지 않습니다.

12. current head evidence를 다시 묶습니다

12.1. current diff

```
git diff --stat \
  origin/main...origin/feature/42-completion-badge
```

```
src/progress.js    |  7 ++++++-
test/badge.test.js | 13 +++++++++++++
2 files changed, 19 insertions(+), 1 deletion(-)
```

12.2. current checks

```
head OID: 32700a6...

node --test
tests 5
pass 5
fail 0

secret-scan
conclusion success
head 32700a6...
```

check 이름과 green icon만 기록하지 않습니다.

반드시 연결할 값	이유
check name	어떤 gate인지
workflow·job run	재현·log 위치
head OID	지금 검토 중인 source인지
conclusion	success·failure·skipped 구분
test count	발견된 test 범위
environment	OS·runtime·dependency 조건

12.3. test pass의 한계

v1에서 tests 4/4가 통과했지만 hidden label criterion은 test에 없었습니다. test pass를 읽을 때 두 질문을 분리합니다.

```
실행된 test는 모두 통과했는가?
필요한 acceptance와 risk가 test에 포함됐는가?
```

coverage percentage도 동일합니다. line이 실행됐다는 사실과 올바른 assertion이 있다는 사실은 다릅니다.

13. merge readiness는 독립 gate의 집합입니다

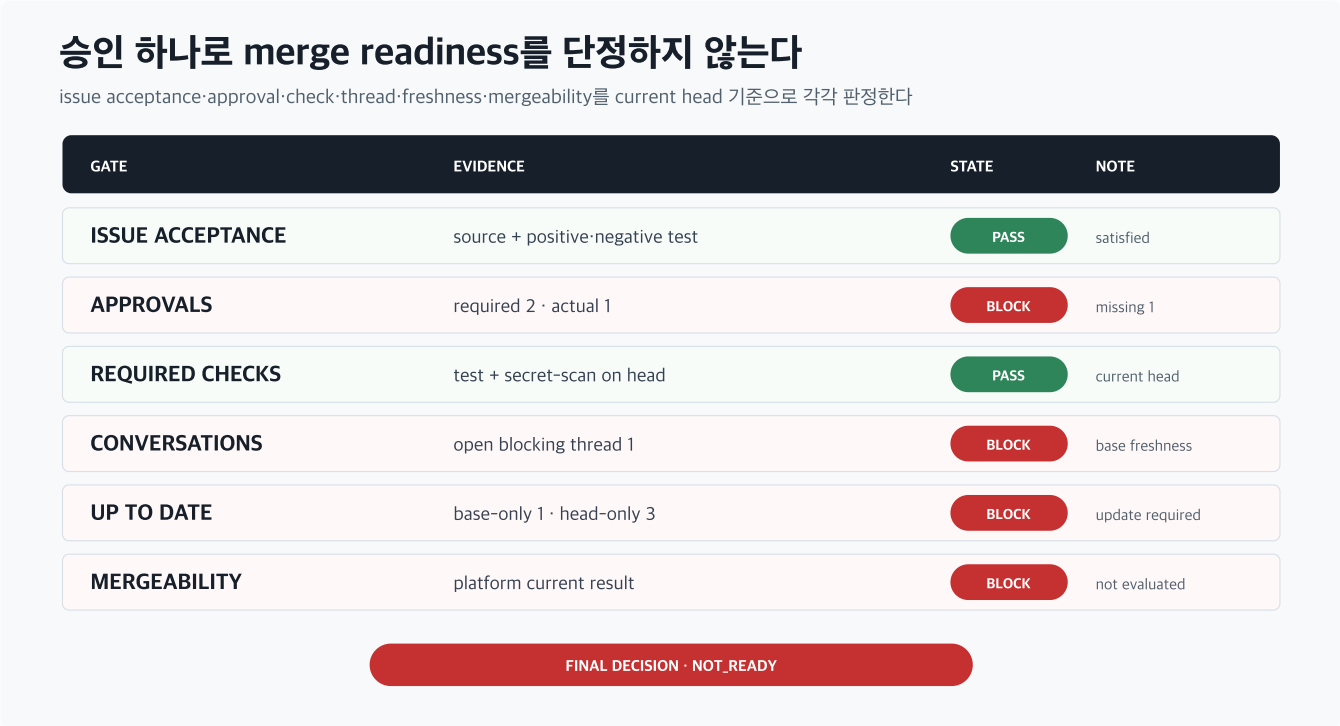


그림 11. 한 gate의 PASS가 다른 gate를 대신하지 않습니다. 모든 required gate를 current head와 current policy 기준으로 평가합니다.

GATE	EVIDENCE
ISSUE ACCEPTANCE	source + positive·negative t
APPROVALS	required 2 · actual 1
REQUIRED CHECKS	test + secret-scan on head
CONVERSATIONS	open blocking thread 1
UP TO DATE	base-only 1 · head-only 3
MERGEABILITY	platform current result

	STATE	NOTE
test	PASS	satisfied
	BLOCK	missing 1
	PASS	current head
	BLOCK	base freshness
	BLOCK	update required
	BLOCK	not evaluated

13.1. 여섯 gate

gate	실습 evidence	state
issue acceptance	source + positive·negative test	PASS
required approvals	required 2, actual 1	BLOCK
required checks	test + secret-scan on current head	PASS
conversations	open blocking thread 1	BLOCK
up to date	base-only 1, head-only 3	BLOCK
mergeability	platform current result 없음	BLOCK

따라서 decision:

```
NOT_READY
```

13.2. approval은 무엇을 뜻하는가

approval은 reviewer가 특정 head와 context를 검토했다는 신호입니다. 그러나 다음을 자동 보장하지 않습니다.

- issue acceptance 전체 충족
- required check current head 통과
- unresolved blocking thread 0
- branch current base 포함
- merge conflict 없음
- deploy·rollback 준비 완료

13.3. check와 policy

GitHub protected branch와 ruleset, GitLab approval rule·status check·merge check는 repository별로 다릅니다. UI button 색, 일반적인 blog 설명, 다른 repository 경험으로 현재 policy를 단정하지 않습니다.

```
policy source URL·path =
required approval count =
code owner rule =
required check names =
conversation rule =
freshness rule =
signed commit rule =
merge queue·train rule =
allowed merge method =
observed at =
```

13.4. status가 오래될 수 있습니다

new push, base update, check rerun, approval dismissal, policy edit는 readiness를 바꿀 수 있습니다. final decision에는 current head OID와 decision time을 붙입니다.

READINESS EQUATION READY는 “approve 하나 + green 하나”가 아닙니다.

required issue evidence ∧ approvals ∧ checks ∧ conversations ∧ freshness ∧ mergeability ∧ current policy

14. merge method는 history와 trace를 바꿉니다

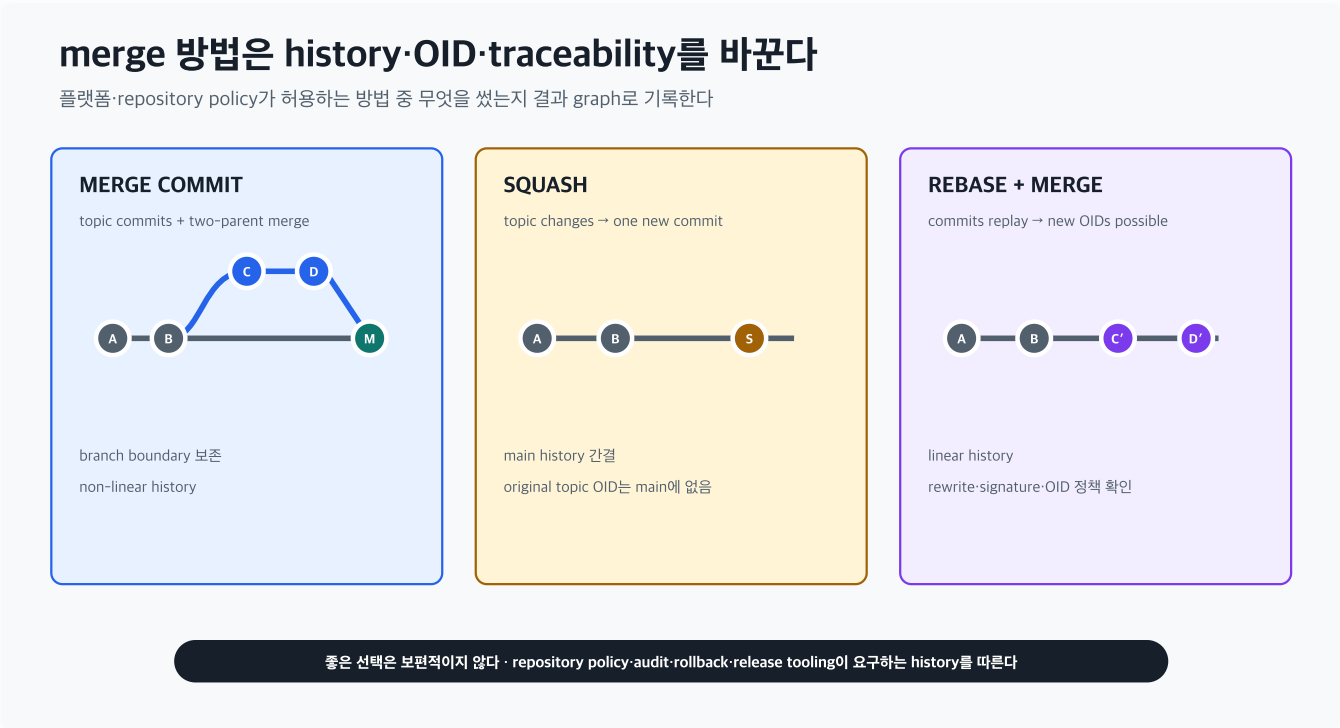
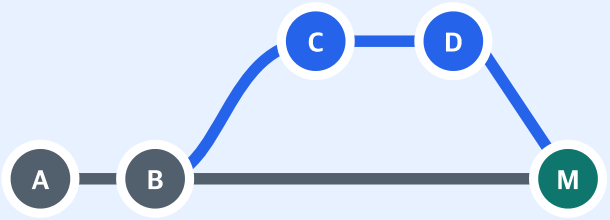


그림 12. merge method 선택은 미관 문제가 아닙니다. main history, 원래 OID, rollback, signature, audit trace를 바꿉니다.

MERGE COMMIT

topic commits + two-parent merge



branch boundary 보존

non-linear history

SQUASH

topic changes → one new

main history 간결

original topic OID는 main

좋은 선택은 보편적이지 않다 · repository policy·audit

commit

에 없음

REBASE + MERGE

commits replay → new OIDs possible

linear history

rewrite·signature·OID 정책 확인

rollback·release tooling이 요구하는 history를 따른다

method	main에 남는 모양	주의할 trace
merge commit	topic commits + merge commit	non-linear history·merge parent
squash	topic 전체를 새 commit 하나로	original topic OID가 main에 없음
rebase and merge	replay된 linear commits	OID·signature가 달라질 수 있음

repository가 어떤 method를 허용하는지, 자동 생성 message가 issue closure·changelog에 어떤 영향을 주는지 확인합니다.

merge 전 기록

```
final head OID =
base OID =
selected method =
expected resulting commits =
approval owner =
rollback owner·method =
release linkage =
```

merge 후 기록

```
merged at =
merged by =
base after merge =
merge or squash commit OID =
source branch state =
issue state =
```

15. MERGED와 DEPLOYED를 분리합니다

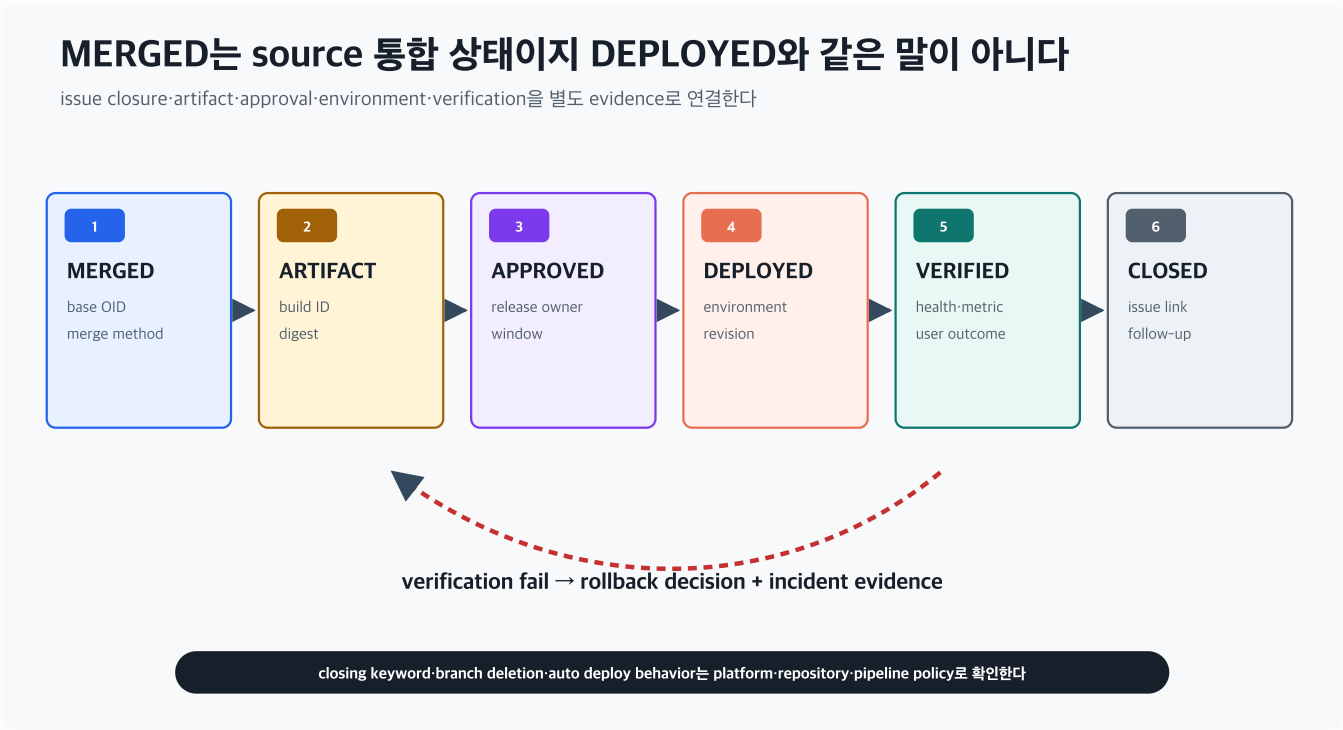
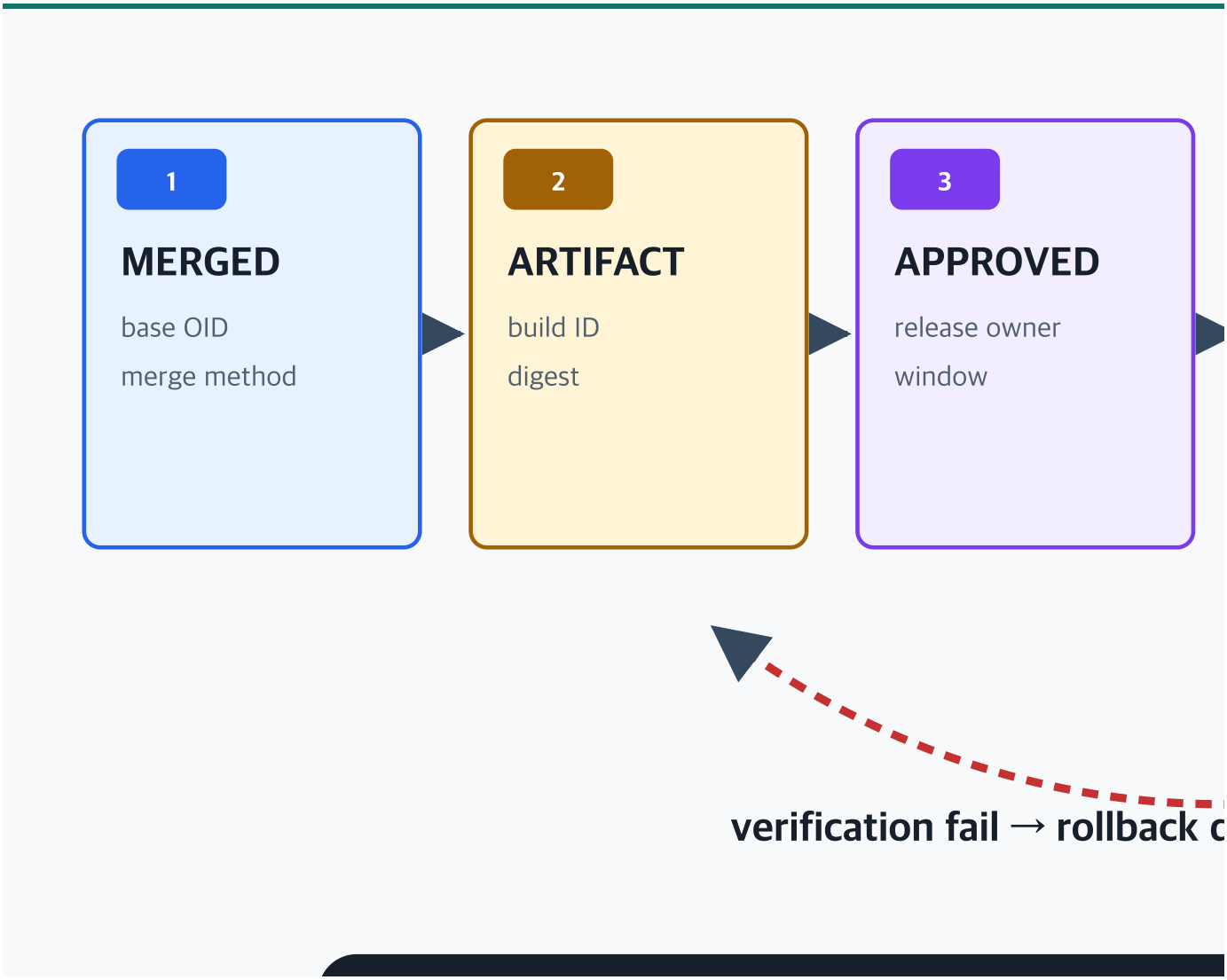
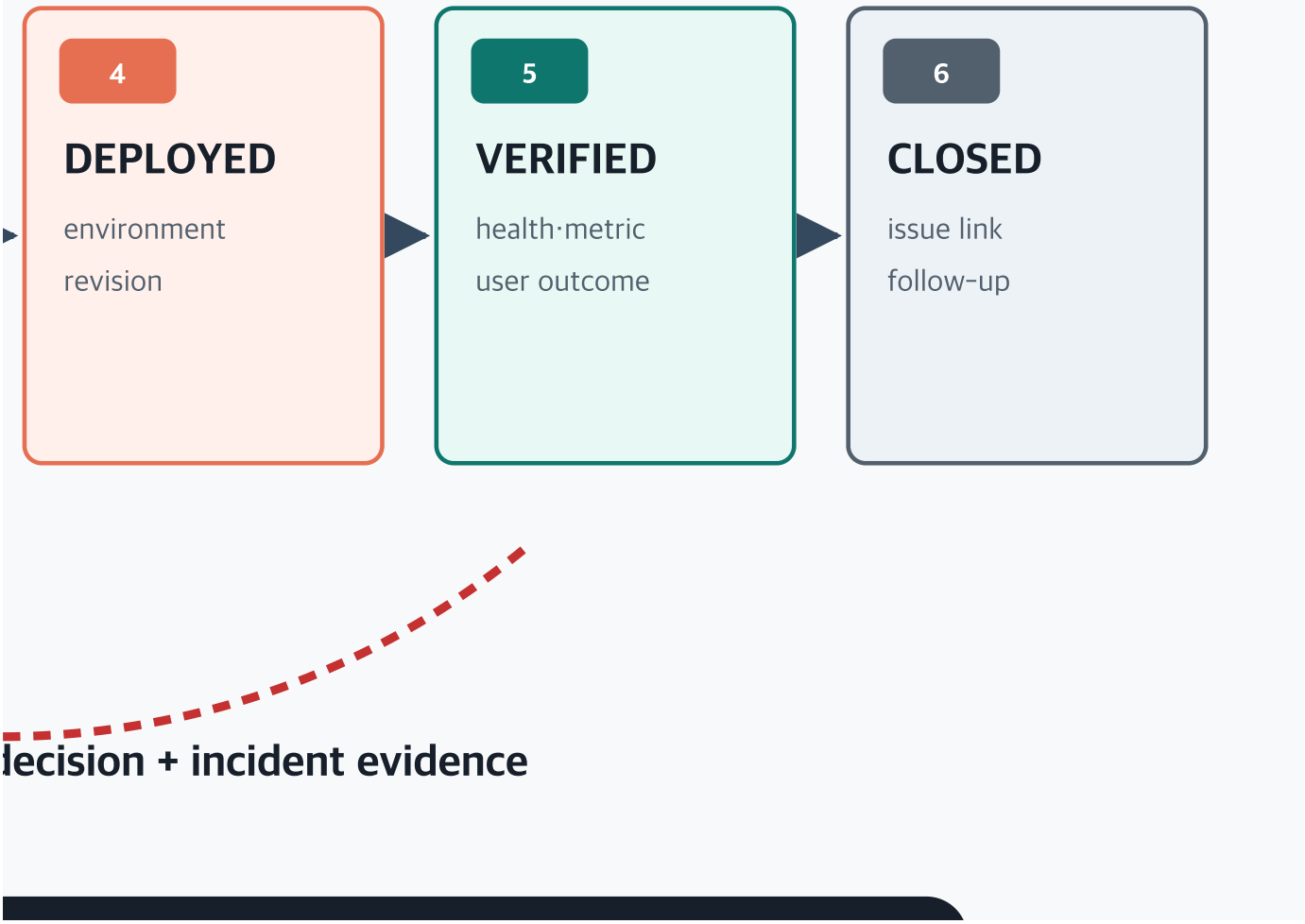


그림 13. source 통합, artifact 생성, release 승인, environment 배포, user outcome 검증은 서로 다른 상태입니다.


1
MERGED
base OID
merge method
2
ARTIFACT
build ID
digest
3
APPROVED
release owner
window
verification fail → rollback c


4
DEPLOYED
environment
revision
5
VERIFIED
health·metric
user outcome
6
CLOSED
issue link
follow-up
lecision + incident evidence

```
MERGED
→ ARTIFACT BUILT
→ RELEASE APPROVED
→ DEPLOYED
→ VERIFIED
→ ISSUE CLOSED
```

각 단계 evidence:

상태	evidence
merged	result OID·method·base
artifact	build ID·digest·provenance
approved	owner·window·change ticket
deployed	environment·revision·time
verified	health·metric·synthetic·user outcome
closed	issue decision·follow-up link

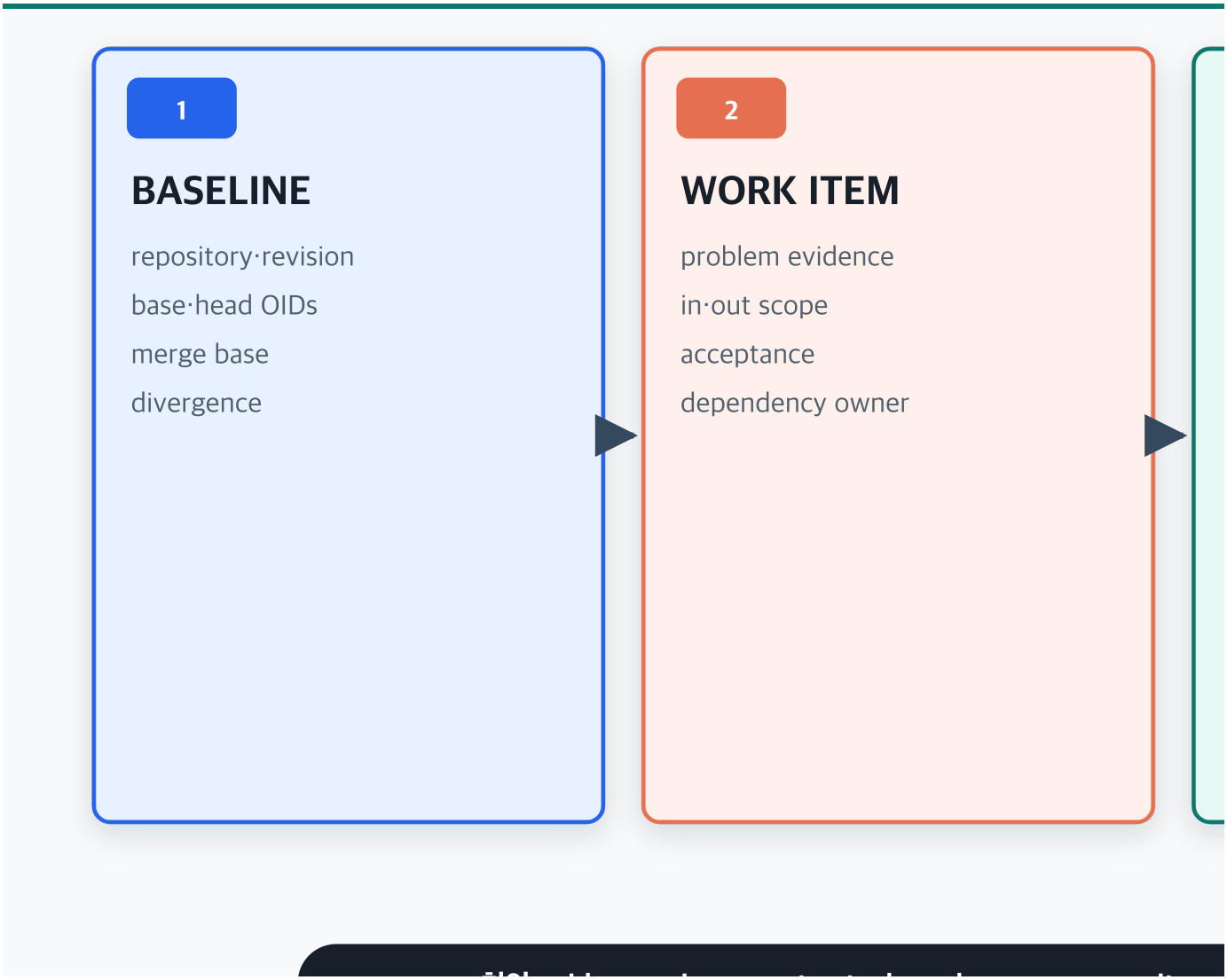
closing keyword가 issue를 자동으로 닫는지, default branch에 merge될 때만 동작하는지, cross-repository에서 지원되는지는 platform 공식 문서와 repository 설정으로 확인합니다. 자동 close는 deployment 성공을 의미하지 않습니다.

branch 삭제

source branch 삭제는 정리 방법이지 evidence 삭제가 아닙니다. review request, commit object 보존 정책, release trace, open follow-up을 확인한 뒤 team policy에 따라 정리합니다.

16. 협업 evidence packet을 완성합니다

그림 14. 최종 기록은 baseline, work item, change·review, decision·follow-up 네 묶음으로 요약할 수 있습니다.


1
BASELINE
repository·revision
base·head OIDs
merge base
divergence
2
WORK ITEM
problem evidence
in·out scope
acceptance
dependency owner

3

CHANGE + REVIEW

commit sequence

three-dot diff

checks on head

threads·approvals

4

DECISION + FOLLOW-UP

READY / NOT_READY

blocking reason

merge method

deploy evidence link

packet A · baseline

```
repository·remote identity
Git·platform version or observed documentation date
base·head·merge base full OIDs
working tree·fetch time
divergence
```

packet B · work item

```
problem evidence
expected outcome
in·out scope
acceptance
dependency·owner
```

packet C · change and review

```
commit sequence
review diff
checks on current head
review comments·responses
resolution commits·tests
approvals·open threads
```

packet D · decision and follow-up

```
READY / NOT_READY
blocking reasons
selected merge method
merge result OID
deploy evidence link
verification result
follow-up issues
```

17. 협업 검토 데스크 실습

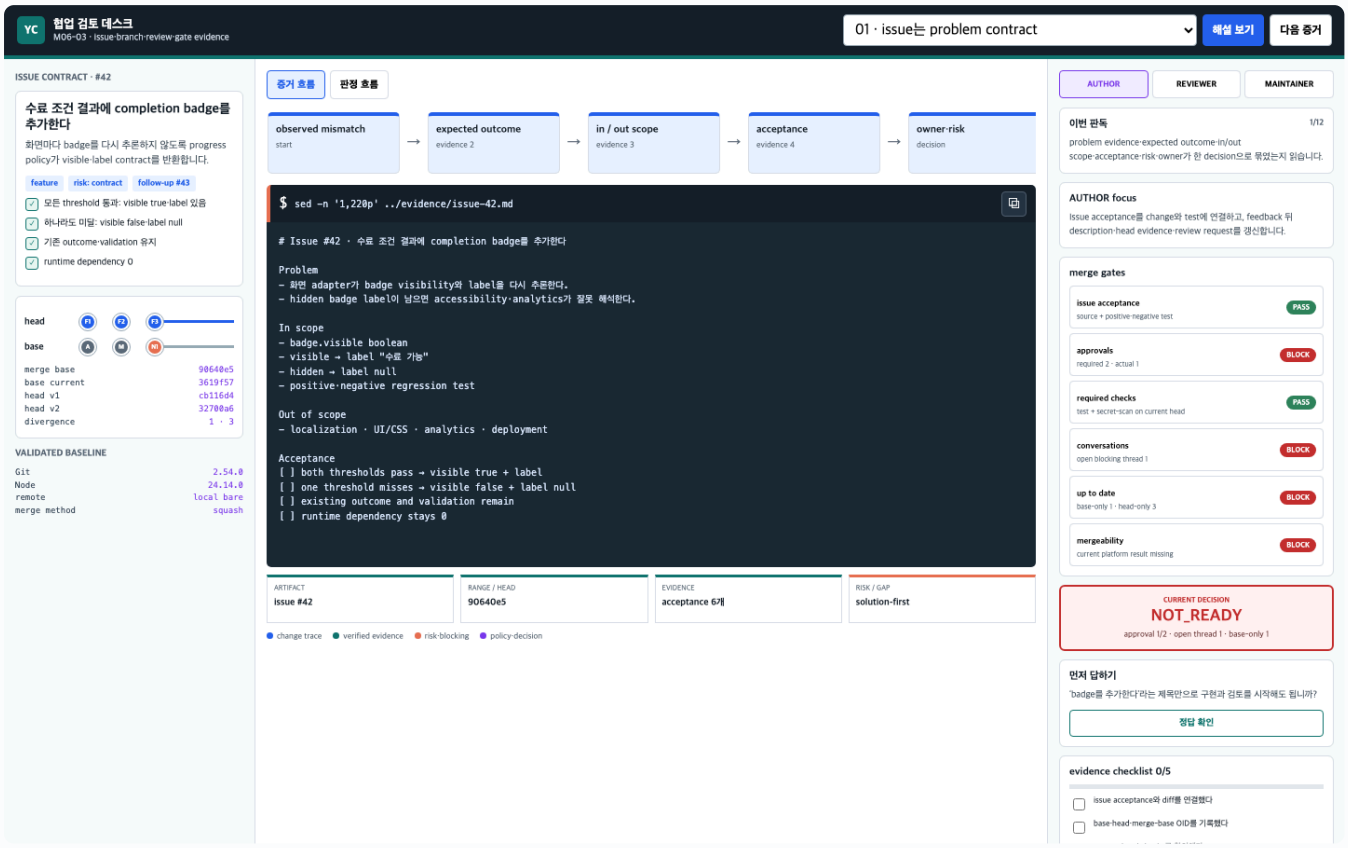


그림 15. 12개 장면에서 author·reviewer·maintainer 역할을 바꾸며 같은 evidence를 서로 다른 책임으로 판독합니다.

17.1. 실습 구성

생성된 folder:

```
gibalja-collaboration-practice/
├── remote.git/
├── author/
├── reviewer/
└── evidence/
    ├── issue-42.md
    ├── review-request-v1.md
    ├── review-request-v2.md
    ├── review-threads.md
    ├── merge-policy.json
    ├── checks-v2.json
    ├── readiness-v2.json
    └── project-facts.txt
```

17.2. 실습 history

```
* 32700a6 feature head v2 · fix + negative test
* cb116d4 review v1 · positive test
* 0c96b36 feature · badge contract
| * 3619f57 current main · release window
|/
* 90640e5 merge base · issue context
* dc6324f initial project
```

17.3. 12개 장면

장면	판정
1	issue가 problem contract인가
2	scope를 follow-up으로 나눌 것인가
3	base·head·merge base가 고정됐는가
4	divergence와 freshness는
5	commit sequence가 review story인가
6	4/4 test에도 acceptance gap이 있는가
7	comment가 재현·영향·expected를 갖는가
8	feedback이 verified resolution인가
9	v1과 v2 series는 어떻게 달라졌는가
10	current diff·checks가 head와 연결되는가
11	merge gate를 각각 판정했는가
12	merged 이후 deploy evidence가 있는가

상세 command와 기록 방법은 단계별 실습서를 따릅니다.

18. 60분 실전 미션

미션 1 · baseline · 5분

```
repository root
remote
current branch·full OID
working tree
Git version
capture time
```

미션 2 · issue contract · 10분

실제 mismatch 하나를 골라 actor, current, expected, impact, reproduction, in·out scope, acceptance, owner를 씁니다.

미션 3 · range · 10분

base, head, merge base, divergence를 기록합니다. branch name과 full OID를 함께 씁니다.

미션 4 · commit plan · 10분

각 commit을 다음 형식으로 계획합니다.

```
commit purpose =
files =
observable behavior =
test evidence =
dependency =
rollback meaning =
```

미션 5 · review request · 10분

requested decision, start-here order, current head, exact checks, risk, open question을 씁니다.

미션 6 · feedback resolution · 10분

review comment 하나를 observation, scenario, impact, expected, blocking으로 씁니다. author response와 resolution evidence도 씁니다.

미션 7 · readiness · 5분

issue acceptance, approvals, checks, conversations, freshness, mergeability를 각각 PASS·BLOCK·UNKNOWN으로 판정합니다.

19. 자주 실패하는 협업 패턴

실패	왜 위험한가	고치는 evidence
“기능 추가”만 있는 issue	완료 판정 불가	problem·acceptance
branch 이름만 기록	ref가 움직임	full OID·time
base와 head 방향 누락	diff·merge target 혼동	repository·branch·OID
commit 제목만 검토	실제 snapshot 누락	`git show <oid>^!`
green check만 캡처	다른 head일 수 있음	check run·head OID
test pass=요구 충족	빠진 test gap	acceptance-test matrix
“수정했습니다”로 resolve	해결 확인 불가	resolution commit·test
approval 하나=READY	다른 gate 누락	gate별 state
버튼 색으로 policy 추정	configuration 차이	policy source
merged=deployed	사용자 반영 미확인	artifact·deploy·verify
follow-up을 comment에만 둠	작업 소실	linked issue·owner
force push 후 old anchor 삭제	review trace 약화	old/new head·range-diff

20. 셀프 테스트 · 먼저 답한 뒤 해설을 엽니다

Q1

issue 제목과 구현 아이디어는 있지만 expected outcome과 acceptance가 없습니다. branch를 바로 만들어도 됩니까?

정답 보기
아닙니다. 먼저 관찰된 mismatch, expected outcome, scope, 검증 가능한 acceptance와 owner를 보완합니다. 구현 아이디어는 후보이며 problem contract를 대신하지 않습니다.

Q2

`main`과 `feature/42` 이름만 기록하면 언제든 같은 diff를 재현할 수 있습니까?

정답 보기
아닙니다. branch ref는 움직입니다. base·head·merge base full OID, repository identity와 관찰 시각을 함께 기록합니다.

Q3

`git diff A...B`와 `git log A...B`의 세 점은 같은 의미입니까?

정답 보기
아닙니다. diff의 세 점은 merge base와 B tree의 차이를, log의 세 점은 한쪽에만 있는 commit의 symmetric difference를 다룹니다.

Q4

divergence가 `1 3`입니다. head는 base 최신 상태를 포함합니까?

정답 보기
아닙니다. left의 1은 base에만 있는 commit이 하나라는 뜻입니다. up-to-date가 required gate라면 update가 필요합니다.

Q5

commit 제목만 읽고 content를 검토해도 됩니까?

정답 보기
아닙니다. staged snapshot의 실제 tree diff, parent, message, metadata와 test evidence를 확인합니다.

Q6

v1 test가 4/4 PASS입니다. issue의 negative criterion도 충족했다고 단정할 수 있습니까?

정답 보기
아닙니다. 실행된 네 test만 통과했습니다. acceptance-test matrix에서 negative criterion의 source·test evidence가 있는지 따로 확인합니다.

Q7

review comment에 꼭 포함하면 좋은 다섯 요소는 무엇입니까?

정답 보기
observation, 재현 scenario, impact, expected behavior, blocking 여부입니다. 가능하면 file·line·commit anchor도 붙입니다.

Q8

author가 “수정했습니다”라고 답했습니다. 바로 resolve해도 됩니까?

정답 보기
아닙니다. resolution commit, current-head diff, targeted regression test를 확인하고 repository policy에 맞는 사람이 resolve합니다.

Q9

`range-diff` output을 안정된 자동화용 JSON처럼 parse해도 됩니까?

정답 보기
아닙니다. 두 patch series를 사람이 비교하도록 만든 output이며 format stability를 automation contract로 가정하지 않습니다.

Q10

required checks가 성공했고 reviewer 한 명이 approve했습니다. READY입니까?

정답 보기
현재 policy가 요구하는 approval 수, unresolved thread, base freshness, current mergeability, issue acceptance를 각각 확인하기 전에는 READY라고 할 수 없습니다.

Q11

squash merge 뒤 original feature commit OID가 main history에 그대로 남습니까?

정답 보기
보통 squash 결과는 새 commit 하나이며 original topic commits는 main ancestry에 그대로 들어가지 않습니다. platform과 repository 동작을 확인하고 result OID를 기록합니다.

Q12

검토 요청이 merged 상태입니다. 사용자에게 기능이 전달됐다고 말해도 됩니까?

정답 보기
아닙니다. artifact build, release approval, target environment deployment, health·metric·user outcome verification evidence가 별도로 필요합니다.

21. 최종 제출 체크리스트

issue

- [ ] problem evidence와 expected outcome이 분리되어 있다.
- [ ] in scope와 out of scope가 있다.
- [ ] acceptance가 입력·상태·관찰 결과로 판정 가능하다.
- [ ] dependency·owner·follow-up이 연결돼 있다.

branch·commit

- [ ] base·head·merge base full OID와 시각이 있다.
- [ ] divergence의 left·right를 방향 있게 해석했다.
- [ ] commit별 purpose·files·test·risk가 있다.
- [ ] working tree·staged diff·untracked를 확인했다.

review

- [ ] requested decision과 start-here가 있다.
- [ ] current head의 exact diff·checks를 기록했다.
- [ ] comment가 observation·impact·expected·blocking을 갖는다.
- [ ] resolution commit·test·re-review evidence가 있다.

merge·post-merge

- [ ] required gate를 각각 PASS·BLOCK·UNKNOWN으로 판정했다.
- [ ] policy source와 관찰 시각이 있다.
- [ ] merge method와 result OID가 있다.
- [ ] merged·artifact·deployed·verified를 분리했다.

22. 공식 자료와 확인 범위

Git

- git-switch 공식 문서
- git-branch 공식 문서
- git-merge-base 공식 문서
- git-rev-list 공식 문서
- git-diff 공식 문서
- git-log 공식 문서
- git-commit 공식 문서
- git-interpret-trailers 공식 문서
- git-range-diff 공식 문서
- git-merge 공식 문서
- git-push 공식 문서

GitHub

- Issues로 작업 추적하기
- Issue quickstart
- Issue dependencies
- Pull Request 소개
- Pull Request 만들기
- 변경 검토 돕기
- Pull Request review
- Protected branches
- GitHub merge methods

- Issue·Pull Request closing keywords

GitLab

- Merge Requests
- Merge Request reviews
- Merge Request approvals
- Merge methods
- External status checks
- Merge Request commits
- Merge Request changes
- Create issues
- Description templates

적용 원칙

공식 문서도 product plan, repository configuration, deployment model, version에 따라 실제 UI와 gate가 달라질 수 있습니다. 이 매뉴얼은 platform-neutral evidence model을 가르치며, 실제 merge 권한과 policy는 현재 repository에서 다시 확인합니다.

다음 매뉴얼: M07-01「AI에 줄 프로젝트 맥락 작성하기」에서 이 협업 evidence를 AI가 오해하지 않는 project context document로 압축합니다.