

개발 문화와 Git

즐거로운 팀플을 해보자!

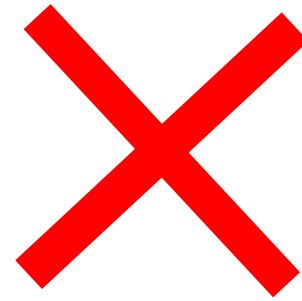
목차

1. 혼한개발문화와 생태계
2. 좋은 개발문화를 위한 새로운 개념
3. Agile(애자일)
4. Waterfall 이란
5. Agile 개념
6. Agile 방법론
7. Scrum(스크럼)
8. Kanban(칸반)
9. 깃 (git)
10. 버전관리 시스템
11. git vs gitHub
12. git 커밋이해하기(로컬에서)
13. Git 커밋 이해하기(로컬과 원격)
14. git 커밋 workflow 살펴보기
15. 커밋을 위한 간단한 git 명령어
16. 커밋취소
17. 로컬저장소와 원격저장소 연결해제
18. Branch란
19. Branch 작업 흐름도
20. Branch 생성하기
21. 새로운 branch 생성과 이동을 동시에
22. Merge란
23. Merge의 사례들
24. Merge 실습과제
25. Branch의 종류 알아보기
26. Pull Request
27. .gitignore
28. Netlify

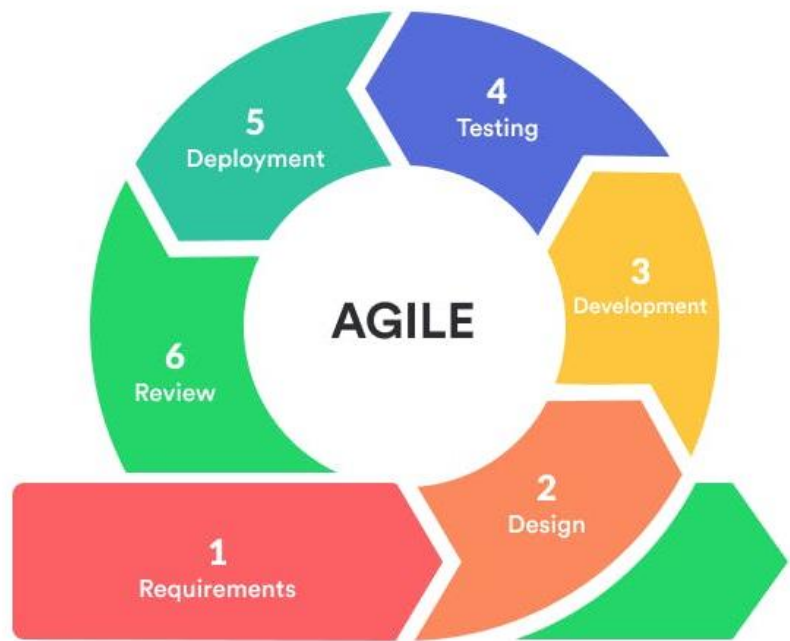
1. 흔한 개발문화 생태계



좋은 개발문화?



2. 좋은 개발 문화를 위한 새로운 개념



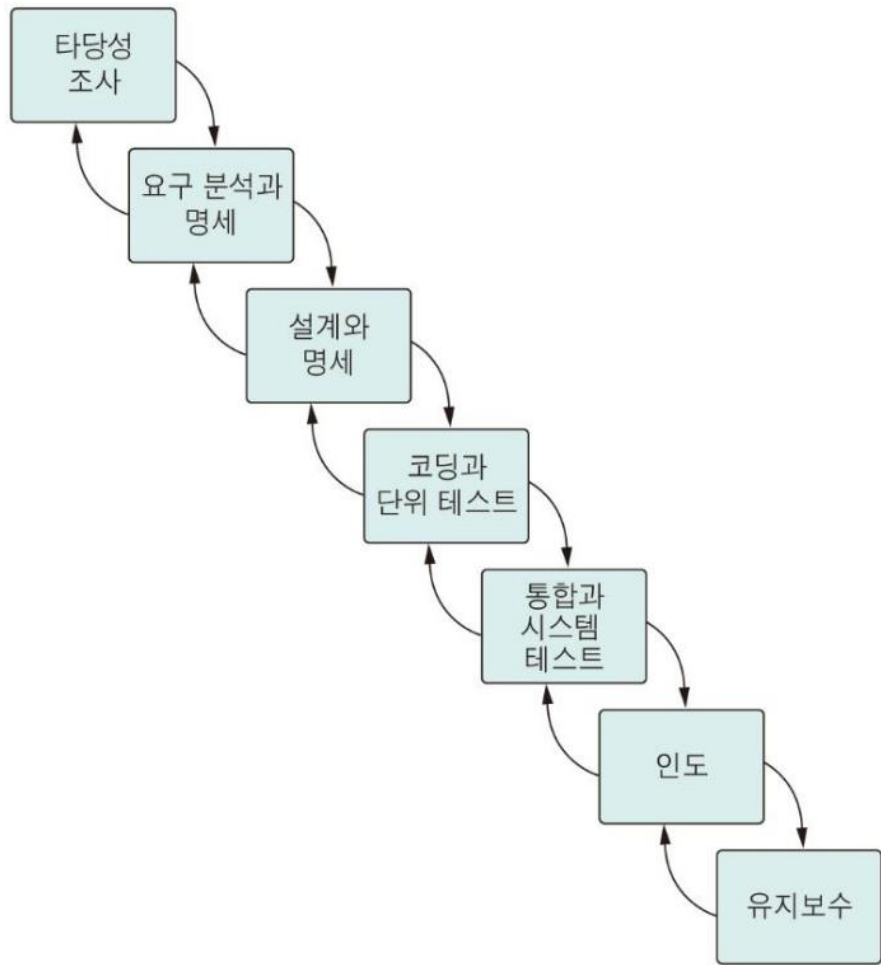
3. Agile (애자일)

agile 미국식 ['ædʒɪl]  영국식 ['ædʒaɪl] 

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

애자일은 신속한 반복 작업을 통해 실제 작동 가능한 소프트웨어를 개발하여 지속적으로 제공하기 위한 소프트웨어 개발 방식이고
협업과 워크플로우를 바라보는 하나의 관점이며, 우리가 무엇을 어떻게 만들지에 관한 선택을 안내하는 가치 체계이다
우리가 이제까지 사용했던 방식을 waterfall (폭포수) 방식이라고 할 수 있다

4. Waterfall



Waterfall Model (폭포수 모델)

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

장점

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

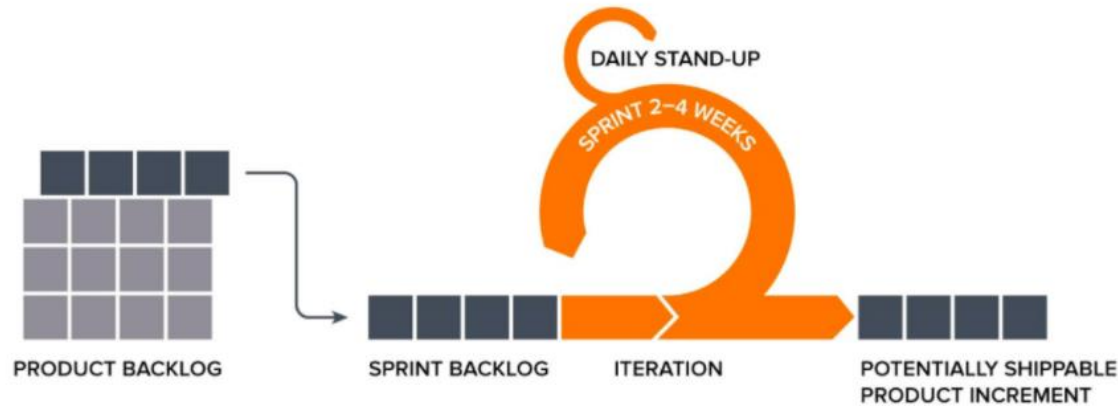
단점

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

5. Agile 개념

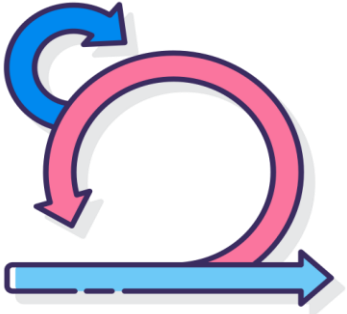
개념을 설명하는 문구들 소개하면

애자일 기반의 조직 개발을 통해 새로운 일하는 방식과 더 좋은 조직 문화를 함께 고민



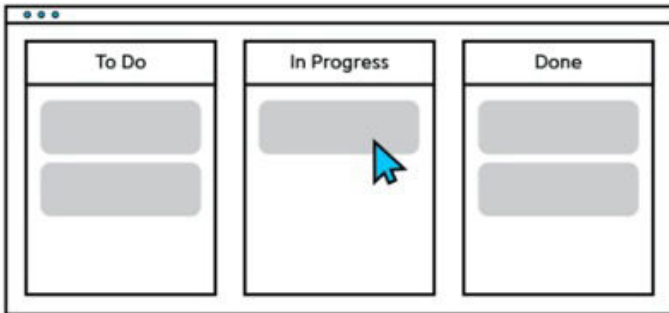
- 짧은 주기로 설계, 개발, 테스트, 배포 과정을 반복
- 요구 사항을 작은 단위로 쪼개 그에 대한 솔루션을 만들고, 빠르게 보여줌
으로써 요구 사항에 대한 검증을 진행

6. Agile 방법론



Scrum(스크럼)

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



Kanban(칸반)

단계별 작업 현황을 열(column)형식의 보드 형태로 시각화하는 프로젝트 관리 방법을 말한다.

장점

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

7. Scrum(스크럼)

- Scrum is an agile team collaboration framework commonly used in software development and other industries.
- 프로젝트 관리를 위한 상호, 점진적 개발방법론이며, 애자일 소프트웨어 개발 중의 하나이다. 스크럼은 소프트웨어 개발 프로젝트를 위해 고안되었지만 소프트웨어 유지보수 팀이나 일반적인 프로젝트 / 프로그램 관리에도 적용될 수 있다 (위키백과)
- dics.naver.com 사전으로 검색해보면 '상호 점진력 개발방법론' 으로 소개하고 있다
- ClickUp를 소개하는 글에는 '자신을 Scrum Project Management' 소개하고 있다
- 이런 개념을 이용하여 저희 회사에서도 ClickUp 이라는 프로그램을 회사에서 활용하고 있다

8. Kanban (칸반)

- Kanban 방법론은 지속적인 개선, 작업 관리의 유연성 및 향상된 워크플로우를 목표로 하는 민첩한 방법입니다
- 구글에서 'What is Agile Kanban ' 검색:
 - Kanban is a popular framework used to implement agile and DevOps software development

예시 : ClickUp

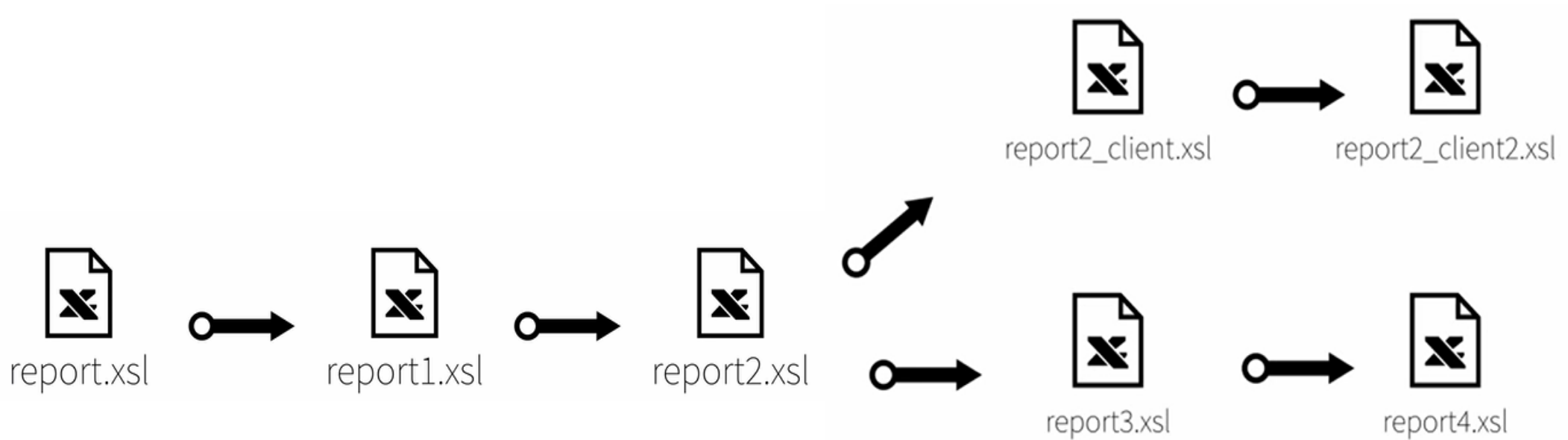
Kanban: Definition

The screenshot displays the ClickUp Kanban board interface. At the top, there's a navigation bar with tabs: Roadmap & Backlog, Quarterly Roadmap, Board, Product Request Form, and My Product Requests. Below this is a search bar and filters. The main area is divided into columns representing different quarters: FY23 Q1, FY23 Q2, FY23 Q3, FY23 Q4, and an Empty column. Each column contains task cards with details like title, dates, and status. For example, in FY23 Q1, tasks include 'Determine Helpdesk Option Availability', 'Modernize Website Menu Look & Feel', and 'Ship 12 UX improvements'. The interface also includes a bottom bar with a '+ Task' button and a 'Task' dropdown menu.

The screenshot displays the ClickUp Kanban board interface with two columns: APPROVAL and READY FOR SPRINT. Each column contains task cards with details like title, project ID, initiative, item type, requestor, and T-shirt size. For example, in the APPROVAL column, tasks include 'Deacom Finding: Dry/Trim/Packaging' and 'Determine Helpdesk Options for California'. The interface also includes a bottom bar with a '+ Task' button and a 'Task' dropdown menu.

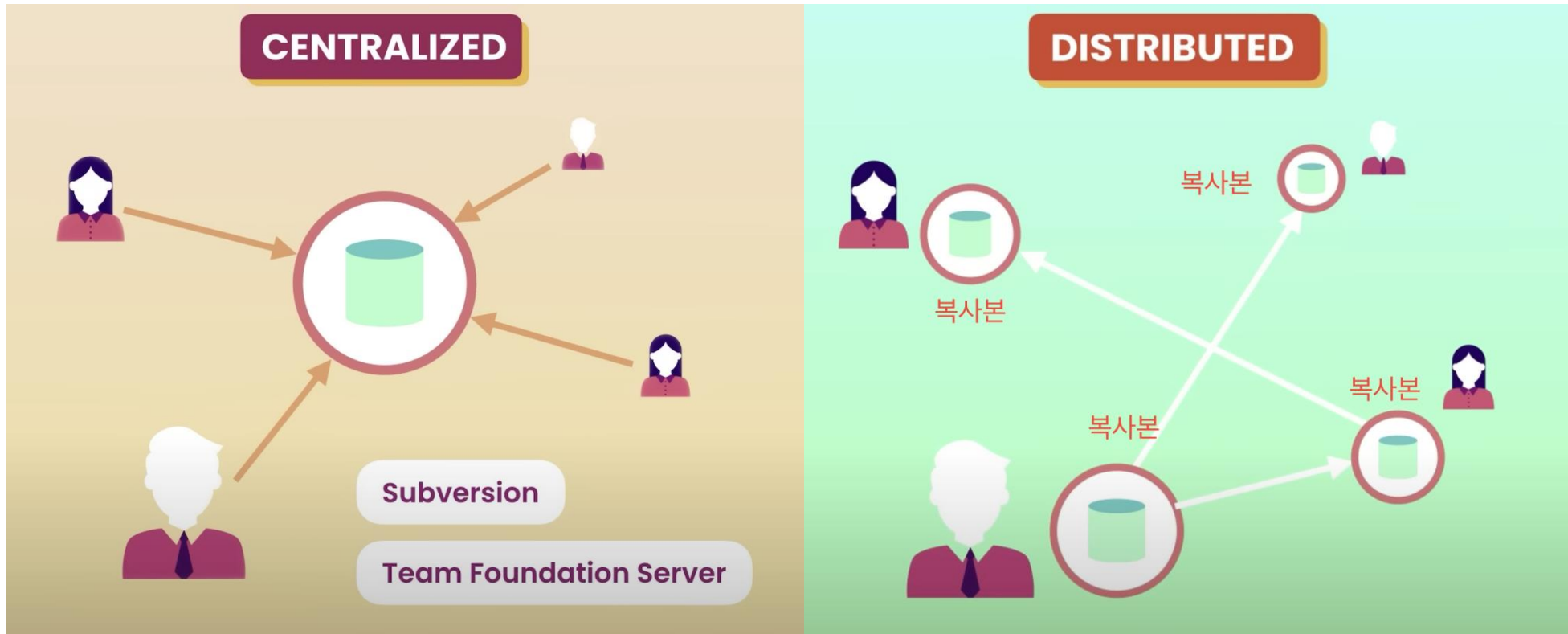
9. Git (깃)

이제까지 우리는 문서의 관리는 이렇게 했다



10. 버전관리 시스템 (VCS, Version Control System)

문서나 코드의 변경사항을 기록하여 과거의 상태를 열람하거나 과거의 시점으로 **복원**할 수 있도록 해주는 것으로 협업시 같은 파일을 변경하거나, 같은 파일이더라도 다른 부분을 수정했다면 코드를 합쳐주고 같은 부분을 수정했다면 나중에 수정한 사람에게 충돌여부를 알려주는 등 예상치 못한 상황을 방지하여 코드와 협업자들 사이에 질서를 부여해 준다.



11. Git vs GitHub

- Git 이란?

소스 코드를 효율적으로 관리하기 위해 만들어진 “분산형 버전 관리 시스템” 으로 마치 컴퓨터에 설치된 운영체제(윈도우)와 같다.

- 사용 이유?

소스 코드의 변경 이력을 쉽게 확인

특정 시점에 저장된 버전과 비교하거나 특정 시점으로 돌아가기 위해



mac 사용자와 windows 사용자의 차이극복

\r (Carriage Return) \n (Line Feed)

윈도우 사용자 : `git config --global core.autocrlf true`

맥 사용자 : `git config --global core.autocrlf input`

12. Git 커밋 이해하기(로컬에서)

로컬저장소

working tree

현재우리가 작업하는
디렉토리

.git directory

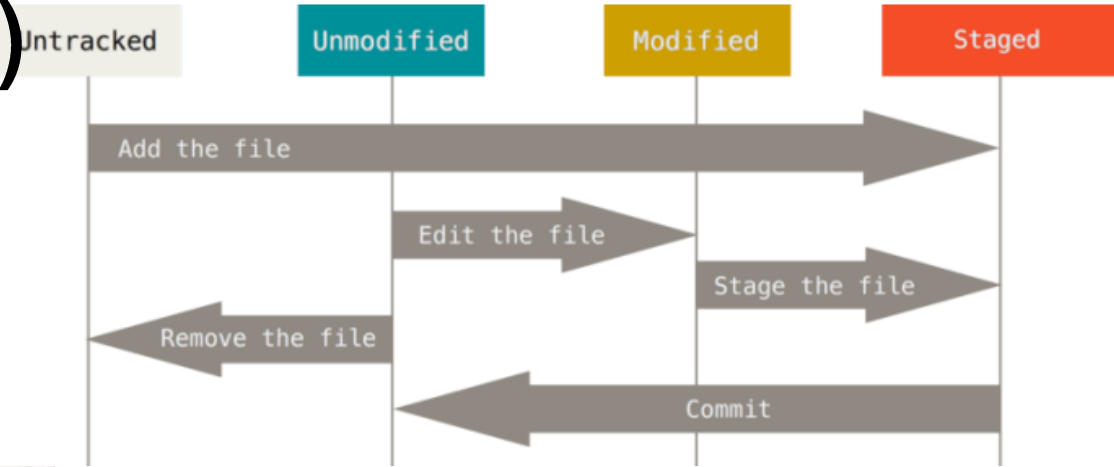
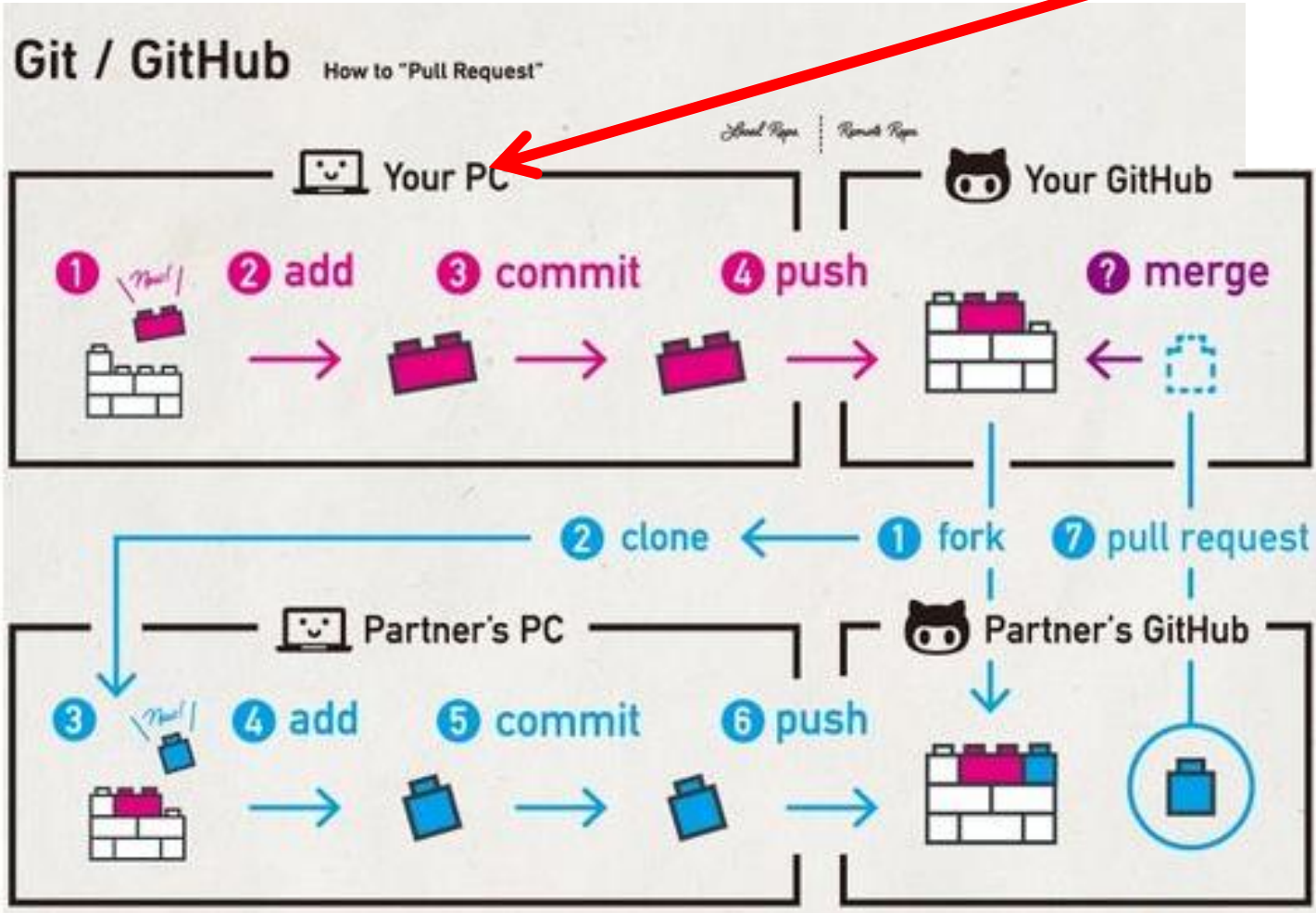
Git 프로그램 설치후에 폴더안
에서 보여지는
.git (숨김속성) 디렉토리

원격저장소

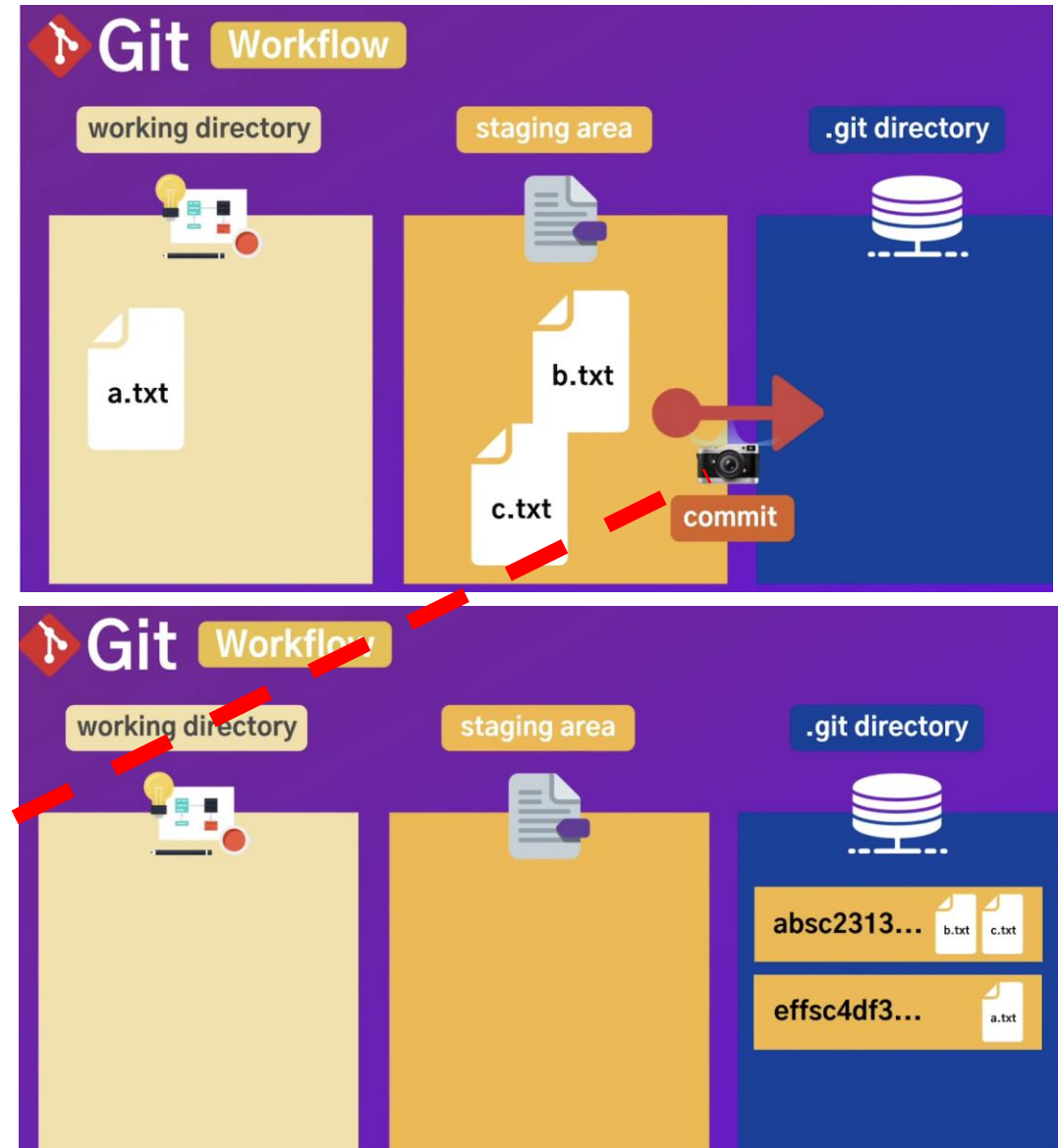
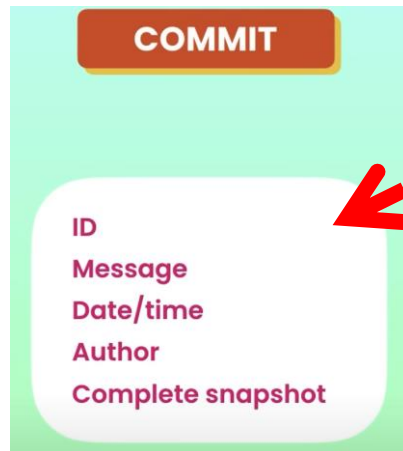
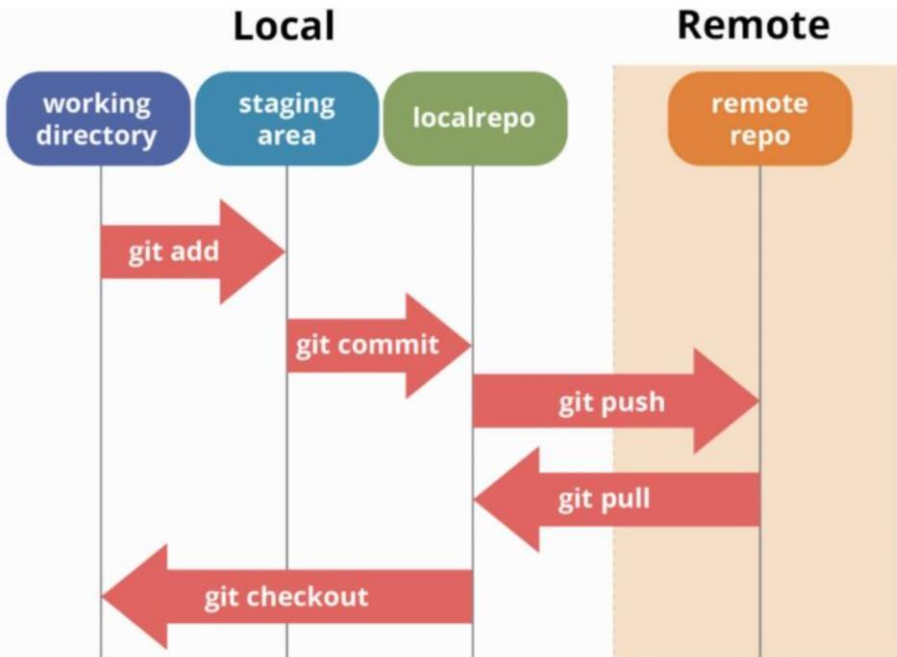
커밋

하나의 프로젝트를
작은 단위인
commit (커밋) 단
위로 나눈다

13. Git 커밋 이해하기(로컬과 원격)



14. Git 커밋의 workflow 살펴보기



15. 커밋을 위한 간단한 git 명령어

파일을 원격저장소에 올리는 것과 같다.

```
git add -all
```

```
git add .
```

위의 내용은 status 에 나온 변경사항을 모두 스테이지에 올려준다

파일을 올릴 때처럼 git add . 으로 변경내용을 모두 스테이지에 올리고

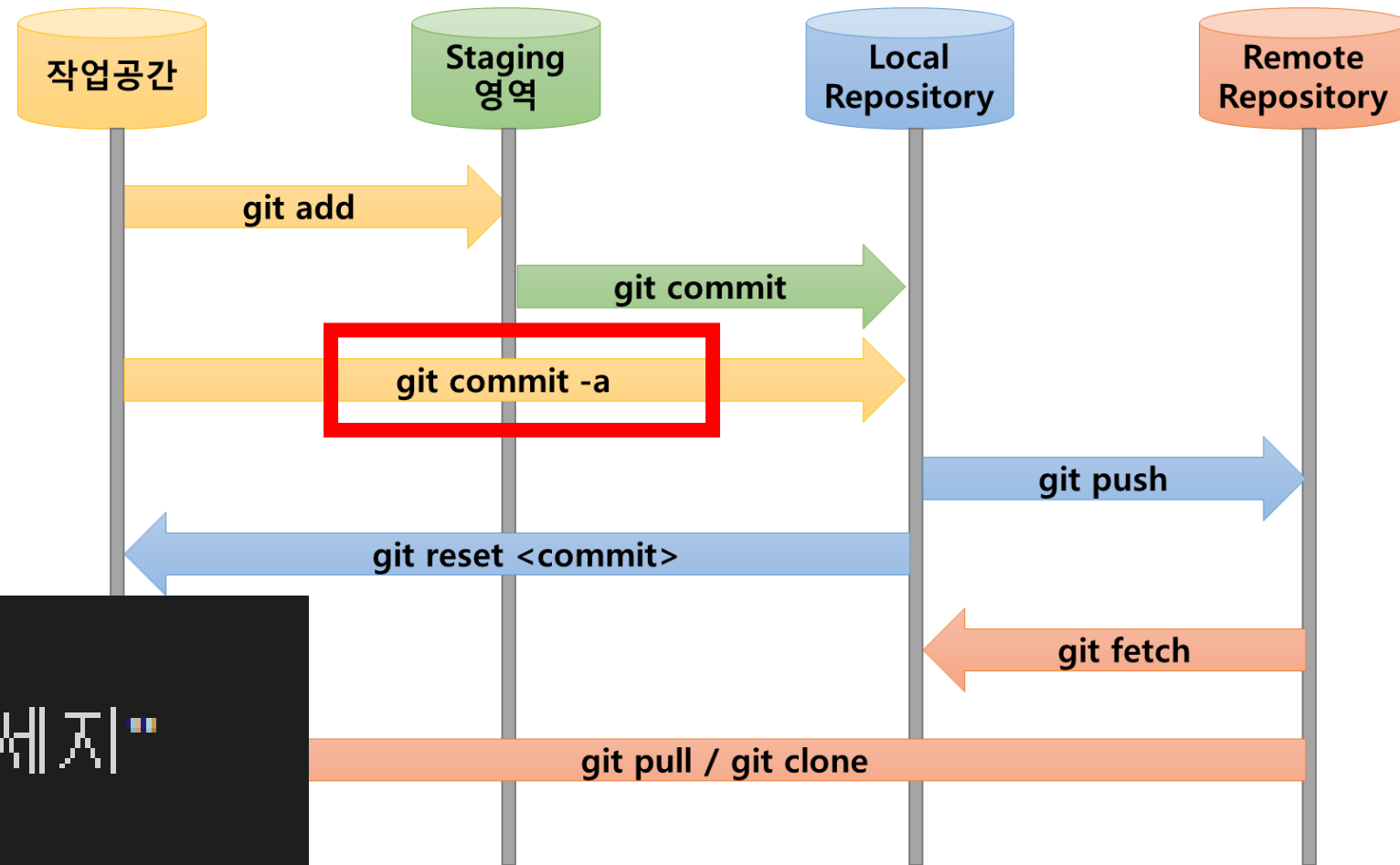
```
git commit -m "메시지"
```

```
git push origin main 또는 git push
```

- git add -u 하나 이전의 스테이지와 비교해서 변경된 부분만 add 하고 새로 만들어진 파일은 add하지 않음
- ✓ git remote -v 이전에 원격저장소를 연결한 경우가 있다면 원격저장소가 있는지 확인한다

git commit -a 커밋메시지
git commit -am 커밋메시지

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



16. 커밋 취소

- 가장 최근 커밋 취소 : `git reset HEAD^`

- 특정 커밋 취소

- git log (취소할 특정 커밋의 번호를 알기 위해)
- git reset --hard 커밋번호

```
$ git log
commit df72409a0cd5864061c73a28a3e4896d08f6e2cb (HEAD)
```

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

--hard 옵션 : 가장 최근 커밋을 되돌리고 unstaged 상태의 변경사항을 모두 제거한다

18. Branch(브랜치)란



독립적으로 어떤 작업을 하기 위해 필요한 개념

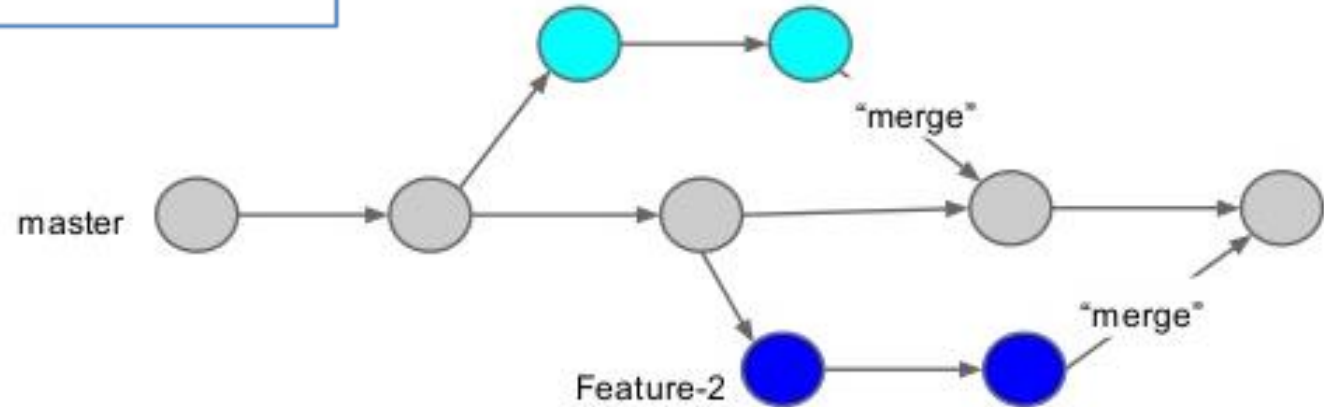
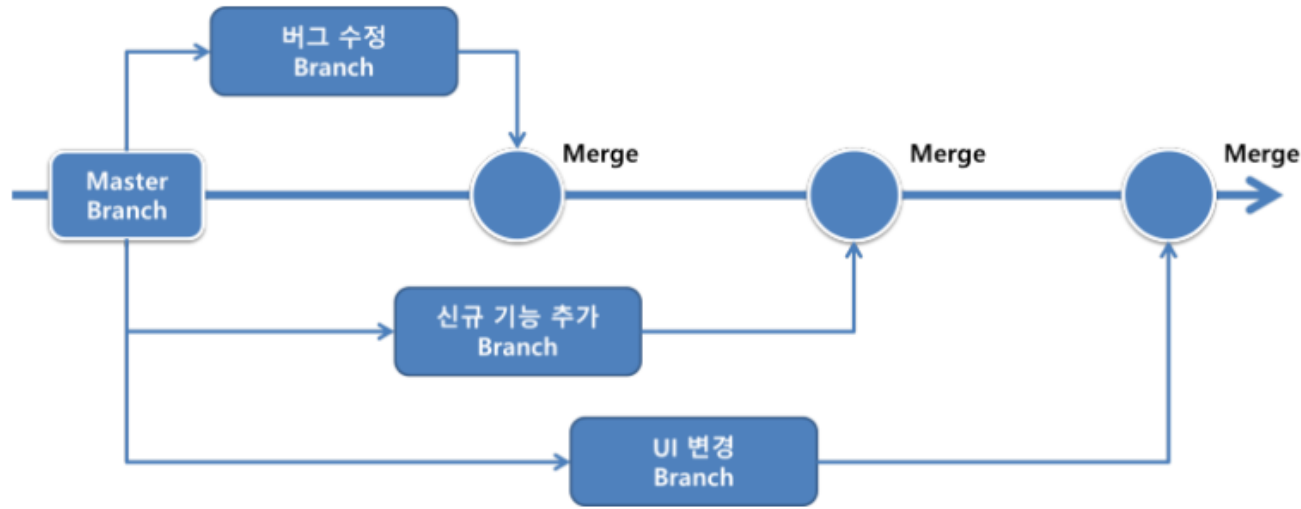
Ex) A라는 사람이 "로그인" 기능을 만들고, B라는 사람이 "버그 수정" 을 할 때
A와 B는 **최초 Branch에서 파생한 각각의 Branch**를 만들어 작업을 진행하고
최초 Branch 로 Merge를 통해 각자가 작업한 것을 합칠 수 있다.

Branch

독립적으로 어떤 작업을 하기 위해 필요한 개념이다.

- 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.
- 매 작업시에 기존의 코드를 망치는 것을 방지하기 위해 되돌아갈 수 있는 시점을 만들면서 branch라는 이름의 작업공간을 따로 만들어서 새로 고치는 코드를 따로 보관하고 원래 소스도 따로 관리하도록 하자는 것이다

19. Branch 작업 흐름도



20. 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)
```

17. 로컬저장소와 원격저장소연결해제

>git remote -v


>git remote remove 원격저장소별칭 엔터 (원격저장소와의 연결해제)

로컬 저장소의 git 히스토리 삭제

>rm -rf .git

21. 새로운 branch 생성 & 이동 동시에

```
git checkout -b "만들 브랜치명"
```

 브랜치를 그래픽으로 확인하고자 할때
`git log --branches --graph`

22. Merge -a 브랜치와 b 브랜치가 있다고 할때

Case1. a 브랜치와 b 브랜치에서 서로 다른 파일을 수정했을 때 두 파일의 내용이 합쳐진다

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

← 먼저 머지할 브랜치로 이동 , 여기서는 a 브랜치로 이동

← b 브랜치와 merge 진행

Case2. 서로 같은 파일에서 다른 부분을 수정했을 때 파일의 내용이 합쳐진다

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때
수동으로 해결해야 한다!!



Conflict
(충돌)

24. Merge 실습과제 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

Merge 실습과제 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
알아두면 좋은 지식!

25. Branch의 종류 알아보기

master

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

develop

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

release

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

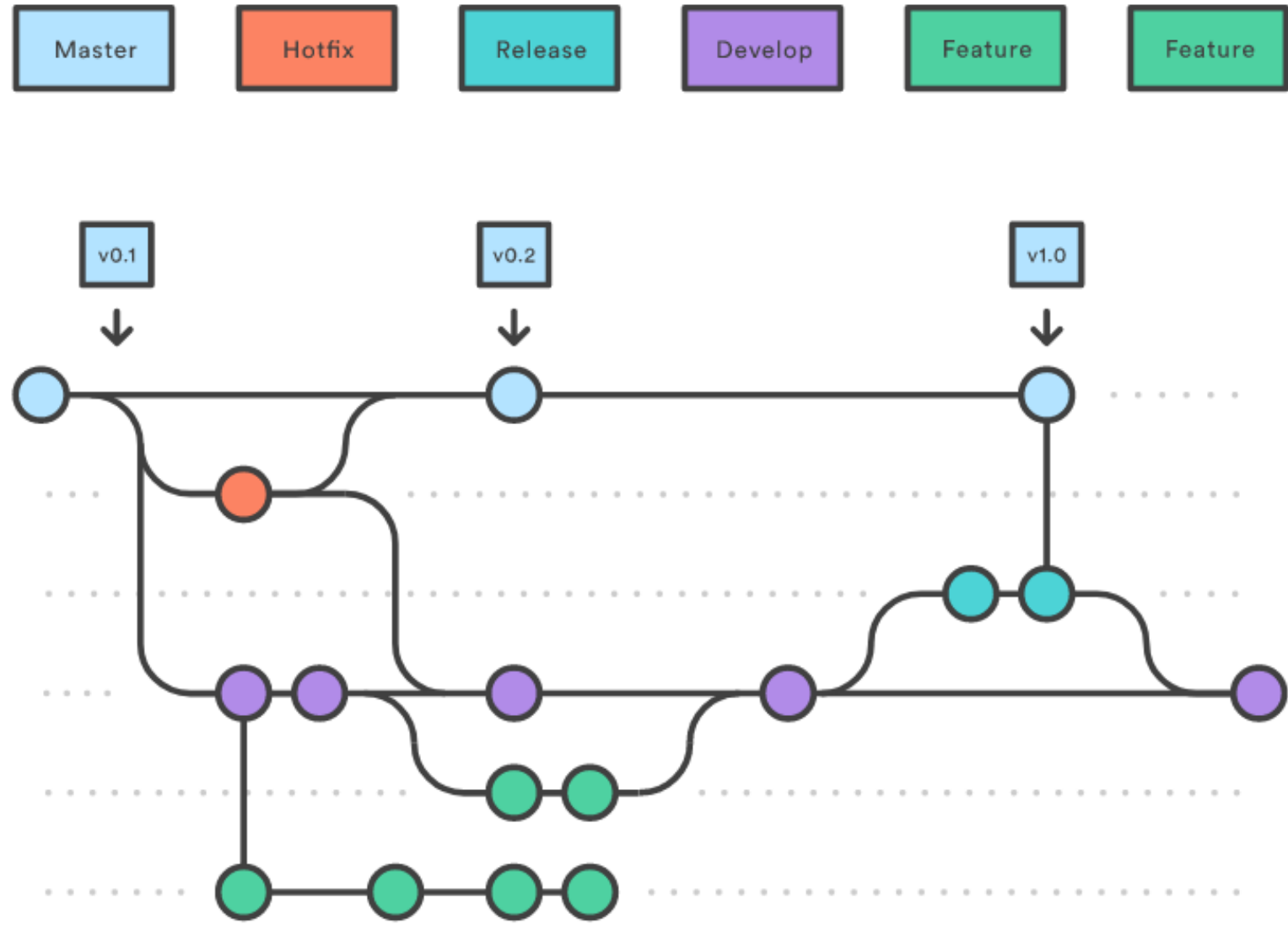
feature

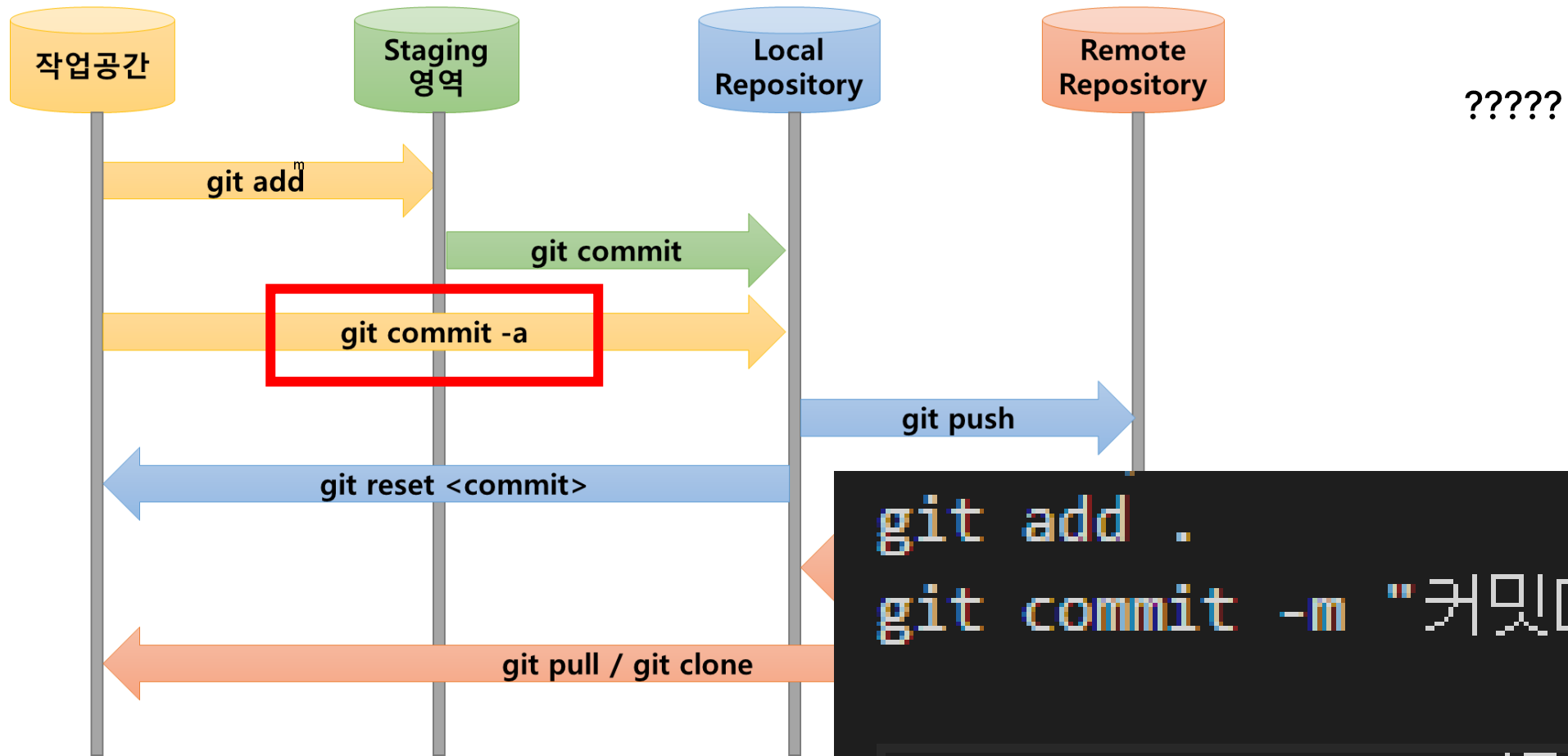
- 기능 개발을 진행하는 브랜치
- 새로운 기능 개발 및 버그 수정을 할 때마다 'develop' 에서 분기
- 공유할 필요가 없어 로컬에서 진행 후 develop 에 merge 해 공유
- 이름 : feature/~~

hotfix

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

Branch 종류

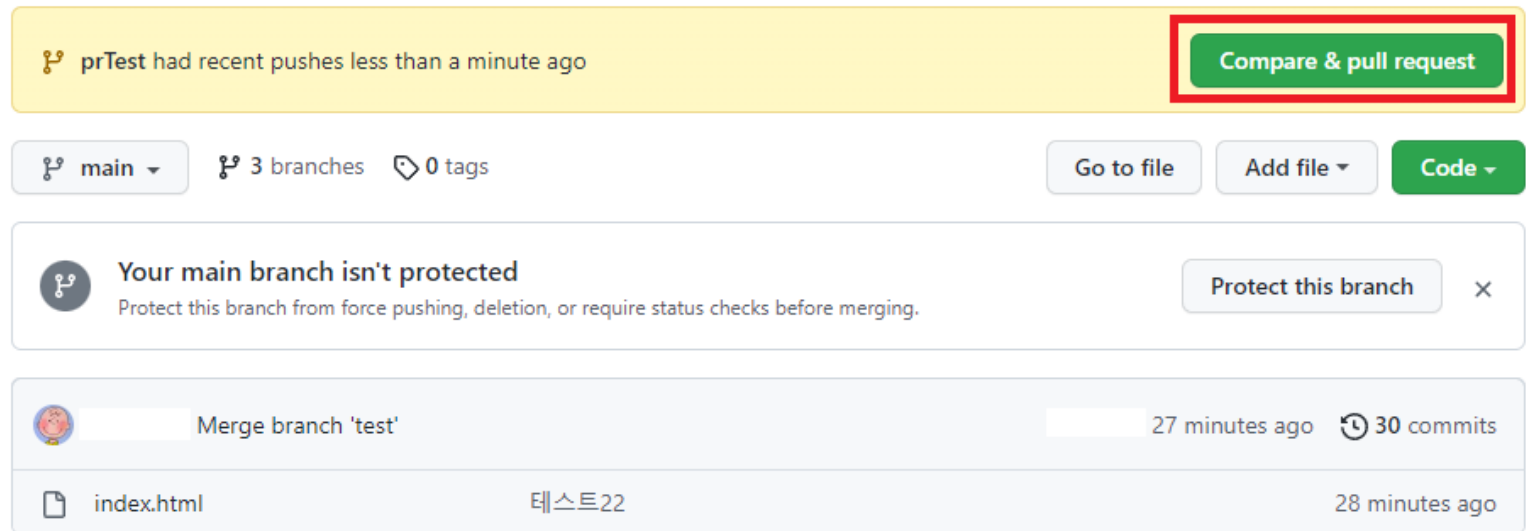




```
git add .  
git commit -m "커밋메세지"  
  
git commit -am "커밋메세지"
```


26. Pull Request란

- 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



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.

test3에서 코드 수정 #1

Merged

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

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

merge 완료!!

27. .gitignore

.gitignore?

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

.gitignore 넣어야 할 내용 예시

***.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.exe 무시)

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

3 directories, 3 files
```

28.Netlify

netlify 란

- 웹 호스팅 서비스
=> 웹 사이트 배포 가능!
- git repository 연결해서
배포 가능

<https://www.netlify.com/>

