



**Wireless Software  
Laboratory**

**Version Control System  
and ETC.**

**wisoft**



### Name

Hyeonsig Jeon

### Major

Data, Indoor Location Algorithm

### Interests

Data Architecture, Software Architecture,  
IS Security

### License

- Data Architecture Professional
- Project Management Professional
- Certified Information System Auditors
- Certified Information Systems Security Professional
- and ETC.

### Ability

Data Design  
& Modeling

1

Database  
Tuning

2

Software  
Design

3

Software  
Development

4



Wireless Software Laboratory

# 소프트웨어 개발 프로세스

wisoft





How the Sponsor explained it



How the project leader understood it



How the analyst designed it



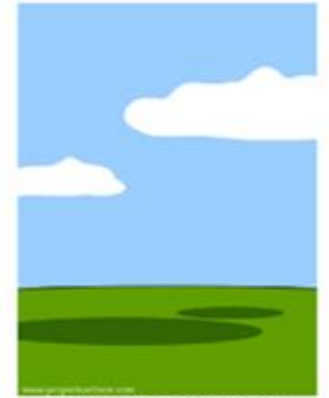
How the programmer wrote it



What the beta testers received



How the business consultant described it



How the project was documented



What operations installed



How the customer was billed



How it was supported



What marketing advertised



When it was delivered



What the customer really needed

SDLC

# 최근 실리콘 밸리의 트렌드

속도가 중요!! **잘**하는 것부터 하자.

스스로 **학습**하고 자기 계발, **내재화**

**소규모** 조직 (10~20명)

개발자가 최고의 **직업**

**변신**과 **협업**, 그리고 **공유**

---

## about SDLC

- Software Development Life Cycle
  - 소프트웨어를 개발하기 위한 정의-개발-유지보수-폐기까지의 과정을 하나의 연속된 주기로 보고, 효과적으로 수행하기 위한 방법론
- 대표적인 SDLC 모델
  - Build-Fix
  - Waterfall
  - Prototyping
  - Rapid Application Development
  - 4<sup>th</sup> Generation Technique

# 소프트웨어 개발 중심의 변화

---

## 시대별 소프트웨어의 중심 가치 및 기술 변화

- **Enterprise**
  - IBM, Microsoft, Oracle, Sun, HP, and etc.  
EJB, JAVA, Servlet/JSP, Oracle, and etc.
- **Network, Social**
  - Google, Facebook, Linked in, and etc.  
Spring, JPA, PHP/ASP, MySQL and etc.
- **Mobile**
  - Twitter, Pinterest, and etc.  
Cloud, Javascript, Node.js, RoR, NoSQL, and etc.

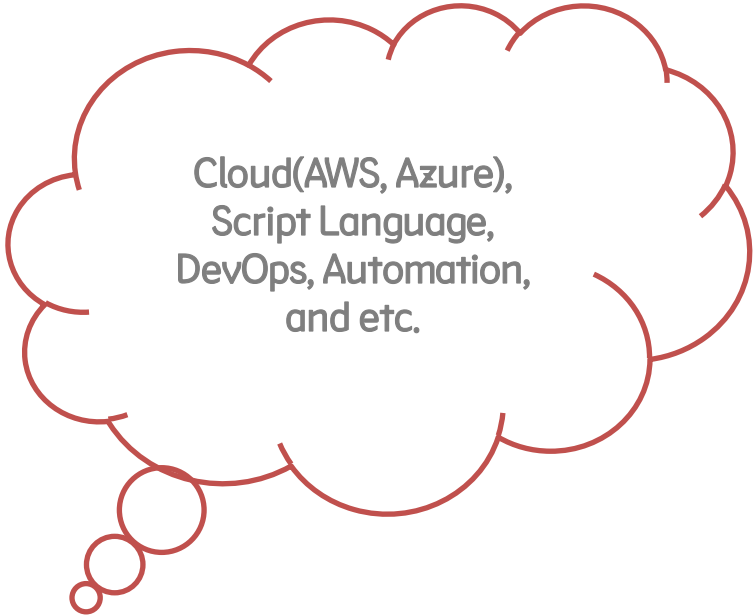


# 왜 이런 현상이 나타날까?

개발자 1인이 해야 할 업무가 많음

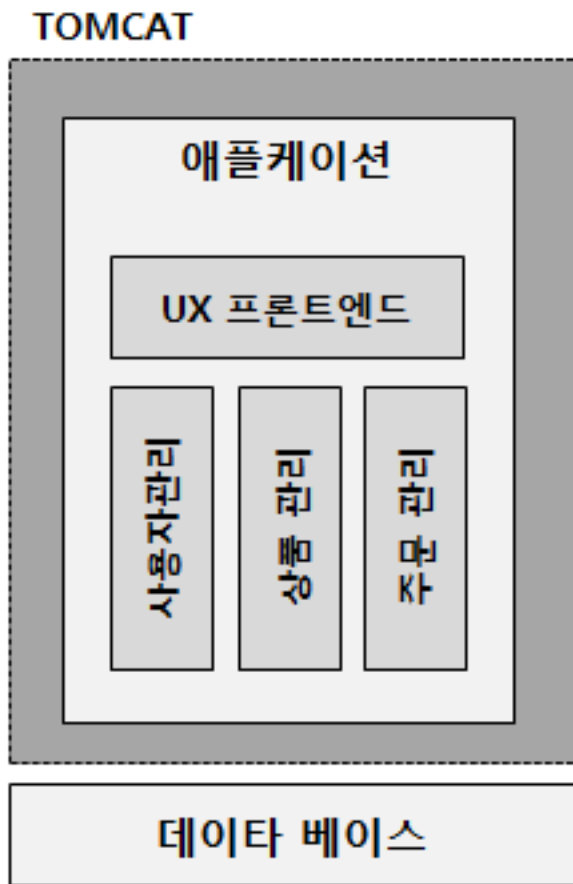
자본의 한계로 말미암아 시장에 빨리 출시하여 인지도를 높여야 함

엄청나게 많은 경쟁자로 빠른 업데이트와 고객의 요구사항에 대한 대응이 필요함



Cloud(AWS, Azure),  
Script Language,  
DevOps, Automation,  
and etc.

# 모놀리식 아키텍처



하나의 서버에 모든 비즈니스 로직이 포함

## 장점

기술 단일화와 관리 용이성

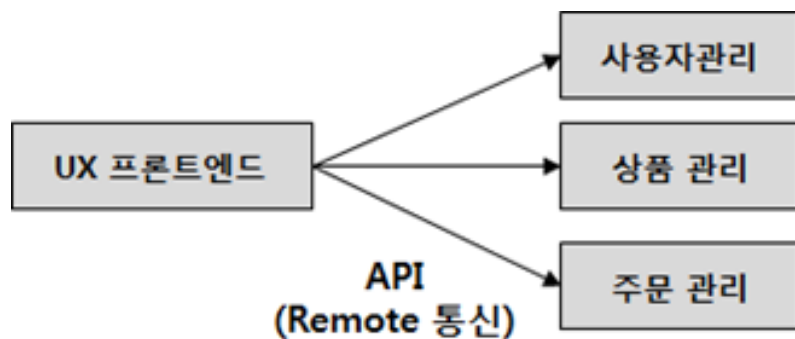
## 단점

배포 및 재기동 시간이 오래 걸림

다른 컴포넌트와의 의존성으로 변경의 어려움

여러 개의 기술을 혼용해서 사용하기 어려움

# MSA



여러 독립 서비스로 나누어 조합하여 서비스를 제공

장점

...

단점

...



# 컨웨이의 법칙

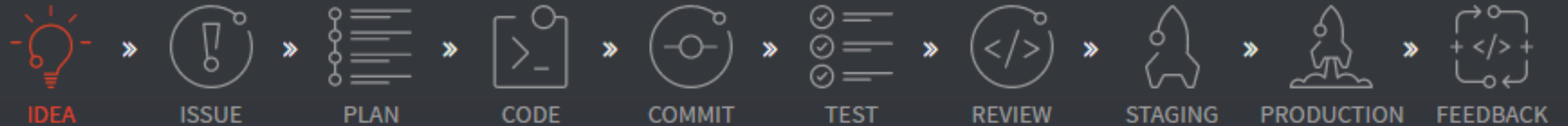
소프트웨어의 구조는 그 소프트웨어를 만드는 팀의 구조와 일치한다.

제대로 된 팀 구조를 만드는 것이 곧 설계이며, 성공의 지름길이다.

# The Modern Development Lifecycle



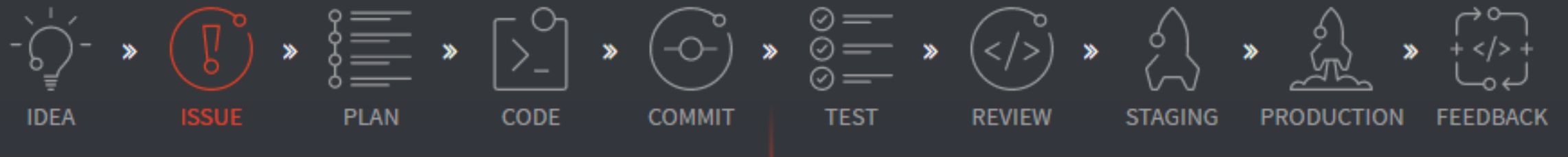
# The Modern Development Lifecycle (1/10)



Chat with your team about ideas

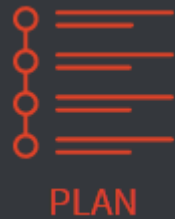
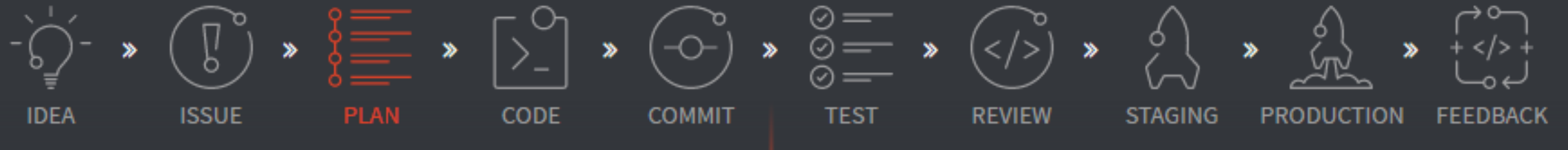


# The Modern Development Lifecycle (2/10)



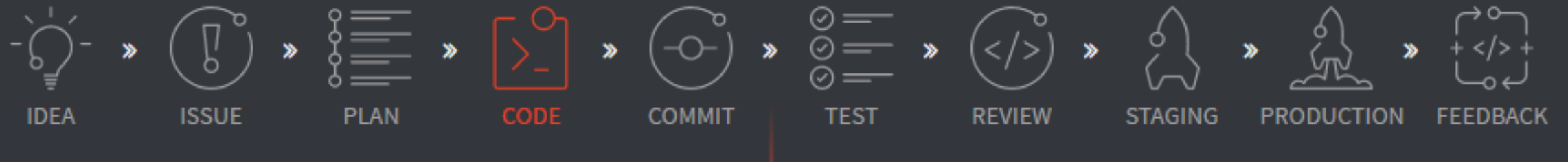
Turn ideas into issues

# The Modern Development Lifecycle (3/10)



Visualize your plans

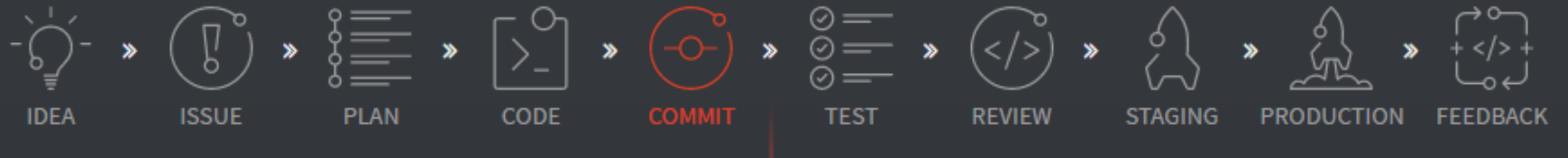
# The Modern Development Lifecycle (4/10)



Contribute right away

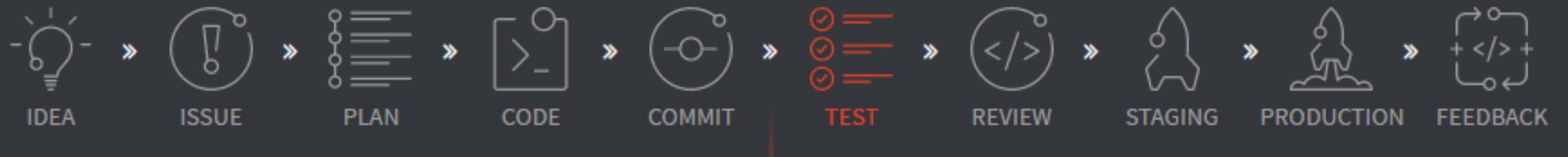


# The Modern Development Lifecycle (5/10)



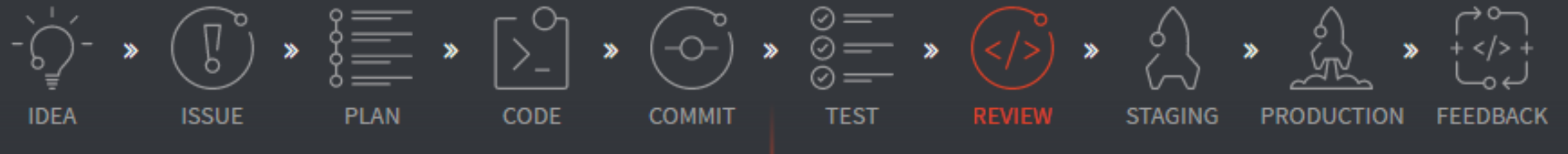
Kick off the feedback process

# The Modern Development Lifecycle (6/10)



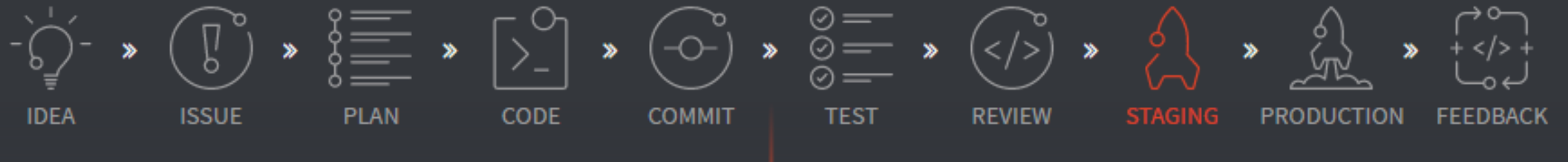
Run tests and monitor progress

# The Modern Development Lifecycle (7/10)



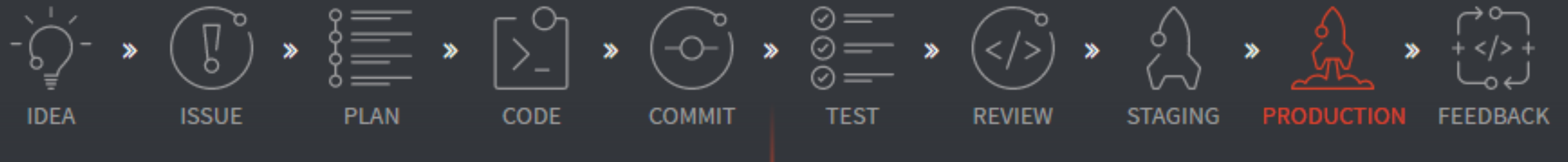
Get feedback from your peers

# The Modern Development Lifecycle (8/10)



Name your environment and deploy to it

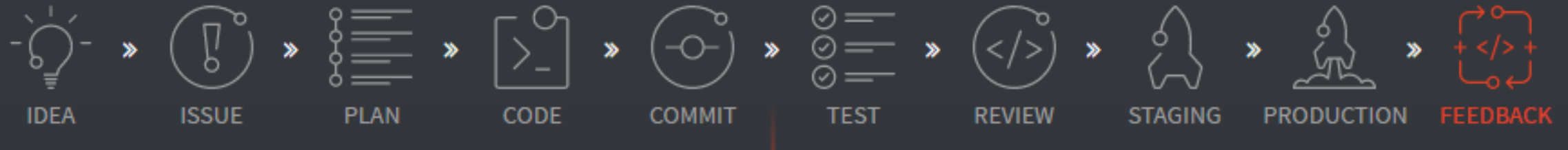
# The Modern Development Lifecycle (9/10)



Close the loop by deploying directly from chat



# The Modern Development Lifecycle (10/10)



Learn from the process and get better each time

*Break Time*



Wireless Software Laboratory

# Git Fundamental



IDEA



ISSUE



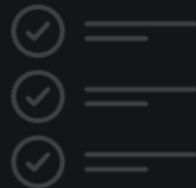
PLAN



CODE



COMMIT



TEST



REVIEW



STAGING



PRODUCTION



FEEDBACK

Hyeonsig Jeon

# SSH Key Configuration

# SSH Key (1/3)

GitLab supports

RSA, DSA, ECDSA, and ED25519  
keys.

## ED25519

```
$ ssh-keygen -o -a 100 -t ed25519 -C "mail@wisoft.io"
```

## RSA

```
$ ssh-keygen -o -t rsa -b 4096 -C "mail@wisoft.io"
```

Ref. <https://linux-audit.com/using-ed25519-openssh-keys-instead-of-dsa-rsa-ecdsa/>

# SSH Key (2/3)

Copy

SSH public key to Clipboard

MacOS

```
$ cat ~/.ssh/id_ed25519.pub | pbcopy
```

Windows

```
$ cat ~/.ssh/id_ed25519.pub | clip
```



# SSH Key (3/3)

Paste

your remote repository

WiSoft GitLab

```
https://git.wisoft.io/profile/keys
```

GitHub

```
https://github.com/settings/keys
```

# GPG Key Configuration

# GPG Key (1/4)

## Installation Guide

### MacOS

- `brew search gpg2`
- `brew install gpg2`

### Windows

- Installed git-bash

## Create GPG Key

```
$ gpg --full-gen-key
```

1. 어떤 알고리즘을 선택할 것인가? (1) RSA and RSA (default)
2. 키의 길이를 선택하세요? 4096
3. 키의 생명주기를 선택하시오? 0 (무한대)

본인의 신원 정보를 입력한 후, 비밀번호 입력(2번).

# GPG Key (2/4)

Copy

GPG public key to Clipboard

Check GPG key

```
$ gpg --list-secret-keys --keyid-format LONG
```

Sample output

```
sec    rsa4096/0x30F2B65B9246B6CA 2020-04-01 [SC]  
       D5E4F29F3275DC0CDA8FFC8730F2B65B9246B6CA  
uid    [ultimate] your_name <your_email>  
ssb    rsa4096/0xB7ABC0813E4028C0 2020-04-01 [E]
```

Copy public key

```
$ gpg --armor --export 0xB7ABC0813E4028C0 | clip  
$ gpg --armor --export 0xB7ABC0813E4028C0 | pbcopy
```

# GPG Key (3/4)

Paste

your remote repository

WiSoft GitLab

```
https://git.wisoft.io/profile/gpg\_keys
```

GitHub

```
https://github.com/settings/keys
```

# GPG Key (4/4)

## Reg. Git Alias

### Registration Git Configure

```
$ git config --global user.signingkey 0xB7ABC0813E4028C0  
$ git config --global commit.gpgsign true
```

### Commit

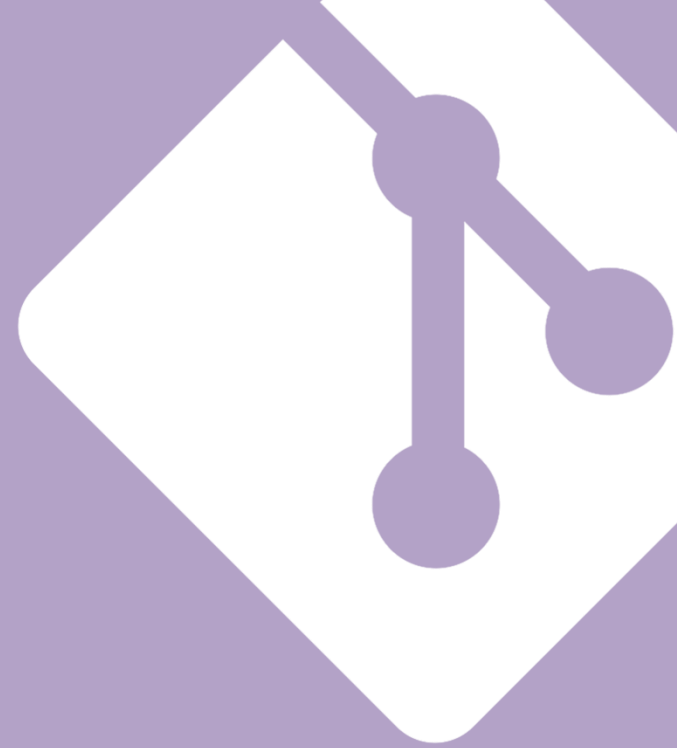
```
$ git commit -S -m "commit message"  
$ git config --global alias.cm "commit -S"
```



# Demonstration

# About Git

wisoft



# 새로운 파일

새로운 작업을 진행하기 위하여  
파일을 하나 생성합니다.



hello.ts

# 작업 중인 파일

파일의 변경 내역을 기록하려면?



hello.ts



history

# 이력 정보를 유지하는 가장 쉬운 방법

- 다른 이름으로 저장하여 여러 파일을 유지하는 방법



hello\_r01.ts



hello\_r02.ts



hello\_r03.ts

## 시간이 흐르면 ...

- Oops! 이런 상황은 우리가 원하는 그림이 아닙니다.



hello\_r01.ts



hello\_r02.ts



Hello\_r03.ts

...



hello\_rn.java



# 파일의 버전을 관리하는 가장 좋은 방법은?



폴더

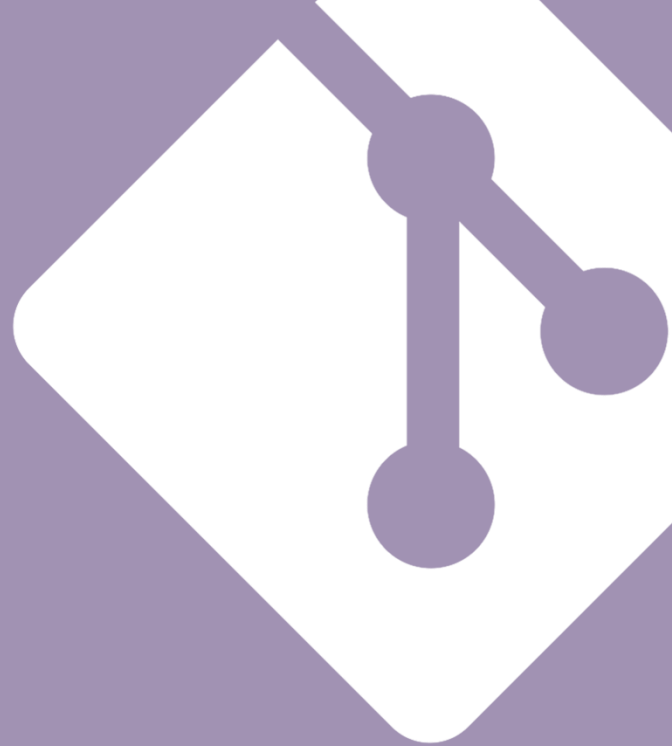
지금까지 여러분의 경험은 어떤가요?



# 그럼 상황을 극복하기 위해서는?

- 파일의 버전을 관리하는 소프트웨어를 사용합니다.





우리는 Git을 사용합니다.

# Git

프로젝트(코드, 리소스 등)를 관리하기 위한  
분산 버전 관리 시스템(DVCS)

네트워크에 접근하거나 중앙 서버에 의존  
하지 않음

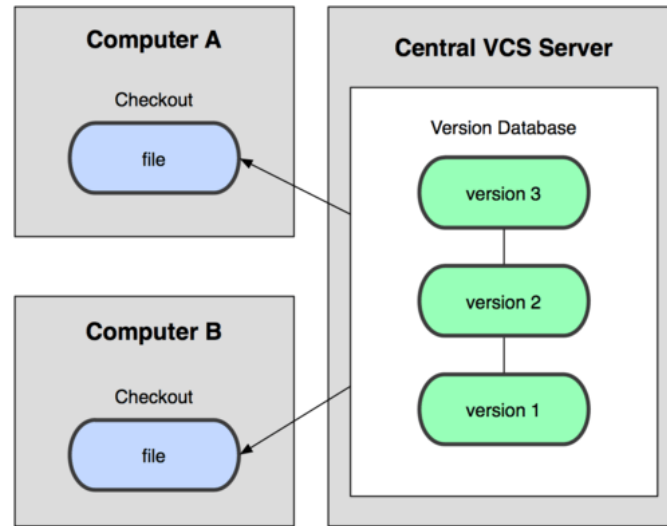
리눅스 커널 개발에 이용하려고 개발  
- Linus Benedict Torvalds



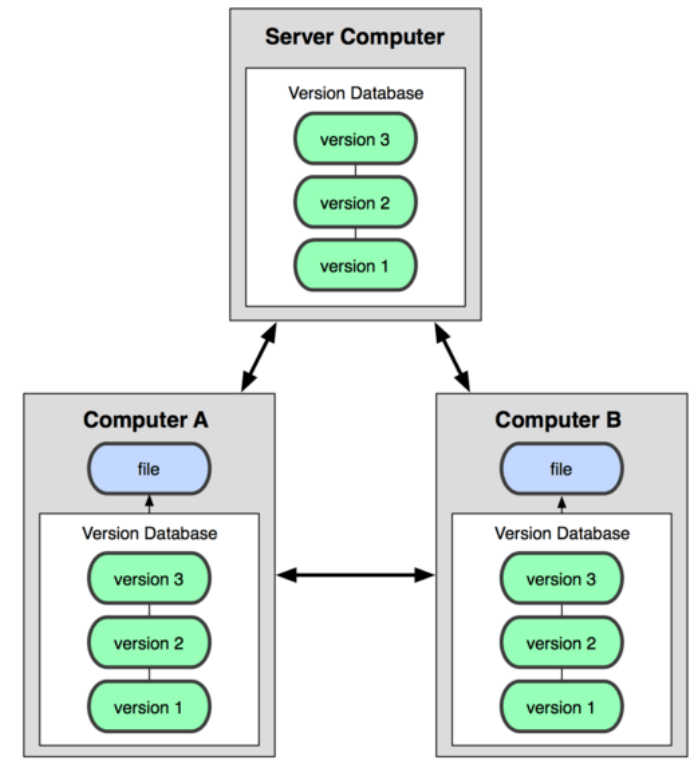
# VCS

## Version Control System

- 파일의 변화를 시간에 따라 기록하여 과거 특정 시점의 버전을 다시 불러올 수 있는 시스템



Centralized Version Control System



Distributed Version Control System

## Hello, World! 구현

(1/2)



hello.ts

### 새로운 요구사항

출력 결과물에 현재 시간을 출력하라.

- hello.ts 파일을 생성하고, 'Hello, World!' 메시지를 출력합니다.

```
class Hello {  
  main(): void {  
    console.log("Hello, World!");  
  }  
}  
  
new Hello().main();
```

```
class Hello {  
  main(): void {  
    let currentTime = new Date().toLocaleTimeString();  
    console.log("Hello, World!");  
    console.log(currentTime);  
  }  
}  
  
new Hello().main();
```

## Hello, World! 구현

(2/2)



hello.ts

### 새로운 요구사항

현재 시간을 삭제하고,  
오늘 년/월/일을 출력하시오.

- 요구사항이 변경되었습니다.

```
class Hello {  
  main(): void {  
    let currentTime = new Date().toLocaleTimeString();  
    console.log("Hello, World!");  
    console.log(currentTime);  
  }  
}
```

```
class Hello {  
  main(): void {  
    let today = new Date().toLocaleDateString();  
    console.log("Hello, World!");  
    console.log(today);  
  }  
}  
  
new Hello().main();
```



# 파일의 버전 관리 기준은?

- 모든 변경 사항을 기록해야 할까? 아니면 더 좋은 방법은?

- 버전 관리 기준은 데이터베이스의 **트랜잭션**처럼~
- 하나의 커밋(Commit)은 여러 파일을 포함할 수 있음
- Q. 파일을 수정하지 않고 새로운 실험을 해보고 싶다면?

# 버전 관리 기준은 데이터베이스의 **트랜잭션**처럼 관리하라.

- 사용자가 파일의 버전 관리가 필요하다고 느낄 때 ...

파일의 수정이 완료되었을 때

git 관리 시스템에게 수정 작업이 끝났다고 알려줘야 함

## COMMIT

# COMMIT



hello.ts

## Commit Message

반드시 기록해야 함  
(조직의 규칙에 맞게...)

COMMIT 3

2020년 4월 3일 @손미연  
날짜 출력 포맷 변경

## Commit Message

issue#3 오늘 날짜(년/월/일) 출력

COMMIT 2

2020년 4월 2일 @서승연  
현재 시간 정보 출력

## Commit Message

issue#2 현재 시간 출력

COMMIT 1

2020년 4월 1일 @김민기  
초기 파일 작성

## Commit Message

issue#1 HelloWorld 파일 작성

# 하나의 Commit은 여러 개의 파일이 포함될 수 있음

- 파일 버전 관리의 기준을 확장하라

하나의 작업(요구사항 등)이 완료 되었을 때, 여러 파일에 영향을 준다면?

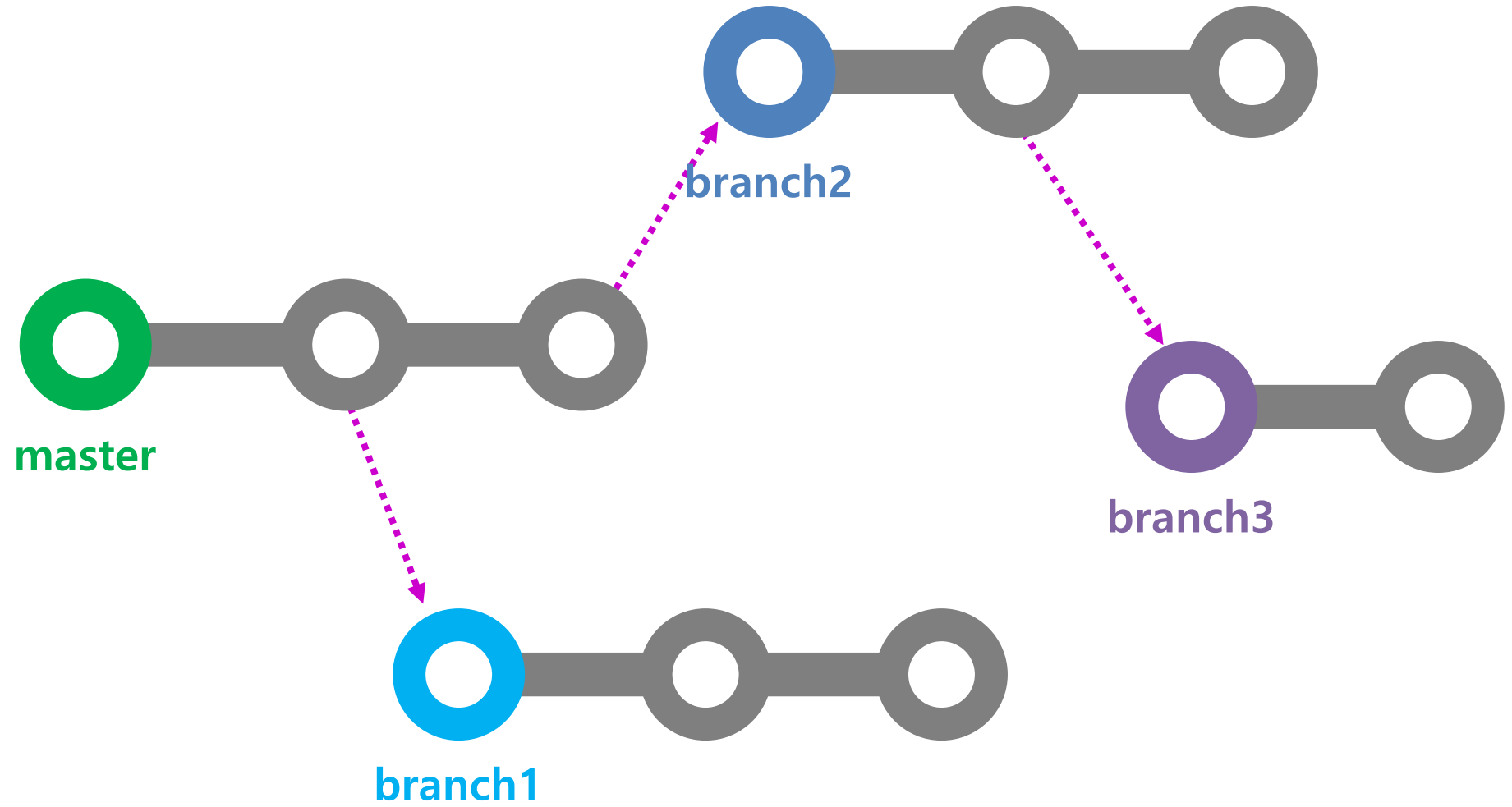
파일 버전 관리의 기준을 어떻게 해야 할까?

# 현재 파일을 수정하지 않고, 새로운 실험을 해보고 싶다면?

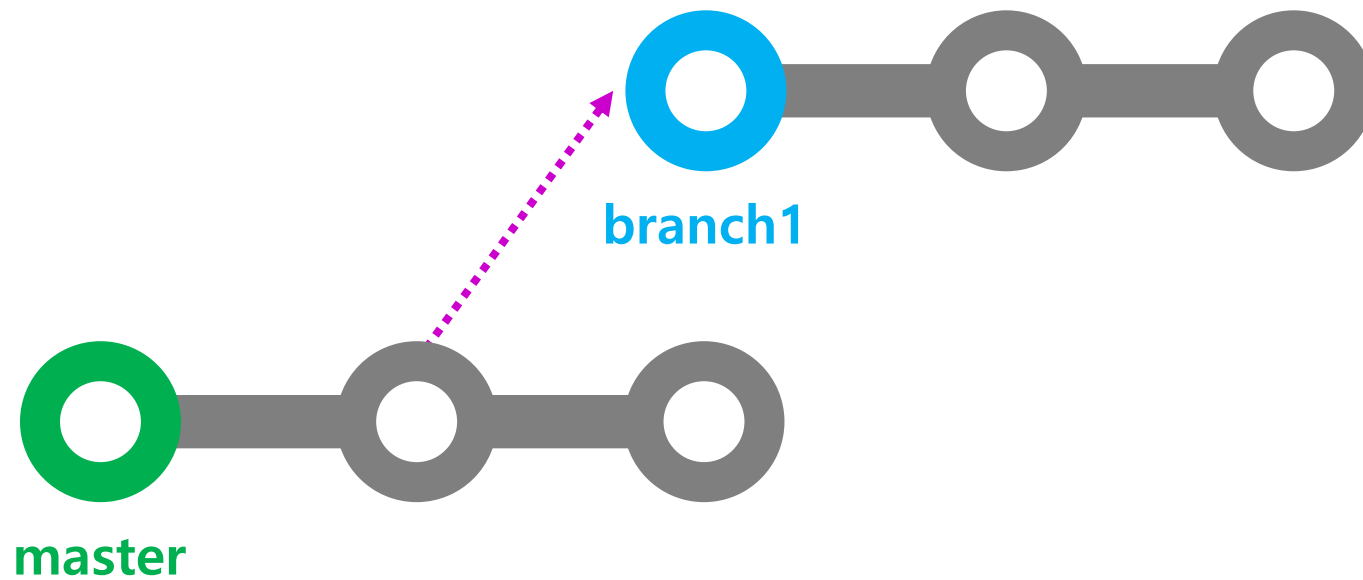
- 새로운 요구사항 등장

- 오늘 날짜와 현재 시간을 함께 출력하시오.

## Branch (1/2)



## Branch (2/2)

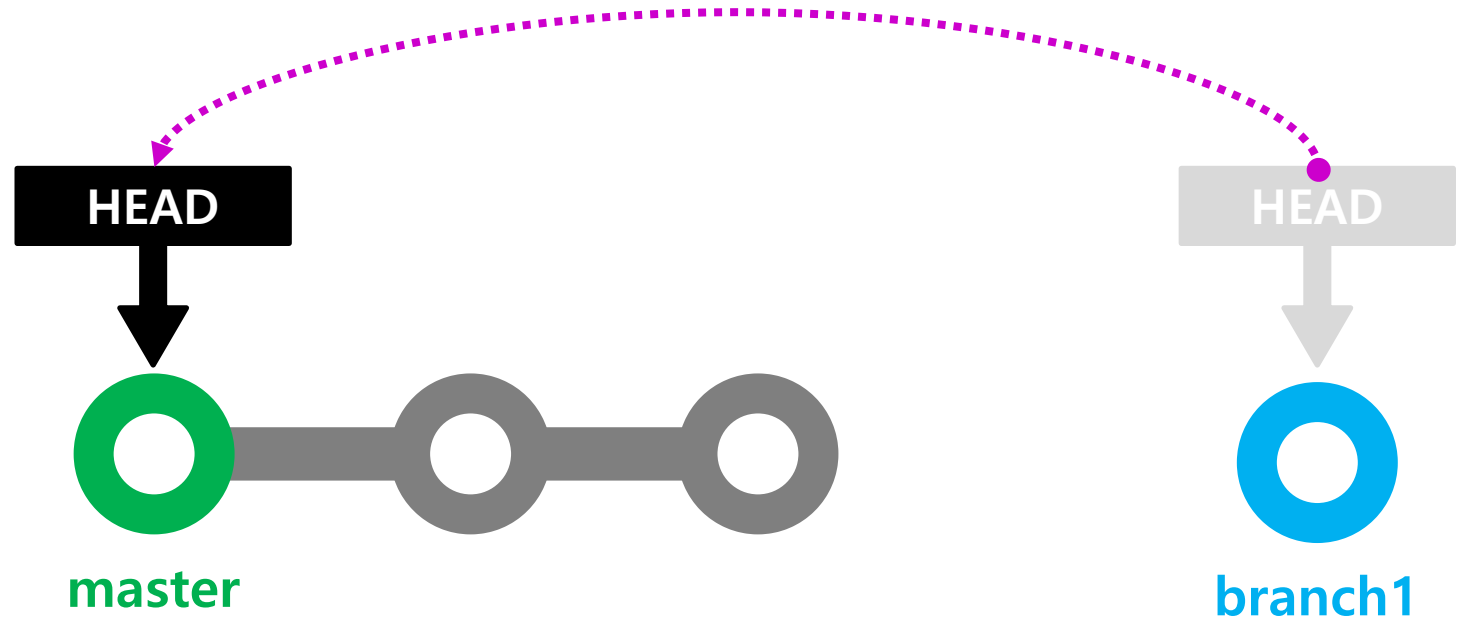


master와 branch1은 완전히 동일한 상태로 복제

그러나 두 공간은 완전히 독립된 공간임

branch1에서 수정한 후 커밋하면 변경사항은 branch1에만 기록됨

## Checkout



branch1에서 실험 과정 중 다른 branch로 이동하고 싶을 때

원한다면 언제든지 다른 branch로 이동할 수 있으며, 이때, branch는 마지막 commit 상태를 유지함



---

## 주요 용어 정리

### COMMIT

사용자가 버전 관리가 필요할 때, 수정 내역을 시스템(git)에 반영한다.

### BRANCH

완전히 독립된 작업 공간을 생성한다.

### CHECKOUT

독립된 작업 공간(BRANCH)을 자유롭게 이동한다.

### MERGE

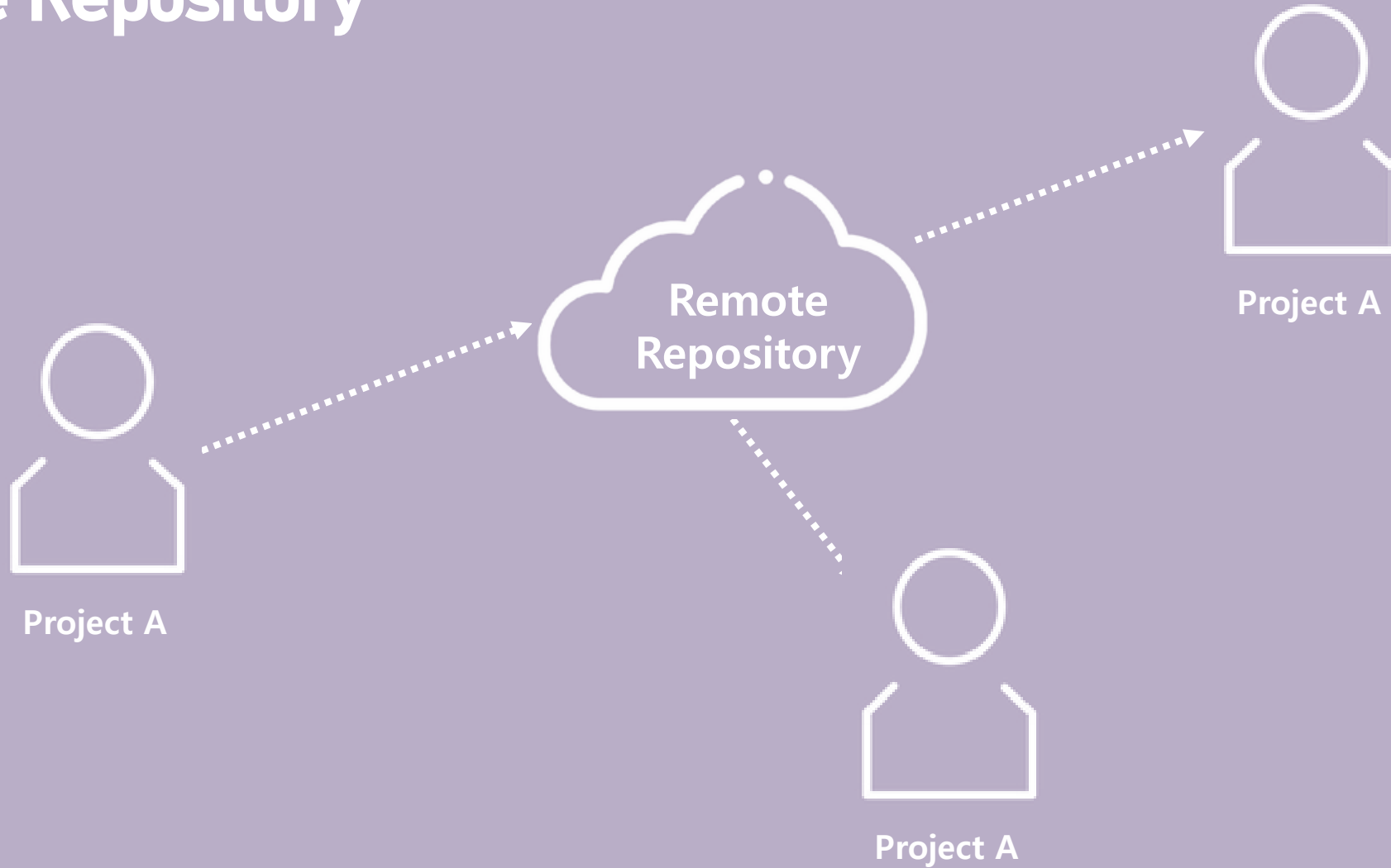
독립된 두 개의 작업 공간(BRANCH)의 내용을 병합한다.

# 아직 해결되지 않은 미션

협업

동료와 함께 작업하려면?

# Remote Repository



---

## 주요 용어 정리

### CLONE

리모트 저장소에서 처음으로 저장소를 내려 받는 작업

### PULL

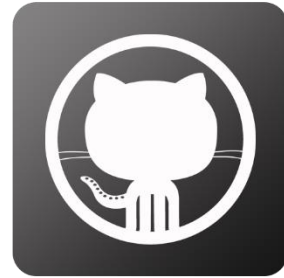
리모트 저장소의 변경 내용을 로컬 저장소에 적용하는 작업

### PUSH

로컬 저장소에서 작업한 내용을 리모트 저장소로 보내는 작업

---

Remote git  
repository service



**GitHub**

<https://www.github.com>



**Bitbucket**

<https://www.bitbucket.org>



**Gitlab**

<https://www.gitlab.com>

*Coffee Break*

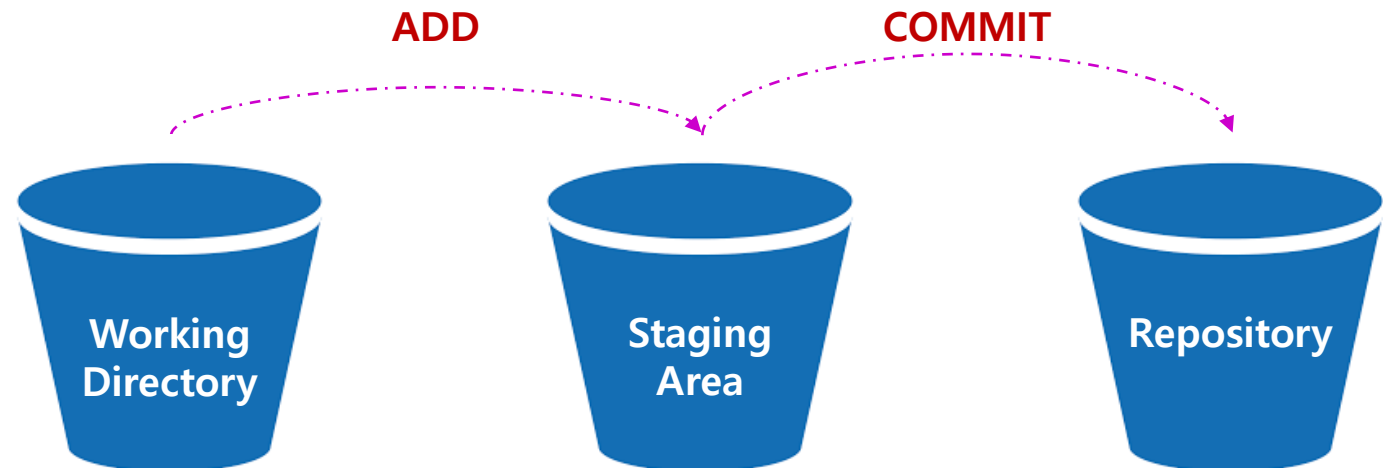




# Git Command

## Git Terms

- Working Copy, Staging Area(Index), Repository





---

## help

- 도움말

```
$ git help command
```

- Example

```
$ git help command
```

```
$ git command --help
```

```
$ man git-command
```

---

## add

- Working Directory의 파일을 추적할 수 있도록 Staging Area에 등록

```
$ git add file_name
```

- Example

```
$ git add *  
$ git add .  
$ git add *.java  
$ git add README.md
```

```
$ git add -A  
$ git add --all
```

```
$ git add -i
```

---

commit

alias: cm

- 변경 내용을 리파지터리에 반영

```
$ git commit
```

- Example

```
$ git commit
```

```
$ git commit -m "commit_message"
```

```
$ git commit -a
```

```
$ git commit -a -m "commit_message"
```

```
$ git commit --amend
```

---

branch

alias: br

- 브랜치 확인 및 생성

```
$ git branch branch_name
```

- Example

```
$ git branch
```

```
$ git branch -r
```

```
$ git branch -a
```

```
$ git branch -v
```

```
$ git branch --merged
```

```
$ git branch --no-merged
```

```
$ git branch brach_name
```

```
$ git branch -m name_from name_to
```

```
$ git branch -d branch_name
```

---

checkout

alias: co

- 다른 커밋으로 이동 (브랜치, 체크섬, 태그 등)

```
$ git checkout branch_name
```

- Example

```
$ git checkout master
```

```
$ git checkout checksum
```

```
$ git checkout tag_version
```

```
$ git checkout -b branch_name
```

```
$ git checkout HEAD files
```

---

clone

alias: cl

- 현재 디렉토리에 저장소를 생성하고, 리모트 저장소의 데이터를 내려 받음.

```
$ git clone repository_path [directory_name]
```

- Example

```
$ git clone $HOME/documents/project/  
$ git clone file:///c:/users/hsjeon/gitrepo/test gitrepo  
$ git clone https://git.wisoft.io/hyeonsig/test.git  
$ git clone git@git.wisoft.io:hyeonsig/test.git
```

```
$ git clone --depth 100 repository_path
```

---

diff

alias: df

- 파일 비교.

```
$ git diff
```

- Example

```
$ git diff
```

```
$ git diff --cached
```

```
$ git diff --staged
```

---

fetch

alias: f

- 리모트 저장소의 데이터 가져오기.

```
$ git fetch remote_repository
```

- Example

```
$ git fetch
```

```
$ git fetch repo_name branch_name
```

```
$ git fetch --tags
```



---

## init

- Git Repository 생성.

```
$ git init
```

- Example

```
$ git init
```

```
$ git init -bare
```

---

## log

- Commit History 조회.

```
$ git log
```

- Example

```
$ git log
```

- Sample

```
[alias]
```

```
...
```

```
lg = log --graph --pretty=format:'%C(auto)%h -  
%d %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit  
--date=relative
```

---

## merge

- 브랜치를 병합.

```
$ git merge branch_name
```

- Example

```
$ git merge master
```

```
$ git checkout master;  
$ git merge issue#1;
```

```
$ git merge -no-commit
```

---

mv

- 파일명 수정.

```
$ git mv filename_from filename_to
```

- Example

```
$ git mv modify.txt modified.txt
```

---

## pull

- 리모드 저장소의 데이터 내려 받기. 머지를 자동으로 수행.

```
$ git pull remote_repo branch_name
```

- Example

```
$ git pull
```

```
$ git pull origin master
```

---

## push

- 리모드 저장소에 데이터 올리기

```
$ git push remote_repo branch_name
```

- Example

```
$ git push
```

```
$ git push origin master
```

```
$ git push remote_repo tag
```

---

## rebase

- 브랜치 병합.

```
$ git rebase branch_name
```

- Merge와의 차이점

- Merge는 두 브랜치의 최종결과만을 기준으로 병합
- Rebase는 브랜치의 변경사항을 순서대로 다른 브랜치에 적용하며 병합  
저장소의 커밋 로그와 이력을 한 줄로 정리하며, 보통 완료된 브랜치를 마스터에  
병합할 때 사용.

---

## reset

- 스테이징/커밋 취소.

```
$ git reset HEAD [file_name]
```

- add로 인덱스에 등록한 파일을 unstaged 상태로 바꿈
- 파일을 명시하지 않으면 모두 되돌림.

- Example

```
$ git reset HEAD~1
```



---

## revert

- 커밋 취소.

```
$ git revert HEAD
```

- HEAD가 부모 커밋으로 되돌아가는 것이 아니라 새로운 커밋을 생성하여 기록.

- Example

```
$ git revert HEAD
```

---

rm

- 파일 및 폴더 삭제.

```
$ git rm file_name
```

- git이 추적중인 파일 혹은 폴더에만 사용할 수 있음

- Example

```
$ git rm README.md
```

```
$ git rm --cached README.md
```

---

## remote

- 리모트 리파지터리 제어.

```
$ git remote
```

- Example

```
$ git remote
```

```
$ git remote -v
```

```
$ git remote add repo_name repo_path
```

```
$ git remote show repo_name
```

```
$ git remote rename name_from name_to
```

```
$ git remote rm repo_name
```

```
$ git remote prune repo_branch_name
```

---

## show

- 커밋 정보 조회.

```
$ git show [HEAD, TAG, CHECKSUM]
```

- Example

```
$ git show v1.1
```

```
$ git show 1c00392...
```

---

## status

- 저장소 상태 확인.

```
$ git status
```

- Example

```
$ git status
```

*Coffee Break*





# Exercise

# Create Local Repository

- Command

```
$ git init
```

- 프로젝트 정보를 바꾸고 싶다면?

```
$ git config user.name "Hyeonsig"
```



# Initial Commit

- Command

```
$ git commit -S --allow-empty -m "Initial repository"  
$ git log
```

---

## Exercise

- git-tutorial Repository를 생성합니다.
- “Initial repository” 커밋을 진행한 후, 커밋 로그를 확인합니다.

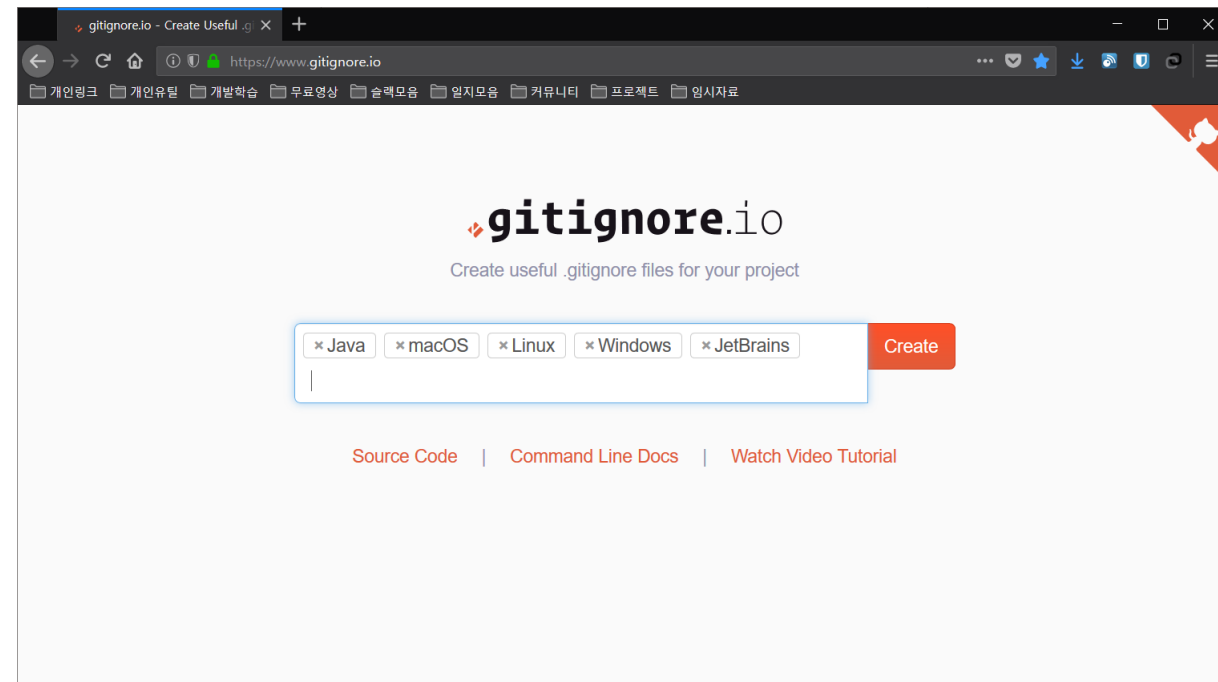
# .gitignore

- Create .gitignore

- <https://gitignore.io>

- Input OS type, IDEs, Programming Languages, ...

- Click Create button



# .gitignore (Command)

- Git Configure

```
git config --global alias.ignore \  
'!gi() { curl -sL https://www.gitignore.io/api/$@ ;}; gi'
```

- Bash

```
echo "function gi() { curl -sL https://www.gitignore.io/api/\$@ ;}" >> \  
~/.bash_profile && source ~/.bash_profile
```

- Zsh

```
echo "function gi() { curl -sLw "\n"  
https://www.gitignore.io/api/\$@ ;}" >> ~/.zshrc && source ~/.zshrc
```

---

## Exercise

- git-tutorial Repository를 생성합니다.
- .gitignore 파일을 생성합니다.
- .gitignore 파일을 생성한 git-tutorial Repository에 커밋합니다.

# Commit template (1/3)

- Commit Template

/configuration/git/templates/message/commit

Subject line limit 50 characters

More detailed explanatory text, if necessary. Wrap the body at 72 characters.

Resolves: #issueNo

See also: #issueNo

## Commit template (2/3)

- Command

```
$ git config --global commit.template \  
"/configuration/git/templates/message/commit"
```

# Commit template (3/3)

- Using visual studio code

> Command palette > Preferences: Open Settings (JSON)

```
{  
  "[git-commit]": {  
    "editor.fontFamily": "D2Coding",  
    "editor.rulers": [50, 72]  
  },  
}
```



---

Create

Example File

- Create Example File - index.html

```
<html>
  <head>
    <title>WiSoft.io Homepage</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

---

## Exercise

- 작성한 파일을 Commit 하시오.

- Subject Line

WiSoft.io 홈페이지 메인 화면 작성

- Body

Title에 WiSoft.io Homepage를 등록하고, 본문에 Hello, World! 메시지를 입력했습니다.

---

## Modify

## Example File

- Create Example File - index.html

```
<html>
  <head>
    <title>WiSoft.io Homepage</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <h2>WiSoft.io 홈페이지에 방문해 주셔서 감사합니다.</h2>
  </body>
</html>
```

---

## Exercise

- 작성한 파일을 Commit 하시오.

- Subject Line

WiSoft.io 홈페이지 메인 화면 수정 - 메시지 추가

- Body

본문에 환영 인사 메시지 구문을 입력했습니다.

## Modify

## Exercise File

- Modify Exercise File - index.html
  - 수평선, 작성자, 작성 일자, 그리고 작성 목적을 추가합니다.  
단, 한 문장을 입력할 때마다 Commit 하시오. (총 4번의 커밋 필요)

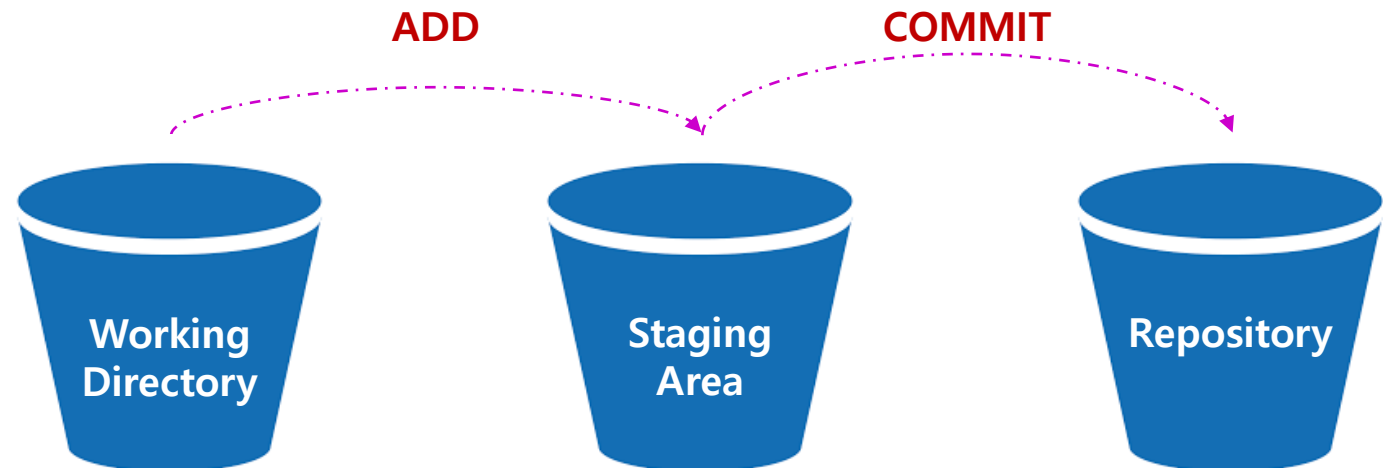
...

```
<body>
  <h1>Hello, World!</h1>
  <h2>WiSoft.io 홈페이지에 방문해 주셔서 감사합니다.</h1>
  <hr/>
  // 작성자
  // 작성 일자
  // 작성 목적
</body>
```

...

## Git Terms

- Working Directory, Staging Area(Index), Repository



# Current Status

- Current Status

2018-about-git

Commit Pull Push Fetch Branch Merge Stash Discard Tag

Git-flow Remote Terminal Explorer Settings

FILE STATUS  
Working Copy

BRANCHES  
master

TAGS

REMOTES

STASHES

All Branches Show Remote Branches Date Order

Graph	Description	Date	Author	Commit
• master 작성목적 추가		28 8 2018 20:31	hyeonsig <hyeonsig@wisoft.io>	ea00138
• 작성일자 추가		28 8 2018 20:31	hyeonsig <hyeonsig@wisoft.io>	3d066b2
• 작성자 추가		28 8 2018 20:30	hyeonsig <hyeonsig@wisoft.io>	3fe82cc
• 수평선 추가		28 8 2018 20:29	hyeonsig <hyeonsig@wisoft.io>	c88eef3
• WiSoft.io 홈페이지 메인 화면 작성		28 8 2018 20:17	hyeonsig <hyeonsig@wisoft.io>	e8c2f23
• Add .gitignore		28 8 2018 20:11	hyeonsig <hyeonsig@wisoft.io>	a82ddd4

Pending files, sorted by file status

Commit: ea00138ba6a9df5052a7626e4b2647e335943ee0 [ea00138]  
Parents: 3d066b2157  
Author: hyeonsig <hyeonsig@wisoft.io>  
Date: 2018년 8월 28일 화요일 오후 8:31:30  
Committer: hyeonsig

작성목적 추가

index.html

Hunk 1 : Lines 10-16

```
<hr />
<h3>작성자: Hyeonsig Jeon</h3>
<h3>작성일자: 2018년 8월 28일</h3>
+ <h3>작성목적: Git seminar</h3>
</body>
</html>
```

File Status Log / History Search

# Current Status

- Command

```
$ git log
```



---

## Modify

## Exercise File

- Modify Exercise File - index.html
  - 작성한 3 문장을 삭제하시오.

...

```
<body>
```

```
  <h1>Hello, World!</h1>
```

```
  <h2>WiSoft.io 홈페이지에 방문해 주셔서 감사합니다.</h1>
```

```
  <hr/>
```

```
</body>
```

...

# Discard

이전 상황으로 돌아가고 싶으면 어떻게 해야 할까?

바로 이전 Commit 상태로 돌아가는 명령

# Discard

- Command

```
$ git checkout -- index.html
```

---

## Exercise

- 삭제한 3 라인을 복원하시오.

# Revert

버전을 유지하면서 이전 Commit 상태로 돌아가는 명령

# Revert

- Command

```
$ git revert checksum
```

```
$ git revert checksum1..checksum2
```

3<sup>rd</sup> Commit 단계로 돌아가시오.

Attention

Revert는 반드시 이전 모든 버전을 역순으로 수행

# Reset

선택한 이전 Commit 상태로 돌아가는 명령



# Reset

- Command

```
$ git reset --option checksum | git reset HEAD~No
```

```
$ git reset --soft checksum
```

```
$ git reset --hard checksum
```

```
$ git reset --mixed checksum
```

---

## Reset Mode

- Hard Mode
  - 현재 시점의 Working Copy 버전도 모두 제거
- Mixed Mode
  - 현재 시점의 Working Copy 버전을 유지하지만 Index는 취소
- Soft Mode
  - 현재 시점의 Working Copy 버전과 Index를 모두 유지

# Exercise

결과를 짐작해보면 무슨 모드 일까요?

The screenshot shows the Git GUI interface for a repository named "2018-about-git". The top menu bar includes File, Edit, View, Repository, Actions, Tools, and Help. Below the menu is a toolbar with icons for Commit, Pull, Push, Fetch, Branch, Merge, Stash, Discard, and Tag. The left sidebar shows the repository structure with sections for FILE STATUS (Working Copy), BRANCHES (master), TAGS, REMOTES, and STASHES. The main area displays a commit history table with columns for Graph, Description, Date, Author, and Commit. The current commit is highlighted with a red dashed box.

Graph	Description	Date	Author	Commit
●	작성목적 추가	28 8 2018 20:31	hyeonsig <hyeonsig@wisoft.io>	ea00138
●	작성일자 추가	28 8 2018 20:31	hyeonsig <hyeonsig@wisoft.io>	3d066b2
●	작성자 추가	28 8 2018 20:30	hyeonsig <hyeonsig@wisoft.io>	3fe82cc
●	수평선 추가	28 8 2018 20:29	hyeonsig <hyeonsig@wisoft.io>	c88eef3
●	WISOFT.io 홈페이지 메인 화면 작성	28 8 2018 20:17	hyeonsig <hyeonsig@wisoft.io>	e8c2f23
●	Add .gitignore	28 8 2018 20:11	hyeonsig <hyeonsig@wisoft.io>	a82ddd4

Below the commit history, the "Pending files, sorted by file status" section shows a list of files. The "index.html" file is selected, and its diff is shown in the right pane. The diff shows a hunk of code with a new line added (indicated by a green background).

```
Hunk 1 : Lines 10-16
10 10      <hr />
11 11      <h3>작성자: Hyeonsig Jeon</h3>
12 12      <h3>작성일자: 2018년 8월 28일</h3>
13 13      + <h3>작성목적: Git seminar</h3>
14 14      </body>
15 15      </html>
```

# Revert vs. Reset

- Revert

- 선택한 버전을 취소하여 그 이전 상태로 돌아가는 것
- 이전 버전과 새로운 버전 유지

- Reset

- 선택한 버전의 상태로 돌아가는 것
- 이전 버전과 새로운 버전 삭제

# Branch

독립적으로 어떤 작업을 진행하기 위한 개념

---

## Branch (1/6)

- Requirement

- Normal

연구실 멤버 소개

- Challenge

HTML5를 위한 메타 태그 등록

---

## Branch (2/6)

- Create Branch

```
$ git branch branch_name
```

```
$ git branch meta-tag
```

---

## Exercise

- master 브랜치에 연구실 멤버를 등록하고 Commit 합니다.
  - 교수님
  - 박사과정
  - 석사과정
  - 학부과정



---

## Branch (3/6)

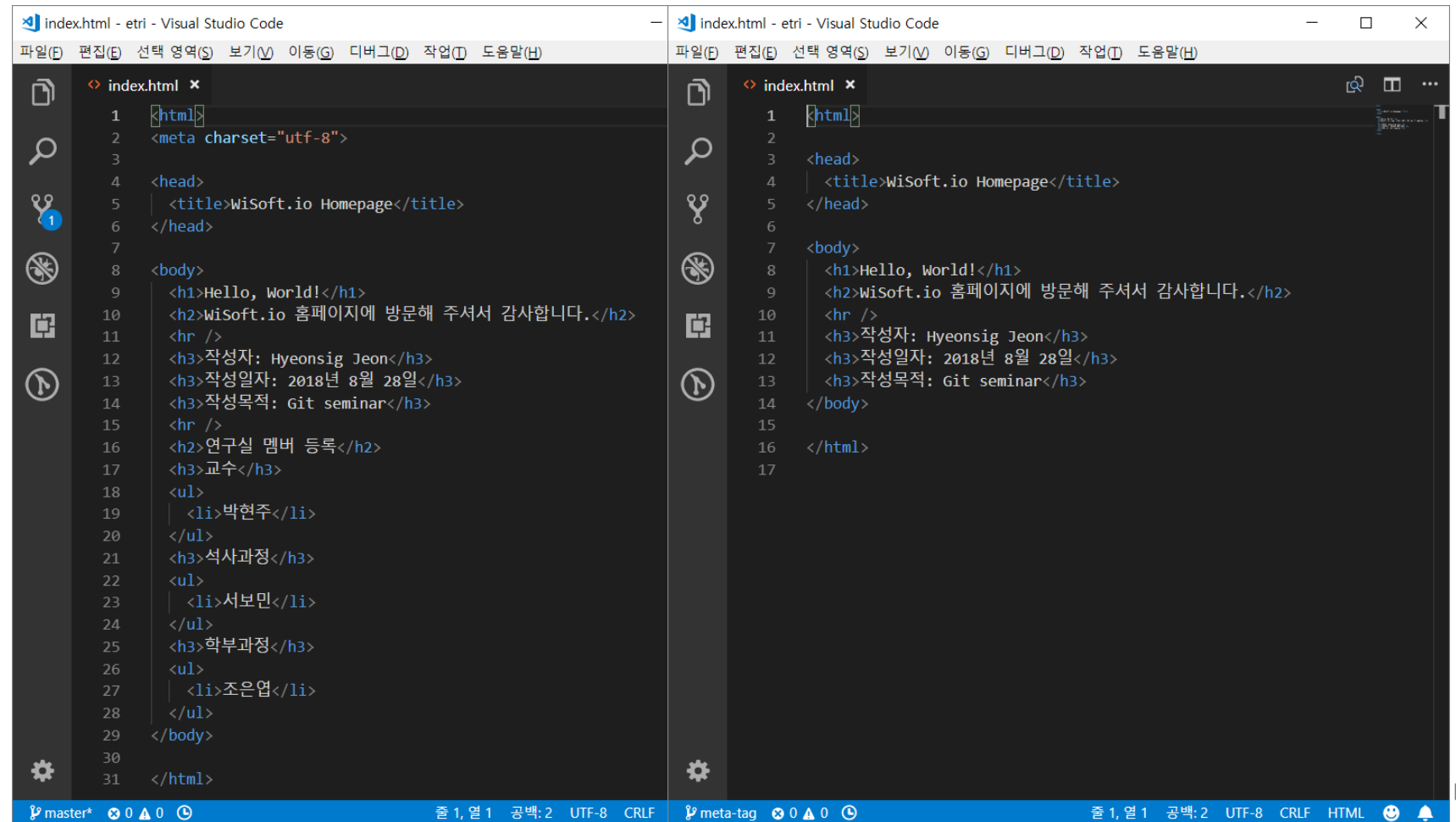
- master Branch

```
$ git log
```

## Branch (4/6)

- meta-tag Branch

```
$ git checkout meta-tag
```



The image displays two side-by-side screenshots of the Visual Studio Code editor, both showing the 'index.html' file. The left window represents the 'master' branch, and the right window represents the 'meta-tag' branch.

**Left Window (master):**

```
1 <html>
2 <meta charset="utf-8">
3
4 <head>
5   <title>WiSoft.io Homepage</title>
6 </head>
7
8 <body>
9   <h1>Hello, World!</h1>
10  <h2>WiSoft.io 홈페이지에 방문해 주셔서 감사합니다.</h2>
11  <hr />
12  <h3>작성자: Hyeonsig Jeon</h3>
13  <h3>작성일자: 2018년 8월 28일</h3>
14  <h3>작성목적: Git seminar</h3>
15  <hr />
16  <h2>연구실 멤버 등록</h2>
17  <h3>교수</h3>
18  <ul>
19    <li>박현주</li>
20  </ul>
21  <h3>석사과정</h3>
22  <ul>
23    <li>서보민</li>
24  </ul>
25  <h3>학부과정</h3>
26  <ul>
27    <li>조은엽</li>
28  </ul>
29 </body>
30
31 </html>
```

**Right Window (meta-tag):**

```
1 <html>
2
3 <head>
4   <title>WiSoft.io Homepage</title>
5 </head>
6
7 <body>
8   <h1>Hello, World!</h1>
9   <h2>WiSoft.io 홈페이지에 방문해 주셔서 감사합니다.</h2>
10  <hr />
11  <h3>작성자: Hyeonsig Jeon</h3>
12  <h3>작성일자: 2018년 8월 28일</h3>
13  <h3>작성목적: Git seminar</h3>
14 </body>
15
16 </html>
17
```

The status bar at the bottom of each window confirms the current branch: 'master\*' on the left and 'meta-tag' on the right.

## Exercise

- meta-tag branch에 meta 정보를 입력하고 Commit 합니다.

...

```
<head>
```

```
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
```

```
  <meta name="description" content="WiSoft.io Homepage" />
```

```
  <meta name="keywords" content="HBNU, WiSoft.io" />
```

```
</head>
```

...

---

## Branch (5/5)

- meta-tag Branch

```
$ git log
```

---

## Branch (6/6)

- Branch Merge
  - meta-tag를 master로 Merge

```
$ git checkout master  
$ git merge meta-tag
```

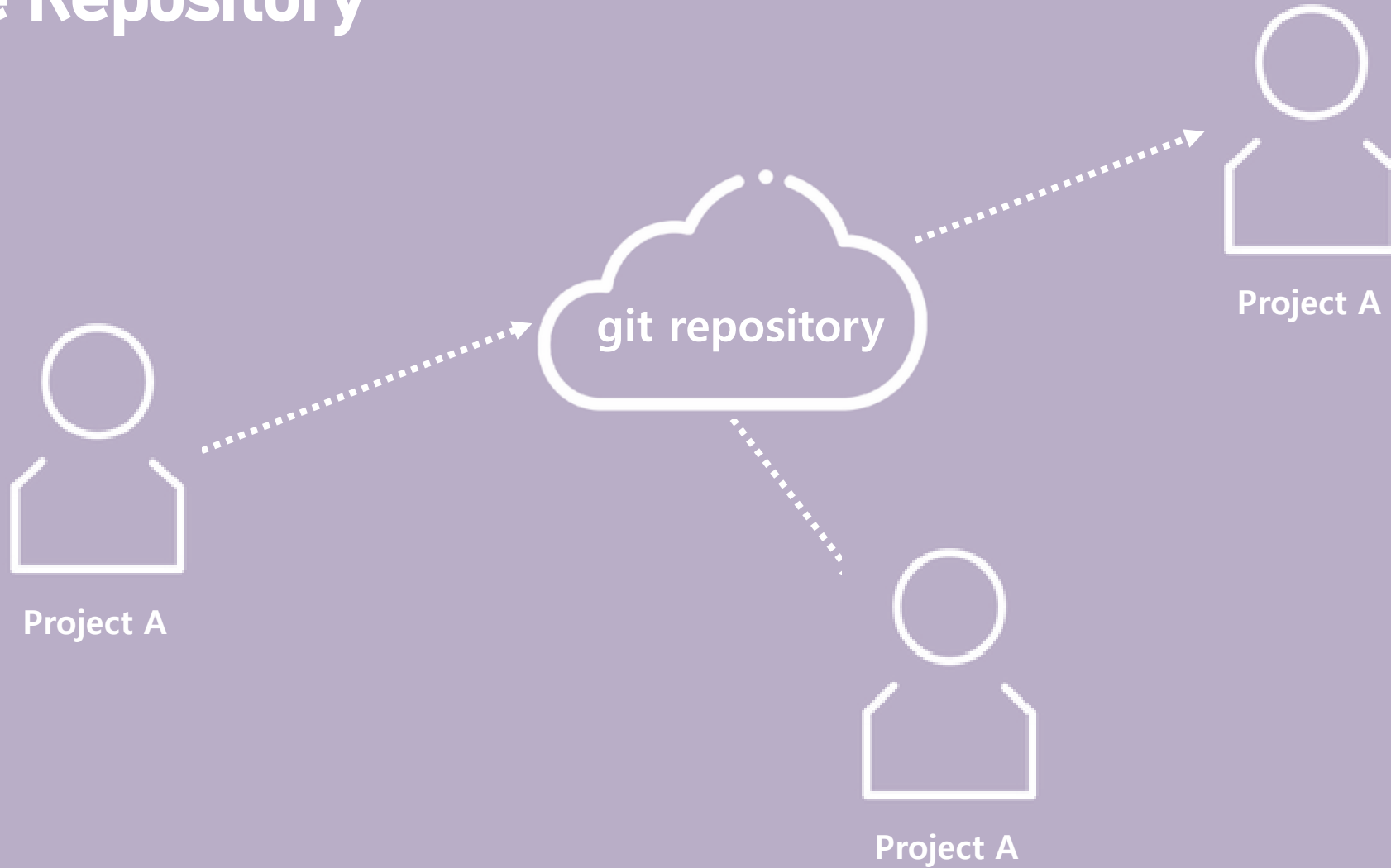
*Coffee Break*





# Cooperative Work

# Remote Repository





# Add Remote

## Repository (1/2)

Name	git-tutorial
Description	Git Tutorial Repository
Visibility level	Private

- Create New Remote Repository

New Project · GitLab

https://gitlab.com/projects/new

GitLab Projects Groups Activity Milestones Snippets Search or jump to...

Projects

### New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

To only use CI/CD features for an external repository, choose **CI/CD for external repo**.

**Tip:** You can also create a project from the command line. [Show command](#)

Blank project

Create from template

Import project

CI/CD for external repo

Project path

https://gitlab.com/ hyeonsig

Project name

git-tutorial

Want to house several dependent projects under the same namespace? [Create a group](#)

Project description (optional)

Git Tutorial Repository

Visibility Level ?

☒ Private

Project access must be granted explicitly to each user.

☐ Internal

The project can be accessed by any logged in user.

☐ Public

The project can be accessed without any authentication.

☐ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

---

## Add Remote

### Repository (2/2)

- Adding remote repository

```
$ git remote add origin <repo-address>
```

---

## Exercise

- 다음 과정을 진행하시오.
  - 새로운 Remote Repository를 생성하세요.
  - 로컬 환경에 Remote Repository를 등록하세요.

---

## Sync Remote Repository

- Sync

```
$ git pull origin master
```

- Push

```
$ git push origin master
```

# Exercise

---

## Preparation

- 팀장과 팀원을 만드시오.

팀장 1인

팀원 2인으로 구성

팀장은 Remote Repository에 자신의 팀원을 등록하세요.

---

## Clone Remote Repository

- Clone

```
$ git clone <repo-address> <location>
```

---

## Exercise

- 다음 과정을 진행하시오.
  - 팀장은 Remote Repository와 현재 상태를 동일하게 만드세요.
  - 팀원은 새로운 Local Repository를 생성하고, Remote Repository와 동일한 상태를 만드세요.



---

## Cooperative Work

### Demo (1/6)

- GitTutorial

팀원 1이 작업합니다. 다음 요구사항을 만족한 후 커밋하세요.

#### Modify Contents

- 3개의 목록(리스트)을 추가
- 세미나: 자료구조, 자바, 파이썬

---

## Cooperative Work

### Demo (2/6)

- 팀원 1은 작업한 내용을 PUSH 하세요.

- PUSH

Remote Repository에 내용을 보내는 것

- PULL

Remote Repository 내용을 가져오는 것

---

## Cooperative Work Demo (3/6)

- Checking Remote Repository
- Sync

Pull Contents from Remote Repository

---

## Cooperative Work

### Demo (4/6)

- New requirement

팀원 1이 작업합니다. 다음 요구사항을 만족한 후 커밋하세요.

### Modify Contents

- 3개의 목록(리스트)을 추가
- 프로젝트: Project A, Project B, Project C

---

## Cooperative Work

### Demo (5/6)

- Original

팀원 2가 작업합니다. 다음 요구사항을 만족한 후 커밋하세요.

#### Modify Contents

- 세미나 목록의 최상단에 1개의 목록(리스트)을 추가
- 개체지향

---

## Cooperative Work

### Demo (6/6)

- Original

Commit 이후 Push에서 오류 발생

- Commit은 항상 Remote Repository의 최신 버전과 싱크를 이룬 상태에서 수행해야 함

# Attention!!

Recommend Sequence

Pull -> Work -> Commit -> Pull -> Push

---

## Conflict (1/5)

- 고의적으로 에러를 발생하겠습니다.

### Modify Contents

- 마지막 목록에 다음 내용을 추가합니다.
- 팀원 1은 Original에서 본문의 마지막 줄에 Project D 입력
- 팀원 2는 Clone에서 본문의 마지막 줄에 Project E 입력



---

## Conflict (2/5)

- Clone

Pull -> Work -> Commit -> Pull -> Push

- Original

Pull -> Work -> Commit -> Pull -> Push

---

## Conflict (3/5)

- Conflict 발생

## Conflict (4/5)

- GitTutorial Conflict 발생

```
<<<<<<< HEAD
    <li>프로젝트 E</li>
=====
    <li>프로젝트 D</li>
>>>>>>> 프로젝트 추가
```

```
...
<h4>프로젝트</h4>
<ul>
    <li>Project A</li>
    <li>Project B</li>
    <li>Project C</li>
    <li>Project D</li>
    <li>Project E</li>
</ul>
...
```

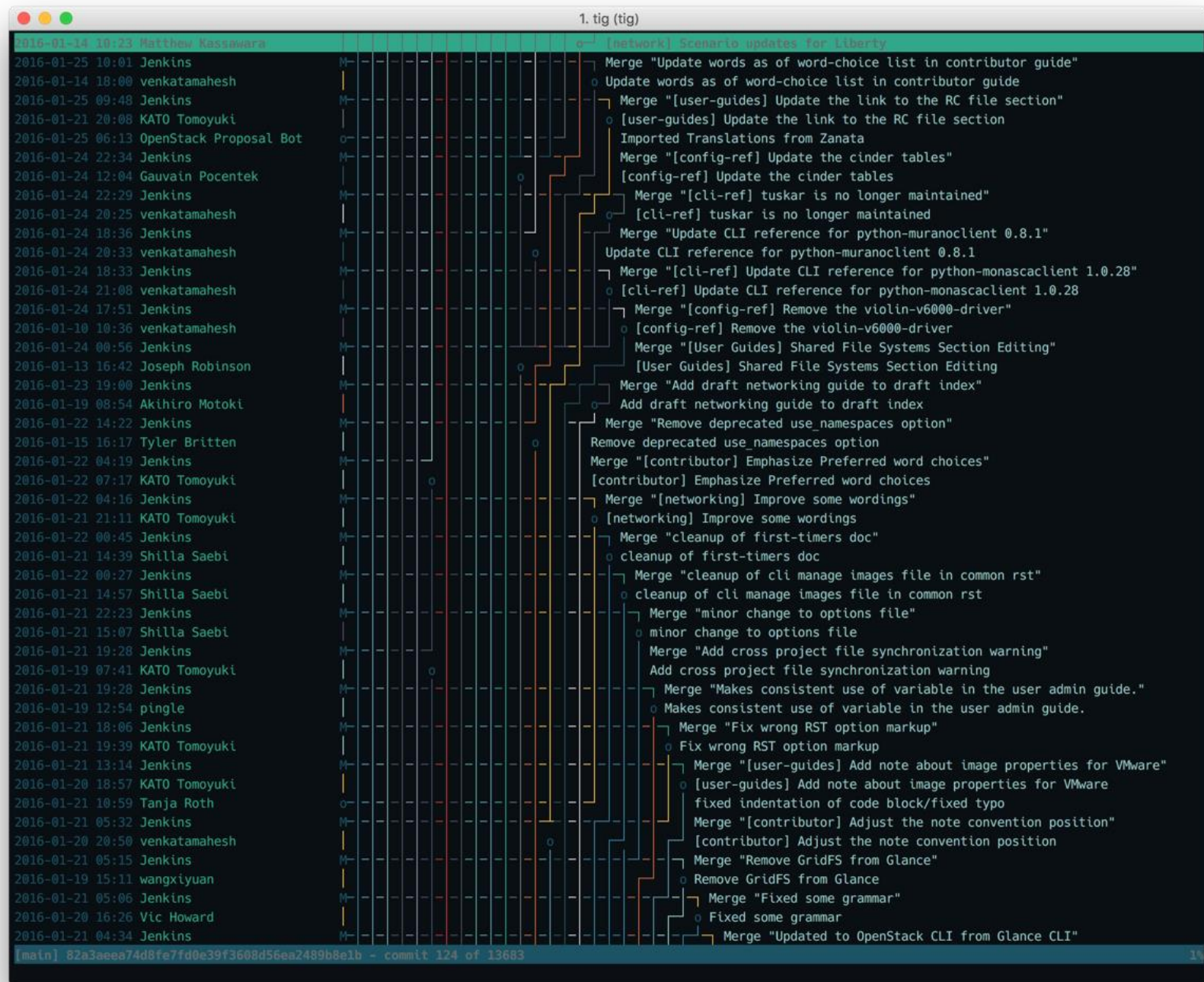
---

## Conflict (5/5)

- Conflict 해결 후 다시 커밋 -> Pair Programming

# Tig

<https://ujuc.github.io/2016/02/10/tig-manual/>



```
1. tig (tig)
2016-01-14 10:23 Matthew Kassawara [network] Scenario updates for library
2016-01-25 10:01 Jenkins Merge "Update words as of word-choice list in contributor guide"
2016-01-14 18:00 venkatamahesh Update words as of word-choice list in contributor guide
2016-01-25 09:48 Jenkins Merge "[user-guides] Update the link to the RC file section"
2016-01-21 20:08 KATO Tomoyuki [user-guides] Update the link to the RC file section
2016-01-25 06:13 OpenStack Proposal Bot Imported Translations from Zanata
2016-01-24 22:34 Jenkins Merge "[config-ref] Update the cinder tables"
2016-01-24 12:04 Gauvain Pocentek [config-ref] Update the cinder tables
2016-01-24 22:29 Jenkins Merge "[cli-ref] tuskar is no longer maintained"
2016-01-24 20:25 venkatamahesh [cli-ref] tuskar is no longer maintained
2016-01-24 18:36 Jenkins Merge "Update CLI reference for python-muranoclient 0.8.1"
2016-01-24 20:33 venkatamahesh Update CLI reference for python-muranoclient 0.8.1
2016-01-24 18:33 Jenkins Merge "[cli-ref] Update CLI reference for python-monascaclient 1.0.28"
2016-01-24 21:08 venkatamahesh [cli-ref] Update CLI reference for python-monascaclient 1.0.28
2016-01-24 17:51 Jenkins Merge "[config-ref] Remove the violin-v6000-driver"
2016-01-10 10:36 venkatamahesh [config-ref] Remove the violin-v6000-driver
2016-01-24 00:56 Jenkins Merge "[User Guides] Shared File Systems Section Editing"
2016-01-13 16:42 Joseph Robinson [User Guides] Shared File Systems Section Editing
2016-01-23 19:00 Jenkins Merge "Add draft networking guide to draft index"
2016-01-19 08:54 Akihiro Motoki Add draft networking guide to draft index
2016-01-22 14:22 Jenkins Merge "Remove deprecated use_namespaces option"
2016-01-15 16:17 Tyler Britten Remove deprecated use_namespaces option
2016-01-22 04:19 Jenkins Merge "[contributor] Emphasize Preferred word choices"
2016-01-22 07:17 KATO Tomoyuki [contributor] Emphasize Preferred word choices
2016-01-22 04:16 Jenkins Merge "[networking] Improve some wordings"
2016-01-21 21:11 KATO Tomoyuki [networking] Improve some wordings
2016-01-22 00:45 Jenkins Merge "cleanup of first-timers doc"
2016-01-21 14:39 Shilla Saebi cleanup of first-timers doc
2016-01-22 00:27 Jenkins Merge "cleanup of cli manage images file in common rst"
2016-01-21 14:57 Shilla Saebi cleanup of cli manage images file in common rst
2016-01-21 22:23 Jenkins Merge "minor change to options file"
2016-01-21 15:07 Shilla Saebi minor change to options file
2016-01-21 19:28 Jenkins Merge "Add cross project file synchronization warning"
2016-01-19 07:41 KATO Tomoyuki Add cross project file synchronization warning
2016-01-21 19:28 Jenkins Merge "Makes consistent use of variable in the user admin guide."
2016-01-19 12:54 pingle Makes consistent use of variable in the user admin guide.
2016-01-21 18:06 Jenkins Merge "Fix wrong RST option markup"
2016-01-21 19:39 KATO Tomoyuki Fix wrong RST option markup
2016-01-21 13:14 Jenkins Merge "[user-guides] Add note about image properties for VMware"
2016-01-20 18:57 KATO Tomoyuki [user-guides] Add note about image properties for VMware
2016-01-21 10:59 Tanja Roth fixed indentation of code block/fixed typo
2016-01-21 05:32 Jenkins Merge "[contributor] Adjust the note convention position"
2016-01-20 20:50 venkatamahesh [contributor] Adjust the note convention position
2016-01-21 05:15 Jenkins Merge "Remove GridFS from Glance"
2016-01-19 15:11 wangxiyuan Remove GridFS from Glance
2016-01-21 05:06 Jenkins Merge "Fixed some grammar"
2016-01-20 16:26 Vic Howard Fixed some grammar
2016-01-21 04:34 Jenkins Merge "Updated to OpenStack CLI from Glance CLI"
[main] 82a3aee74d8fe7fd0e39f3608d56ea2489b8e1b - commit 124 of 13683
```



# QnA

hyeonsig@wisoft.io

<https://hyeonsig.tistory.com>

---

## References

- The Clean Coder: A Code of Conduct for Professional Programmers
  - Robert C. Martin
- Git Manual
  - <http://git-scm.com/book/ko>
- Code School - Try git
  - <https://try.github.io/>

---

## References

- Learn git branching
  - <http://pcottle.github.io/learnGitBranching/>
- Atlassian git tutorial
  - <https://www.atlassian.com/git/tutorial>



***Thank You***