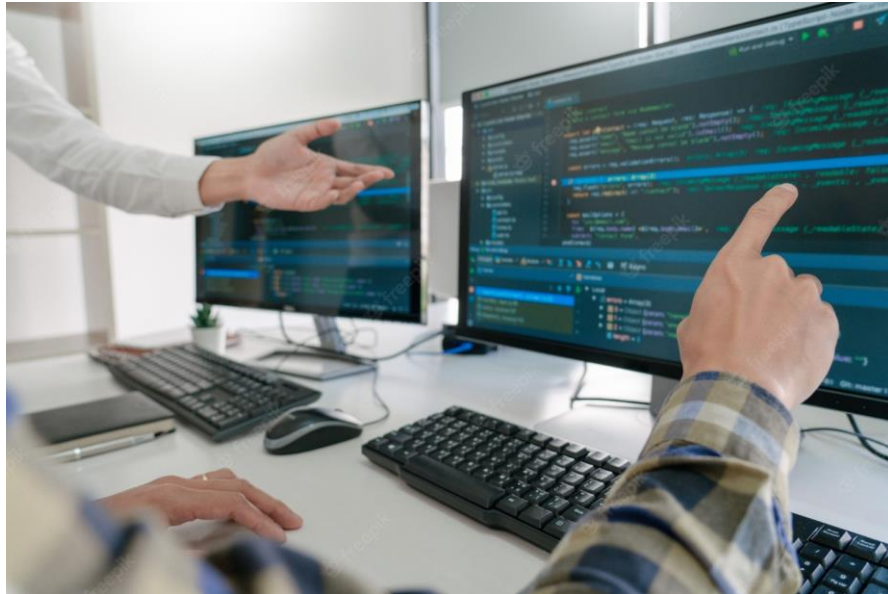
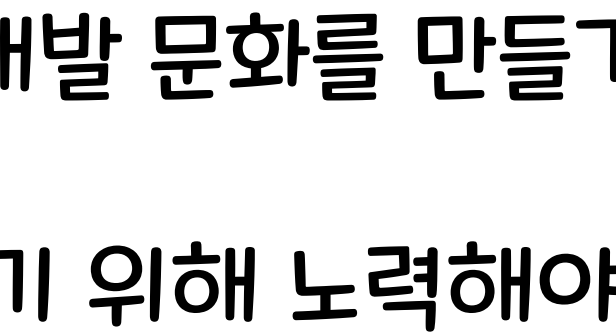


K-Digital Training 웹 풀스택 과정

git 협업

개발문화



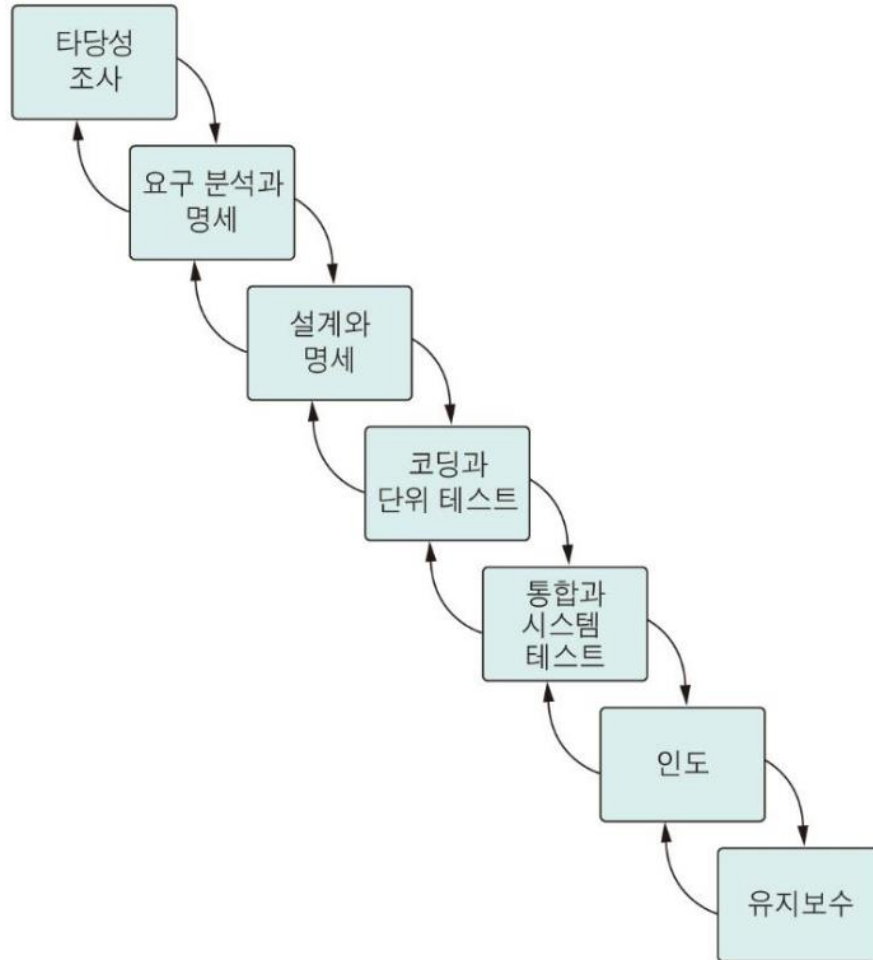


좋은 개발 문화를 만들기 위해,
수립하기 위해 노력해야 하는 것!



Agile (애자일)

Waterfall



Waterfall Model (폭포수 모델)

- 가장 익숙한 소프트웨어 개발 기법
- 고전적인 소프트웨어 생명 주기
- 병행 수행되지 않고 순차적으로 수행

Q. Waterfall Model의
장단점으로는 무엇이 있을까요?

Waterfall Model의 장단점

장점

- 단순한 모델이라 이해가 쉽다.
- 단계별로 정형화된 접근이 가능해 문서화가 가능하다.
- 프로젝트 진행 상황을 한눈에 명확하게 파악 가능하다.

단점

- 변경을 수용하기 어렵다.
- 시스템의 동작을 후반에 가야지만 확인이 가능하다.
- 대형 프로젝트에 적용하는 것이 부적합하고, 일정이 지연될 가능성이 크다.

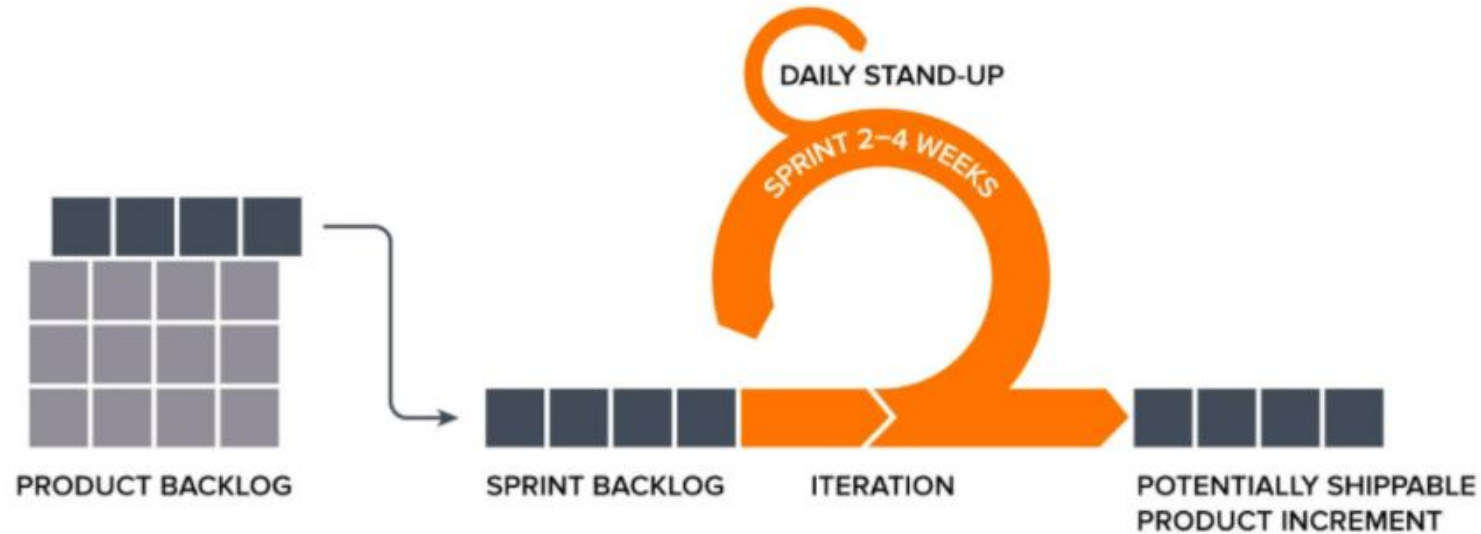
Agile

agile 미국식 ['ædʒɪl]  영국식 ['ædʒaɪl] 

1. [형용사] 날렵한, 민첩한 (=nimble)
2. [형용사] (생각이) 재빠른, 기민한

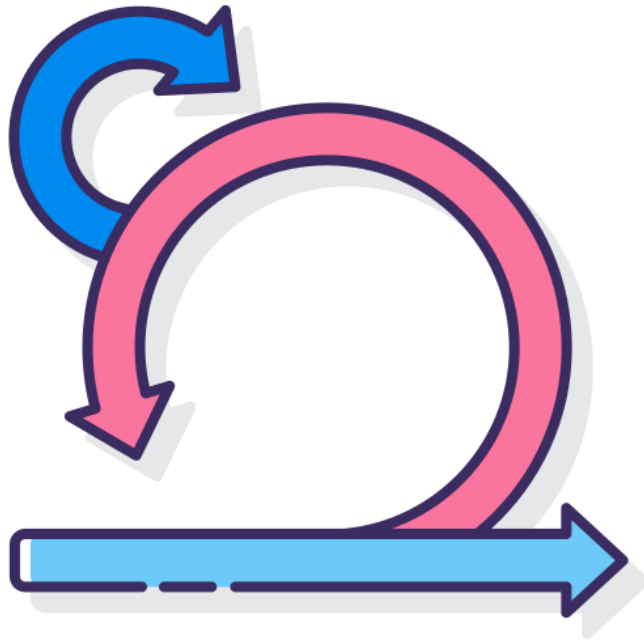
- 짧은 주기의 개발 단위를 반복해 하나의 큰 프로젝트를 완성해 나가는 것
- 🔑 협력과 피드백
- 🔑 유연한 일 진행 + 빠른 변화 대응

Agile



- 짧은 주기로 설계, 개발, 테스트, 배포 과정을 반복
- 요구 사항을 작은 단위로 쪼개 그에 대한 솔루션을 만들고, 빠르게 보여줌으로써 요구 사항에 대한 검증을 진행

Agile 방법론



Scrum(스크럼)

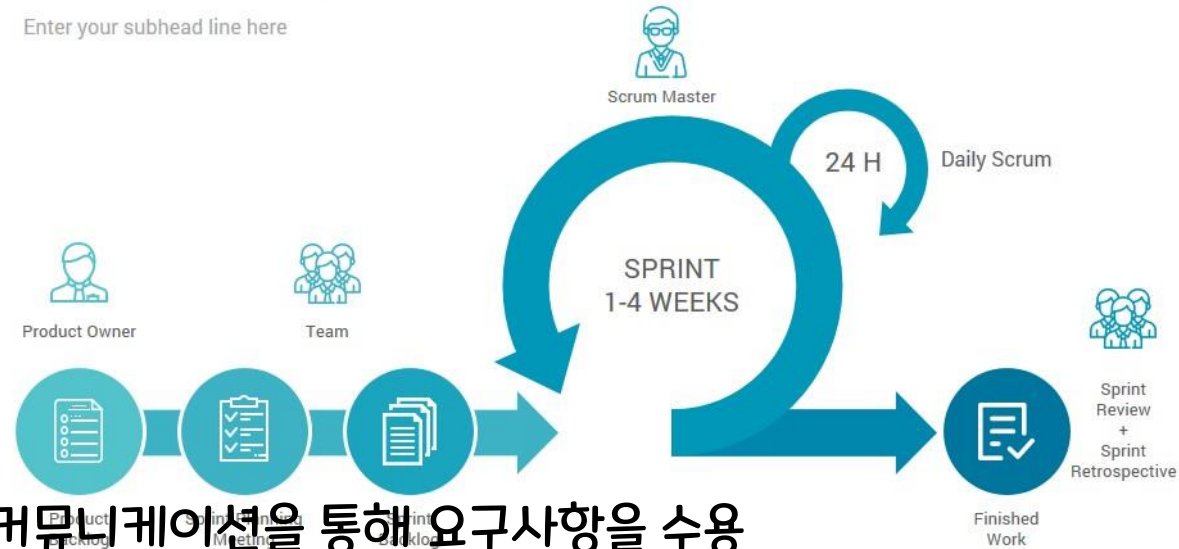


Kanban(칸반)

Scrum(스크럼)

Scrum Process

Enter your subhead line here

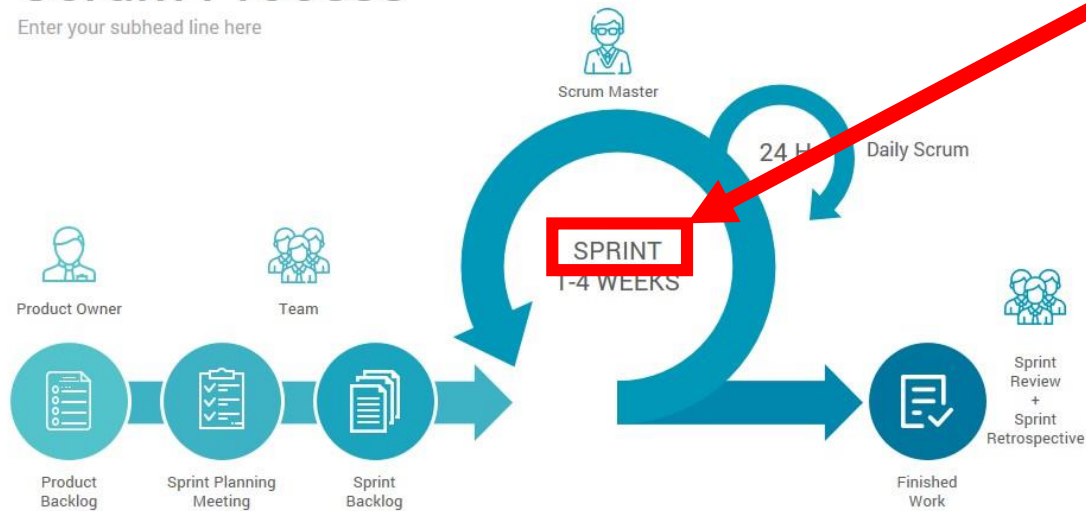


1. 개발자와 고객 사이의 지속적인 커뮤니케이션을 통해 요구사항을 수용
2. 고객이 결정한 사항을 가장 우선적으로 시행
3. 팀원들과 주기적인 미팅을 통해 프로젝트를 점검
4. 주기적으로 제품 시현을 하고 고객으로부터 피드백 수용

Scrum(스크럼)

Scrum Process

Enter your subhead line here



작은 기능에 대해

“계획, 개발, 테스트, 기능 완료”

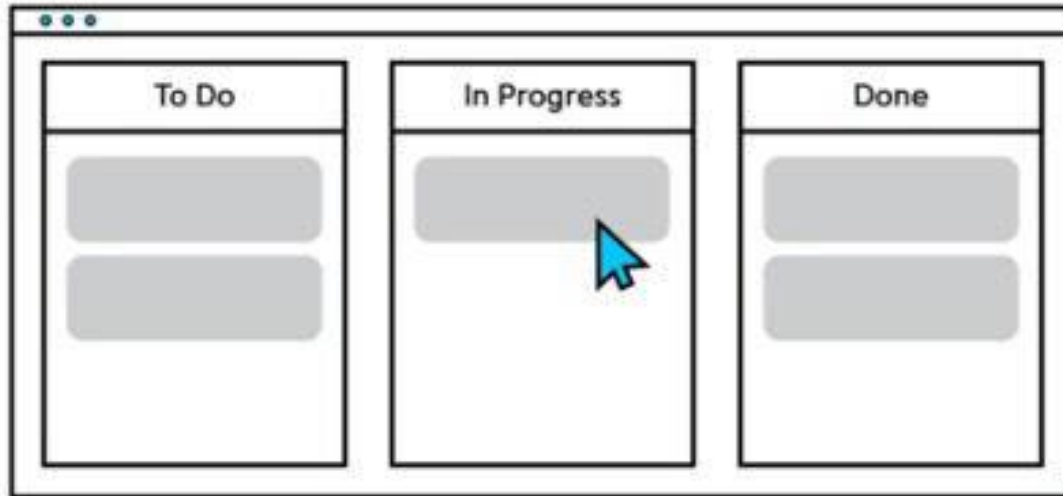
에 대해 주기적으로 시행하는 것

일반적인 스프린트 주기 : 1~2주

Kanban(칸반)

칸반(Kanban)이란?

칸반(Kanban)은 단계별 작업 현황을 열(column) 형식의 보드 형태로 시각화하는 프로젝트 관리 방법을 말합니다.



장점

- 업무 흐름의 시각화
- 진행 중 업무의 제한
- 명시적 프로세스 정책 수립
- 업무 흐름의 측정과 관리

Git (깃)

Git

- Git 이란?

소스 코드를 효율적으로 관리하기 위해 만들어진 “분산형 버전 관리 시스템”

- 사용 이유?

소스 코드의 변경 이력을 쉽게 확인

특정 시점에 저장된 버전과 비교하거나 특정 시점으로 돌아가기 위해

Branch(브랜치)

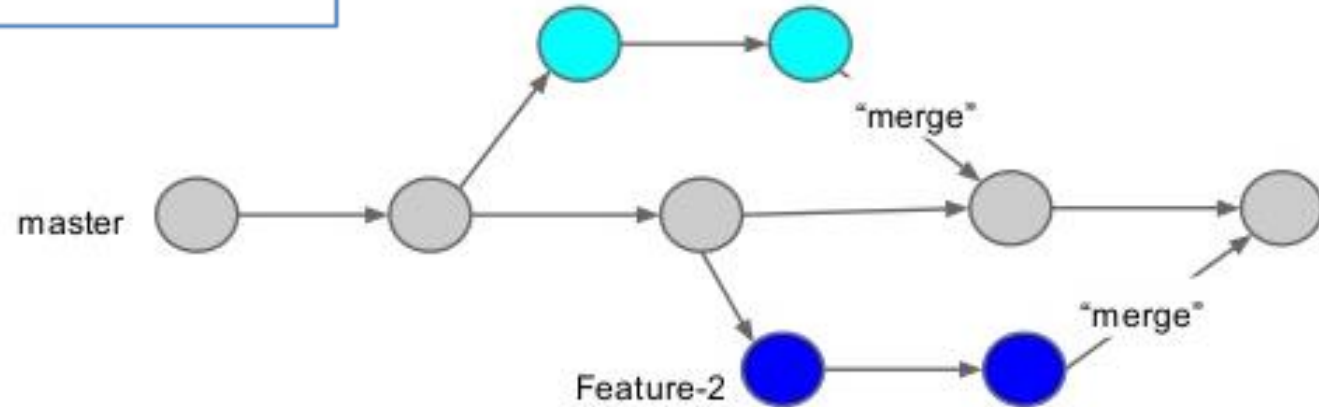
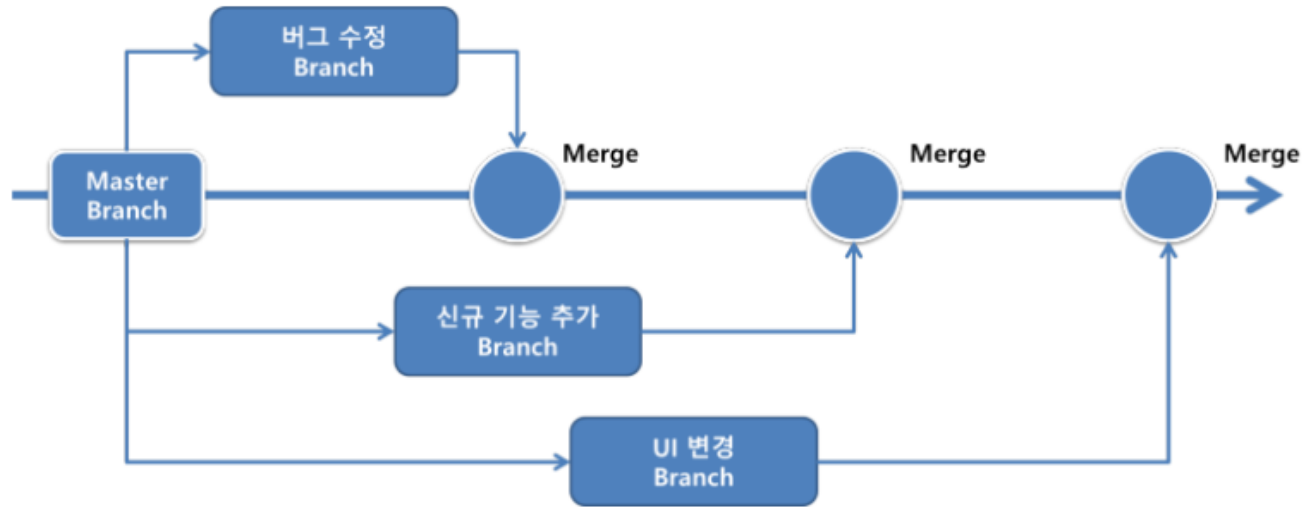


Branch

독립적으로 어떤 작업을 하기 위해 필요한 개념

Ex) A라는 사람이 “로그인” 기능을 만들고, B라는 사람이 “버그 수정” 을 할 때
A와 B는 **최초 Branch에서 파생한 각각의 Branch**를 만들어 작업을 진행하고
최초 Branch 로 Merge를 통해 각자가 작업한 것을 합칠 수 있다.

Branch



Branch 생성하기

```
git branch # local branch 목록 확인
```

```
git branch "브랜치명" # 현재 branch에서 새로운 branch 생성
```

```
git checkout "전환 브랜치명" # branch 이동
```

```
git branch -d "브랜치명" # bran 삭제
```

```
# (단, 삭제할 branch가 현재 branch에 합쳐져 있을 경우에만)
```

새로운 branch 생성 & 이동 동시에

```
git checkout -b "만들 브랜치명"
```

Merge

git branch를 다른 branch로 합치는 과정

Ex) a 브랜치에 b 브랜치를 합치고 싶은 경우

```
git checkout a
```

← a 브랜치로 이동

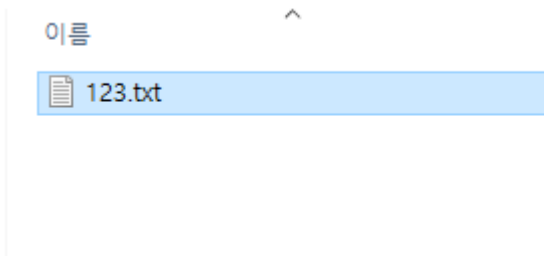
```
git merge b
```

← b 브랜치와 merge 진행

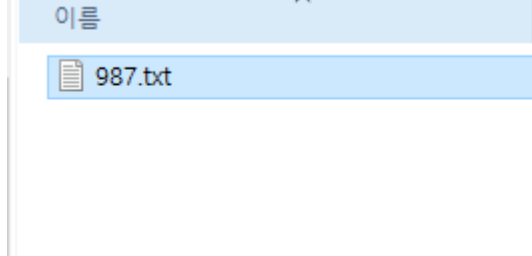
Merge

Case1. a 브랜치와 b 브랜치에서 서로 다른 파일을 수정했을 때

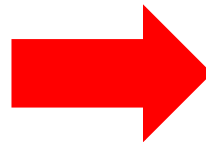
a 브랜치



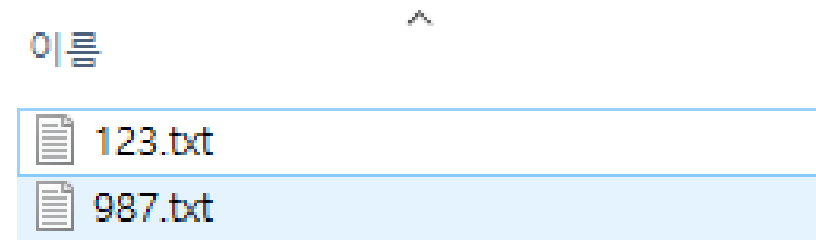
b 브랜치



```
git checkout a  
git merge b
```



a 브랜치



알아서 합쳐진다! 😊✌️

Merge

Case2. 서로 같은 파일에서 다른 부분을 수정했을 때

원본

```
<html>
  <head>
    <title>원본</title>
  </head>
  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

a 브랜치

```
<html>
  <head>
    <title>a에서 수정</title>
  </head>
  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

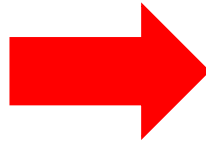
b 브랜치

```
<html>
  <head>
    <title>원본</title>
  </head>
  <body>
    <h1>b에서 수정</h1>
  </body>
</html>
```


Merge

Case2. 서로 같은 파일에서 다른 부분을 수정했을 때

```
git checkout a  
git merge b
```



a 브랜치

```
<html>  
  <head>  
    <title>a에서 수정</title>  
  </head>  
  <body>  
    <h1>b에서 수정</h1>  
  </body>  
</html>
```

알아서 합쳐진다! 😊✌️

Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

원본

```
<html>
  <head>
    <title>원본</title>
  </head>

  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

a 브랜치

```
<html>
  <head>
    <title>a에서 수정</title>
  </head>

  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

b 브랜치

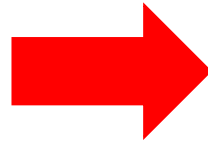
```
<html>
  <head>
    <title>b에서 수정</title>
  </head>

  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

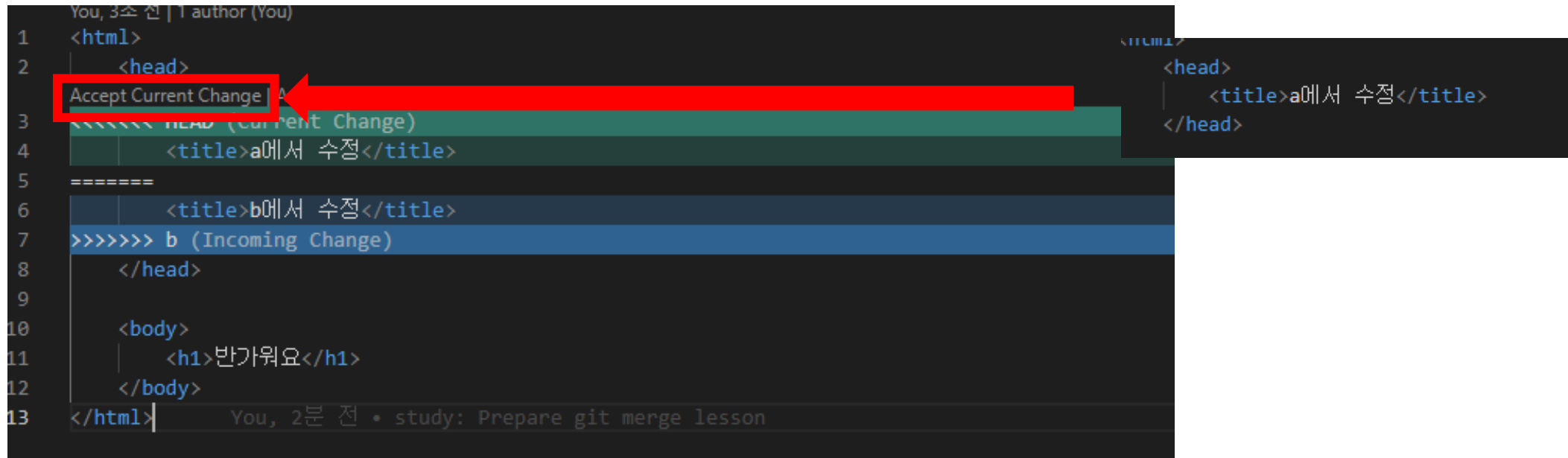
```
git checkout a  
git merge b
```



수동으로 해결해야 한다!!

Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때



```
1 <html>
2   <head>
3   <title>a에서 수정</title>
4   </head>
5   <body>
6   <h1>반가워요</h1>
7   </body>
8 </html>
```

Accept Current Change

=====
>>>>>> b (Incoming Change)
</head>

<body>
 <h1>반가워요</h1>
</body>
</html>

You, 3초 전 | 1 author (You)

You, 2분 전 • study: Prepare git merge lesson

Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```
You, 3초 전 | 1 author (You)
1 <html>
2 |   <head>
3 |   Accept Current Change Accept Incoming Change Accept Outgoing Change
4 <<<<<< HEAD (Current Change)
5 |   <title>a에서 수정</title>
6 =====
7 |   <title>b에서 수정</title>
8 >>>>>> b (Incoming Change)
9 |   </head>
10 |
11 |   <body>
12 |   |   <h1>반가워요</h1>
13 |   </body>
14 </html>
You, 2분 전 • study: Prepare git merge lesson
```

```
<head>
|   <title>b에서 수정</title>
</head>
```

Merge

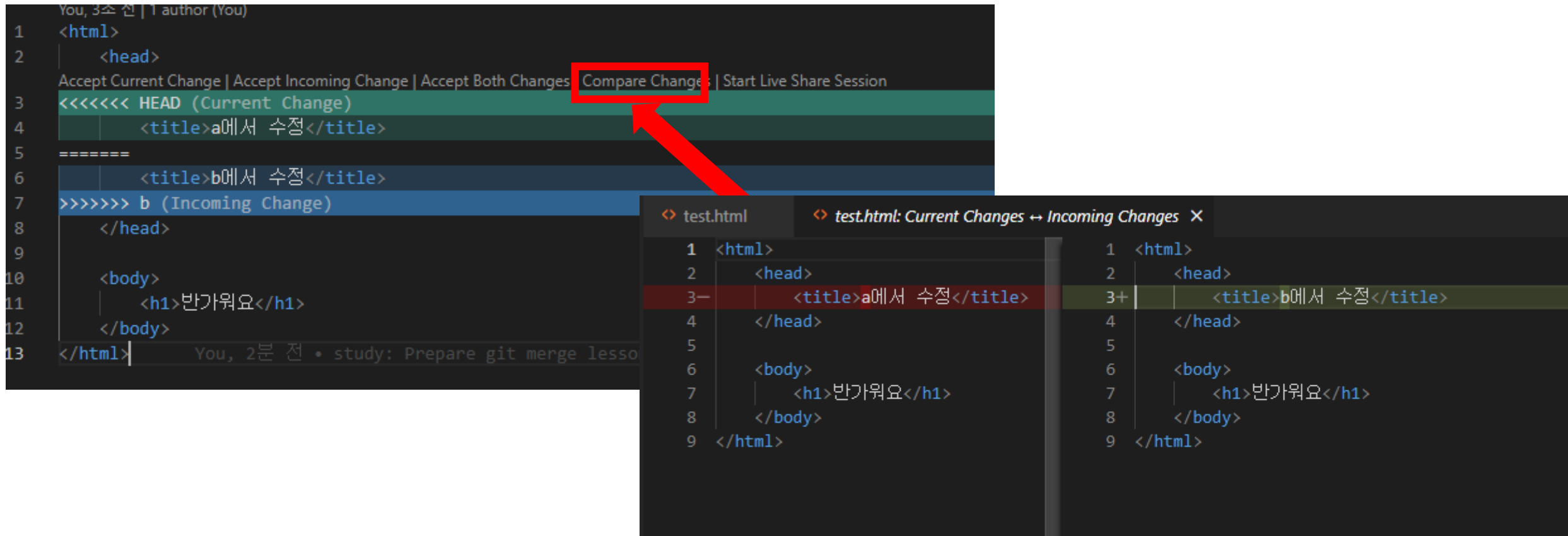
Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```
You, 3초 전 | 1 author (You)
1  <html>
2  |   <head>
3  |   Accept Current Change | Accept Incoming Change | Accept Both Changes | C
4  <<<<<< HEAD (Current Change)
5  |   <title>a에서 수정</title>
6  =====
7  |   <title>b에서 수정</title>
8  >>>>>> b (Incoming Change)
9  |   </head>
10 |
11 |   <body>
12 |   |   <h1>반가워요</h1>
13 |   </body>
14 </html> You, 2분 전 • study: Prepare git merge lesson
```

```
<head>
|   <title>a에서 수정</title>
|   <title>b에서 수정</title>
</head>
```

Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때



```
1 <html>
2   <head>
3   <<<<<< HEAD (Current Change)
4   <title>a에서 수정</title>
5   =====
6   <title>b에서 수정</title>
7   >>>>>> b (Incoming Change)
8   </head>
9
10  <body>
11    <h1>반가워요</h1>
12  </body>
13 </html>
```

test.html

test.html: Current Changes ↔ Incoming Changes X

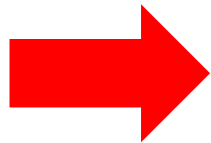
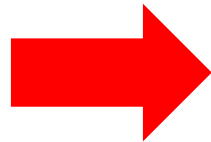
```
1 <html>
2   <head>
3-  <title>a에서 수정</title>
4   </head>
5
6   <body>
7     <h1>반가워요</h1>
8   </body>
9 </html>
```

```
1 <html>
2   <head>
3+  <title>b에서 수정</title>
4   </head>
5
6   <body>
7     <h1>반가워요</h1>
8   </body>
9 </html>
```

Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```
git checkout a  
git merge b
```



해결 후 파일 저장 -> add -> commit -> push

수동으로 코드를 머지하면 된다! 😊✌️

실습 22. Merge(1)

```
<html>  
  <head>  
    <title>test</title>  
  </head>  
  <body>  
    <h1>반가워요</h1>  
    <h5>안녕하세요</h5>|  
  </body>  
</html>
```

1. master(main) 에서 test.html 파일을 위와 같이 만든 후 push
2. “test” 라는 이름의 branch 만들기
3. master(main)에서 test.html의 h5 부분 수정하고 commit
4. test 브랜치로 옮겨가기
5. test 브랜치에서 test.html 내의 title과 h5 내용 수정 후 push
6. master(main) 브랜치에 test 브랜치 merge 하기
7. 충돌 해결 후 push

실습 22. Merge(2)

1. master(main) 에서 test.html 파일을 위와 같이 만든 후 push
2. “test” 라는 이름의 branch 만들기
3. master(main)에서 test.html의 h5 부분 수정하고 commit
4. test 브랜치로 옮겨가기
5. test 브랜치에서 test.html 내의 title과 h5 내용 수정 후 push
6. master(main) 브랜치에 test 브랜치 merge 하기
7. 충돌 해결 후 push

```
B I S | 🔗 | ☰ ☷ | ⌵ | </> 📄  
  
1.  
git add .  
git commit -m "study: Create test.html file"  
git push origin master  
2. git checkout -b test  
3.  
.....  
+ | 📄 🗣️ | 😊 @ Aa
```

위와 같이 git 명령어와

7번까지 해결한 이후 github 홈페이지에서 test.html 내용 캡처해서 제출

대형 프로젝트를 위해 알아 두면 좋은 지식!

Branch의 종류

main(master)

develop

feature

release

hotfix

Branch - main(master)

- 제품으로 출시될 수 있는 브랜치
- 배포(Release) 이력을 관리하기 위해 사용
- 배포 가능한 상태만을 관리하는 브랜치

Branch - develop

- 다음 출시 버전을 개발하는 브랜치
- 기능 개발을 위한 브랜치들을 병합하기 위해 사용
- 평소 개발을 진행하는 브랜치

Branch - feature

- 기능 개발을 진행하는 브랜치
- 새로운 기능 개발 및 버그 수정을 할 때마다 'develop' 에서 분기
- 공유할 필요가 없어 로컬에서 진행 후 develop 에 merge 해 공유
- 이름 : feature/~~

```
git checkout -b feature/이름 develop
```

```
/* 개발~~~ */
```

```
git checkout develop  
git merge feature/이름  
git branch -d feature/이름  
git push origin develop
```

Branch – release

- 출시 버전을 준비하는 브랜치
- 배포를 위한 전용 브랜치
- 이름 : release-0.0

```
git checkout -b release-1.2 develop
```

```
/* 배포 사이클 */
```

```
git checkout master
```

```
git merge release-1.2
```

```
git tag -1 1.2
```

```
git checkout develop
```

```
git merge release-1.2
```

```
git branch -d release-1.2
```


Branch – hotfix

- 출시 버전에서 발생한 버그 수정 브랜치
- 배포한 버전에 긴급하게 수정해야 할 필요가 있는 경우 사용
- Master에서 분기
- 이름 : hotfix-0.0.0

```
git checkout -b hotfix-1.2.1 master
```

```
/* 문제 수정 */
```

```
git checkout master
```

```
git merge hotfix-1.2.1
```

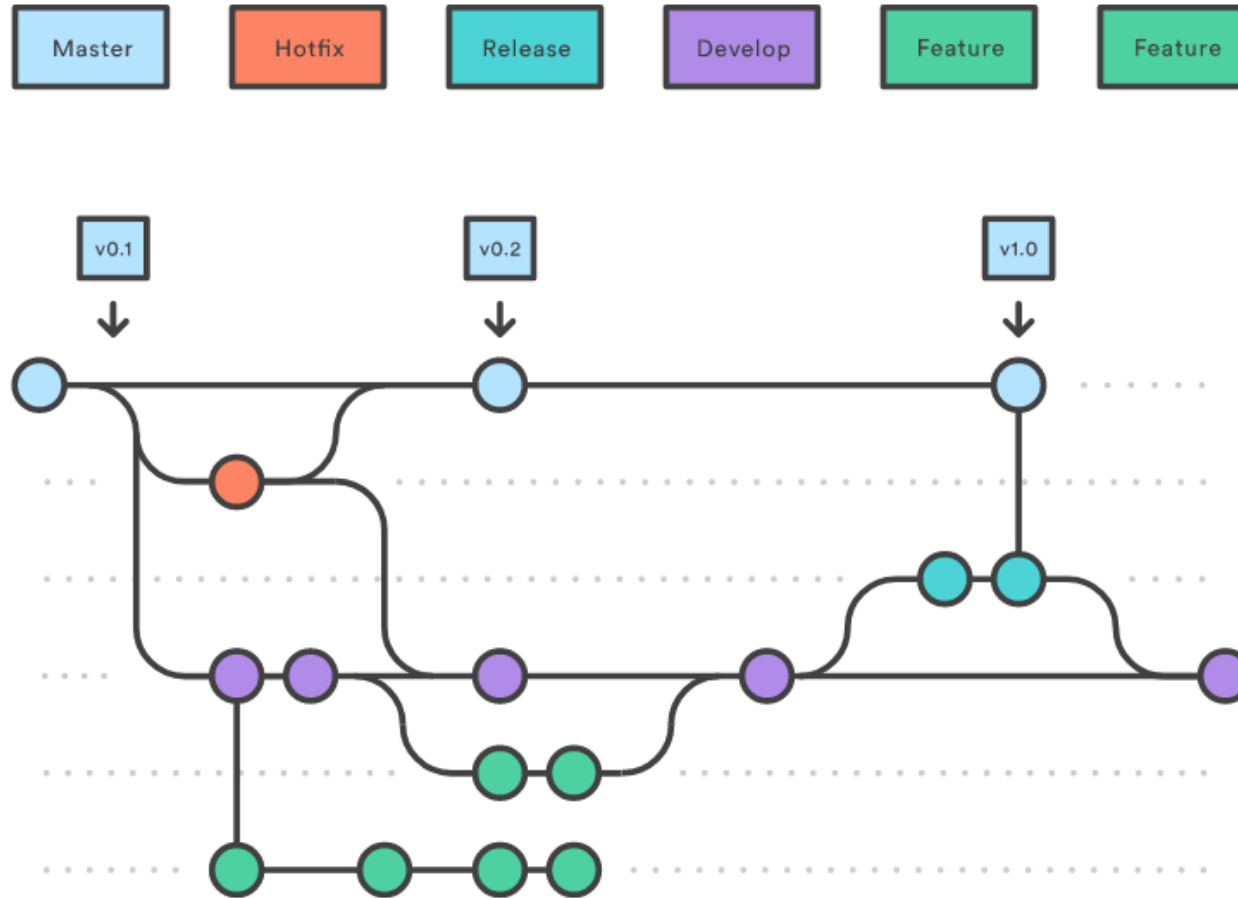
```
git tag -a 1.2.1
```

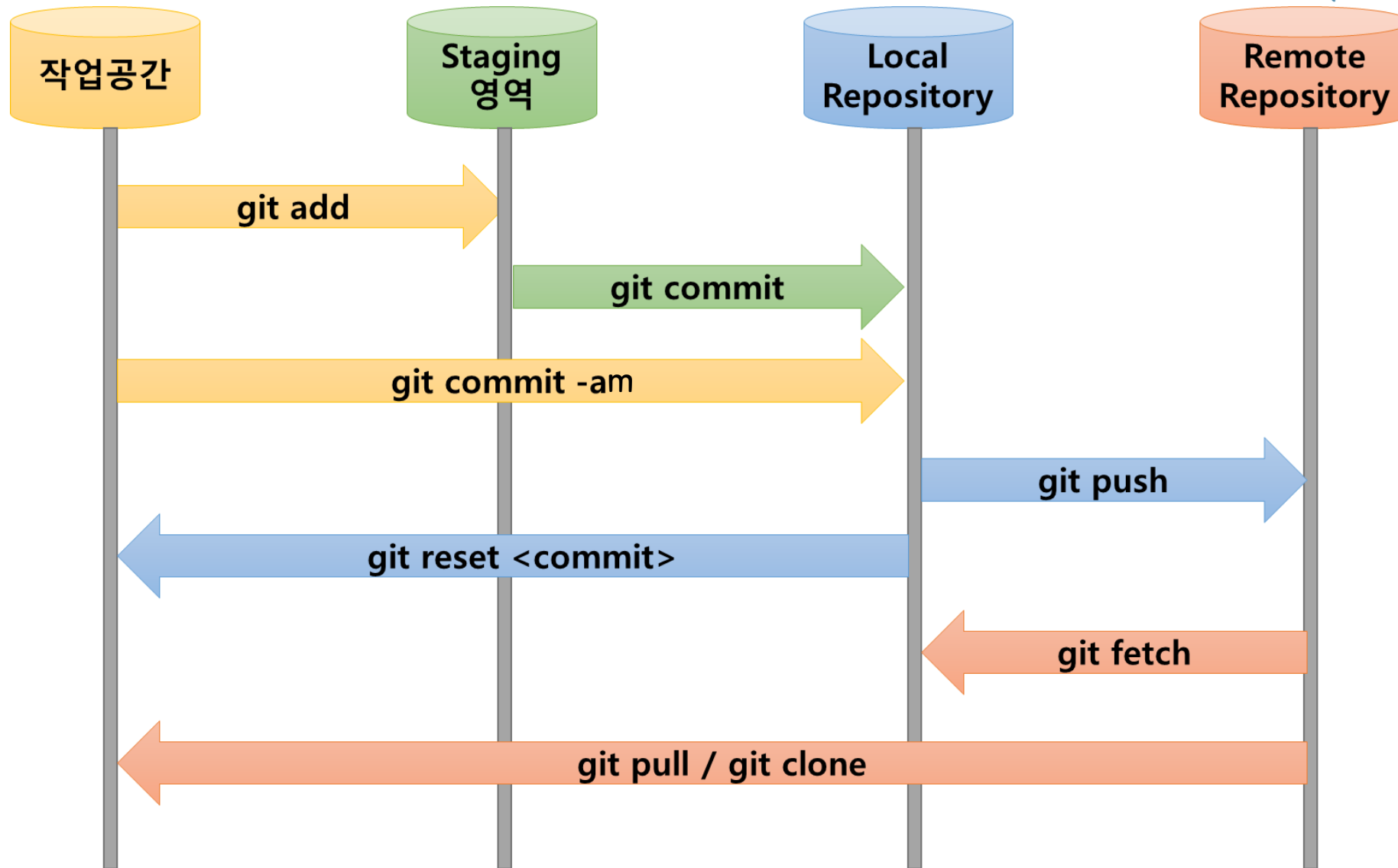
```
git checkout develop
```

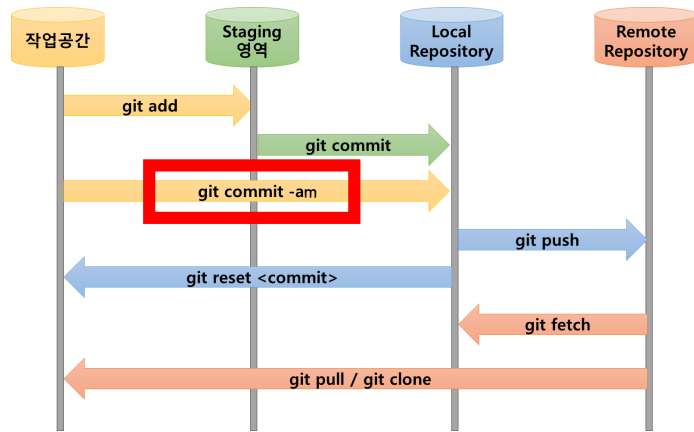
```
git merge hotfix-1.2.1
```

```
git branch -d hotfix-1.2.1
```

Branch 종류

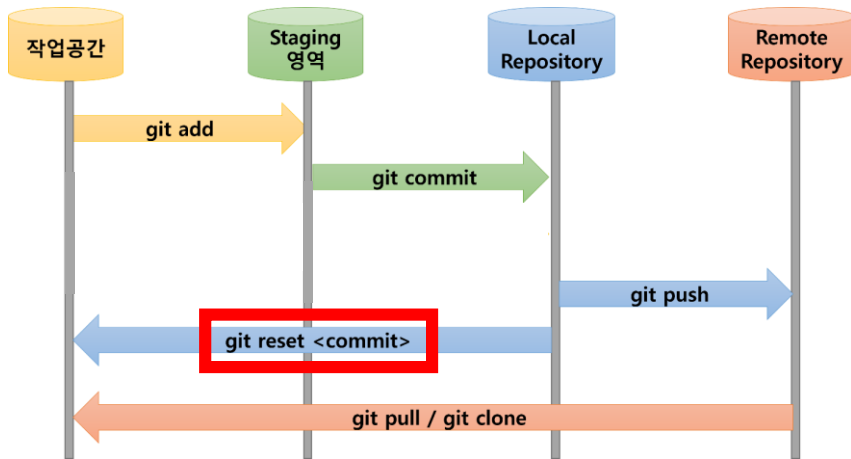






```
git add .  
git commit -m "커밋메세지"
```

```
git commit -am "커밋메세지"
```



가장 최근 커밋 취소

```
git reset HEAD^
```

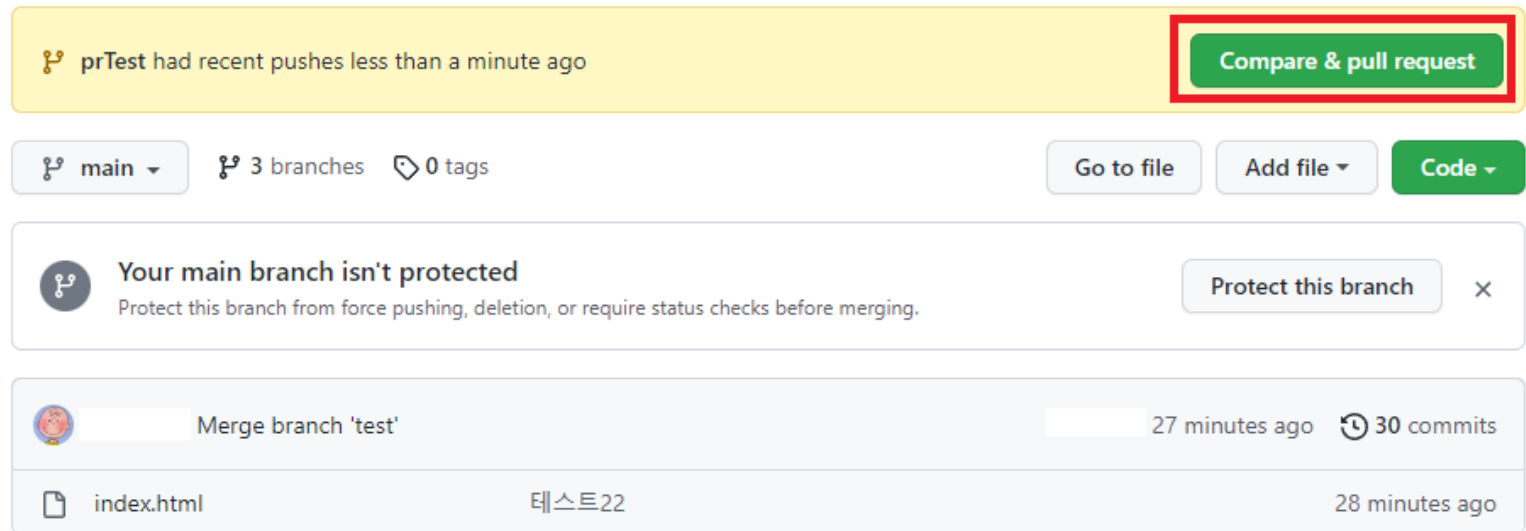
특정 커밋 취소

```
Linda@Linda MINGW64 /e/Development/github/SeSAC4_
$ git log
commit df72409a0cd5864061c73a28a3e4896d08f6e2cb (
  (HEAD)
```

```
git reset --hard df72409a0cd5864061c73a28a3e4896d08f6e2cb
```

Pull Request


- Push 권한이 없는 오픈 소스 프로젝트에 기여할 때 많이 사용함.
- “ 내가 수정한 코드가 있으니 내 branch를 가져가 검토 후 병합(merge) 해주세요!! ”
- 당황스러운 코드 충돌을 줄일 수 있음



Pull Request

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: main ← compare: prTest ✓ **Able to merge.** These branches can be automatically merged.

github pr 테스트

Write

Preview

H B I ≡ <> 🔗 ≡ ≡ ☑ @ ↗ ↶

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Merge pull request

You can also [open this pull request](#)

✓ Create a merge commit

All commits from this branch will be added to the base branch via a merge commit.

Squash and merge

The 1 commit from this branch will be added to the base branch.

Rebase and merge

The 1 commit from this branch will be rebased and added to the base branch.

Pull Request

test3에서 코드 수정 #1

Merged

merged 1 commit into `main` from `test3` 2 minutes ago

merge 완료!!

Conversation 0

Commits 1

Checks 0

Files changed 1

commented 4 minutes ago

Owner

코드 수정했으니까 검토 후 머지해주세요!!

test3에서 코드 수정 57eae68



merged commit into `main` 2 minutes ago

Revert



Pull request successfully merged and closed

You're all set—the `test3` branch can be safely deleted.

Delete branch

.gitignore

.gitignore?

- Git 버전 관리에서 **제외할 파일 목록을 지정**하는 파일
- Git 관리에서 특정 파일을 제외하기 위해서는 git에 올리기 전에 .gitignore에 파일 목록을 미리 추가해야 한다.

.gitignore

***.txt** → 확장자가 txt로 끝나는 파일 모두 무시

!test.txt → test.txt는 무시되지 않음.

test/ → test 폴더 내부의 모든 파일을 무시 (b.exe와 a.exe 모두 무시)

/test → (현재 폴더) 내에 존재하는 폴더 내부의 모든 파일 무시 (b.exe무시)

```
(base) [07:25 PM] cwjcsk:~/99_test/99_tmp$ tree -a
.
├── .gitignore
├── test
│   └── b.exe
└── tmp
    ├── test
    │   └── a.exe
```

3 directories, 3 files