Version Control System

GIT (깃)

한국전기연구원/전기의료기기 연구센터

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

INDEX

연자 소개

- **의료 영상 처리 알고리즘** : 실시간 초음파/CT/MRI 영상 정합 (삼성전자 의료기기 S-fusion)
- **음성 신호 처리 알고리즘** : 보청기 알고리즘 개발 프레임워크 개발
- **인공지능 알고리즘 연구 (현재)**:
  - 무선전력전송용 안테나 인공지능 기술
  - 의료영상 망막 산소포화도 측정 인공지능 기술

서울대학교
전기공학부, 의용생체공학
(1999년~2011년)

삼성전자
종합기술원
(2011년~2015년)

한국전기연구원
첨단의료기기연구본부
(2015년~현재)

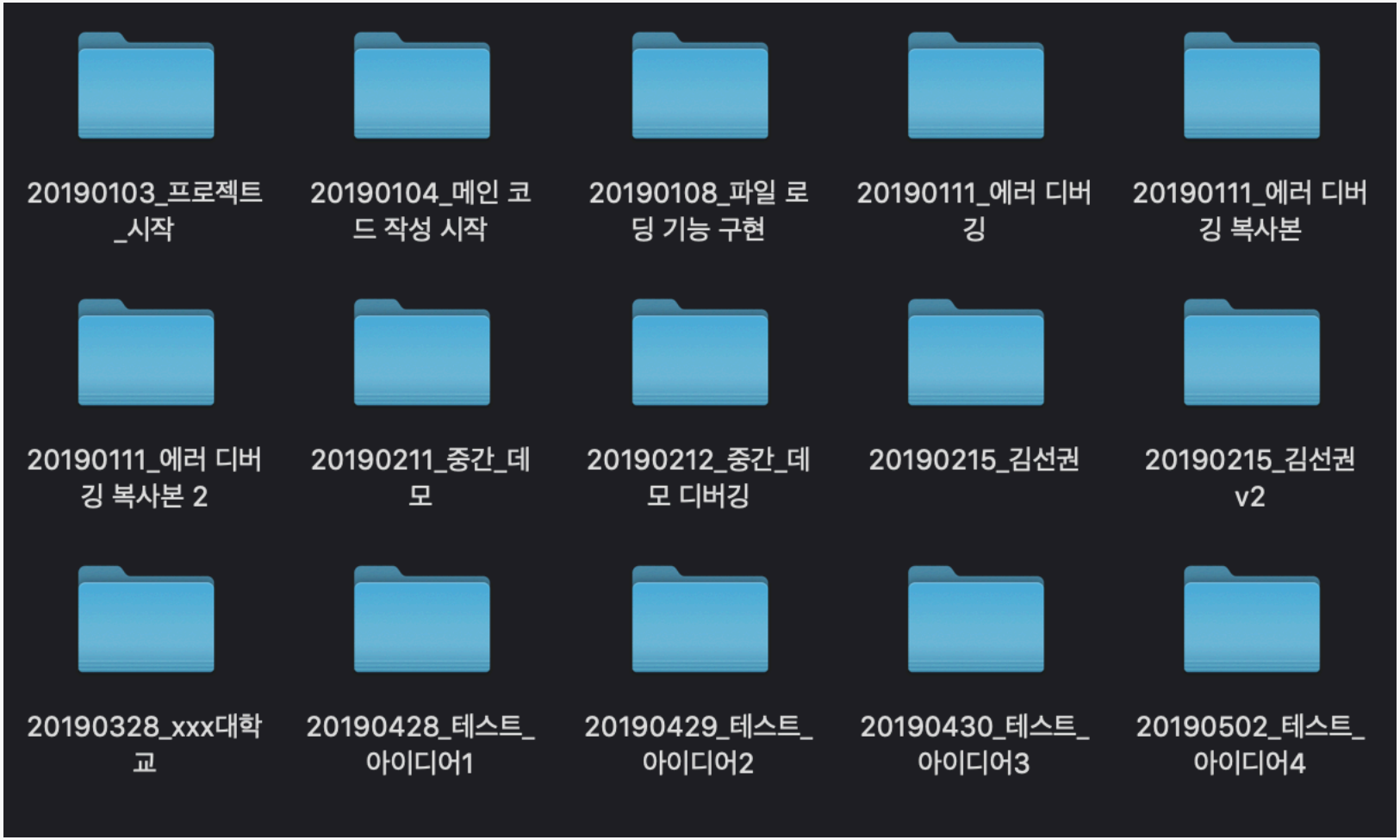
Git이 탄생한

1. 배경 설명

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

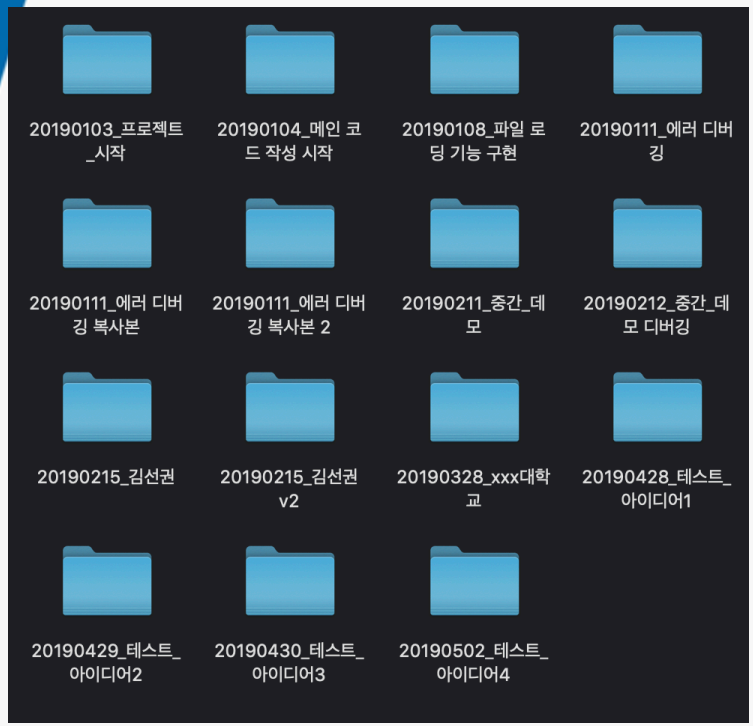
1. 프로그램을 개발하다 보면...

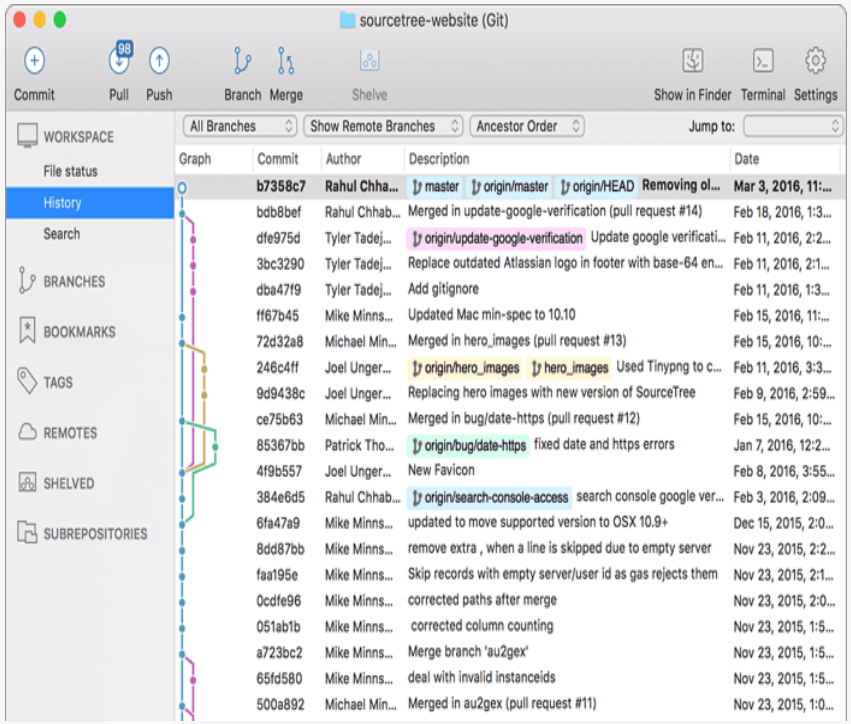


1. 프로그램을 개발하다 보면...

1. 프로그램을 개발하다 보면...

코드 관리를 도와줄 뭔가가 필요하다.





Version Control System

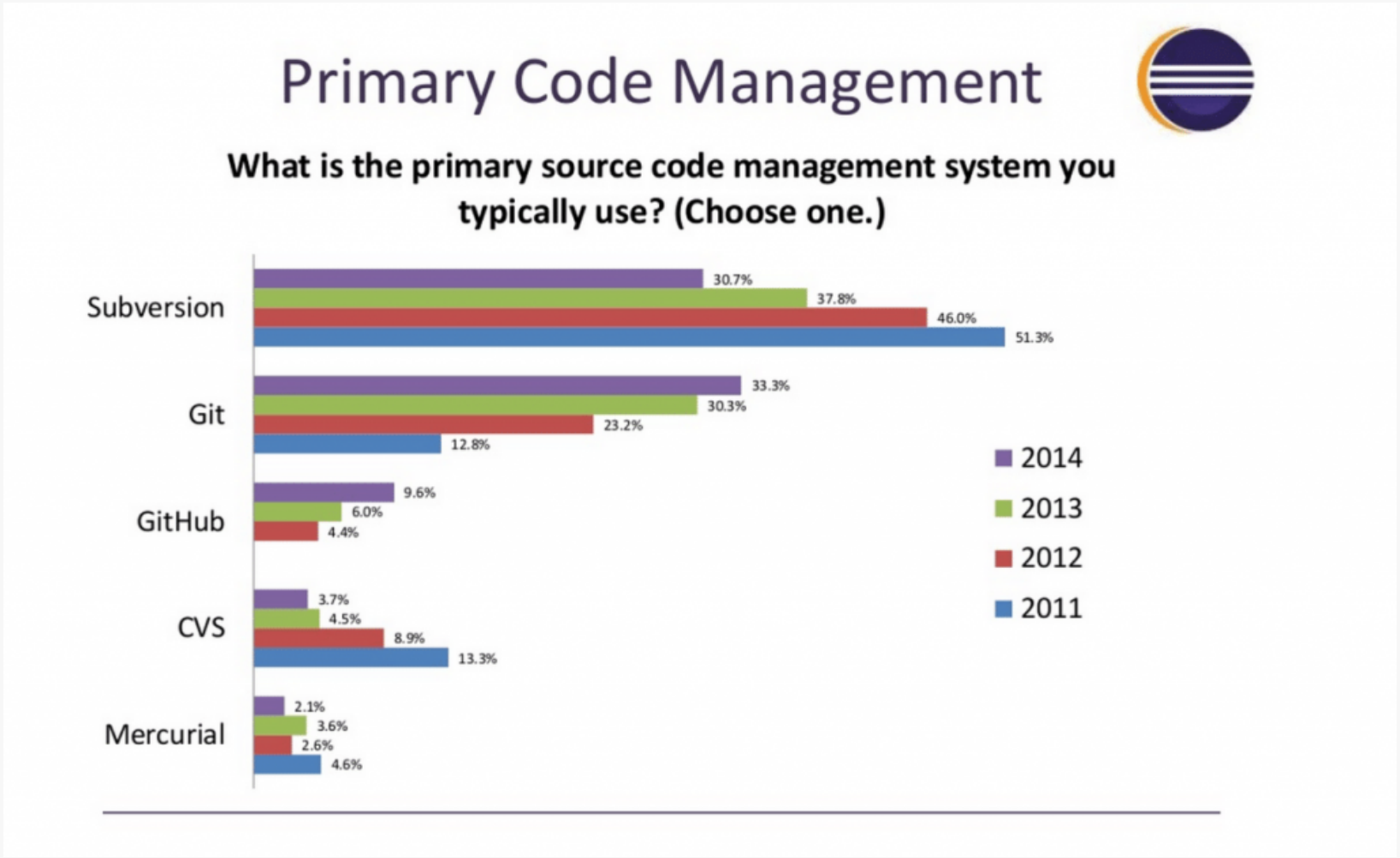
Local → Centralized → Distributed

- RCS

- CVS
- Subversion
- Perforce

- **Git**
- Mercurial
- Bazaar

1. 배경설명 – Git의 현재



1. 배경설명 – Git의 현재

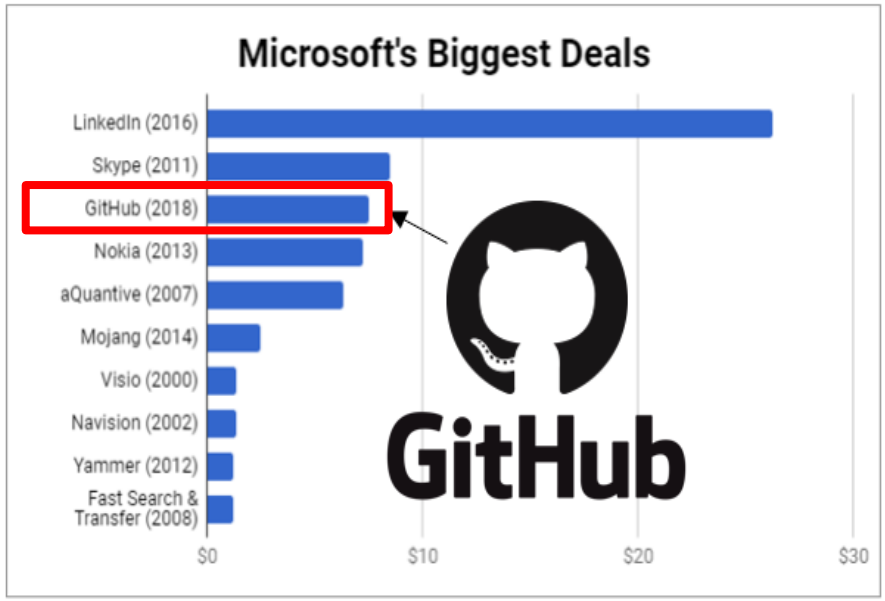
KERI

BEWARE OF OCTOCLIPPY —

Microsoft snaps up GitHub for $7.5 billion

Biggest open source hosting service joins biggest proprietary software company.

PETER BRIGHT - 6/4/2018, 10:56 PM



In billions. Source: Microsoft SEC filings, GeekWire reporting GEEKWIRE.COM

GitHub

31 million
Users

57 million
repositories

Largest host of source codes

1. 배경설명 – Git의 창시자

리누스 토발즈

리눅스 개발에 소스 코드 관리 프로그램인 BitKeeper를 사용하고 있었음

➡ BitKeeper가 무상 제공이 종료됨에 따라 리누스 토발즈가 직접 소스 코드 관리 프로그램(Git)을 직접 개발하여 사용함

➡ 당시 소스 컨트롤 프로그램들의 단점을 보완(극복)할 수 있도록 개발됨.

1. 배경설명 – Git의 창시자

Linus Torvalds
torvalds

Follow

Linux Foundation
Portland, OR

Block or report user

Organizations

Overview | Repositories 6 | Projects 0 | Stars 2 | Followers 93.6k | Following 0

Popular repositories

linux ● C ★ 43.3k ⑂ 16.5k (2017.03)
Linux kernel source tree
● C ★ 74.8k ⑂ 26.2k (2019.05)

uemacs
Random version of microemacs with my private modificatons
● C ★ 380 ⑂ 57

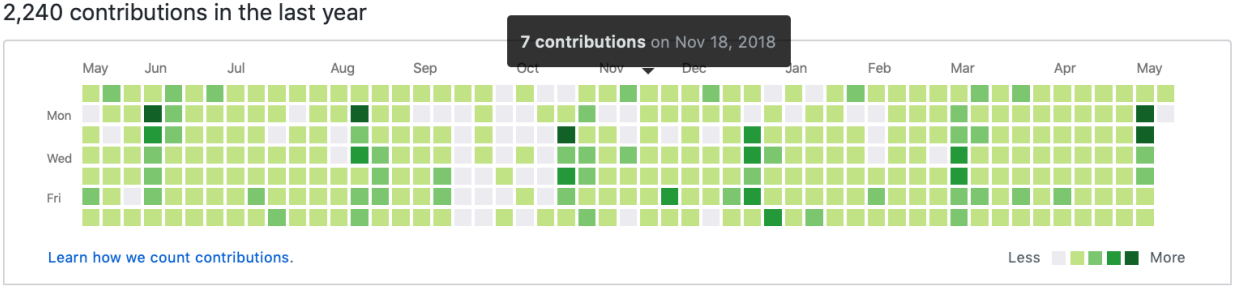
test-tlb
Stupid memory latency and TLB tester
● C ★ 212 ⑂ 69

pesconvert
Brother PES file converter
● C ★ 112 ⑂ 12

subsurface-for-dirk
Forked from Subsurface-divelog/subsurface
Do not use - the real upstream is Subsurface-divelog/subsurface
● C++ ★ 85 ⑂ 31

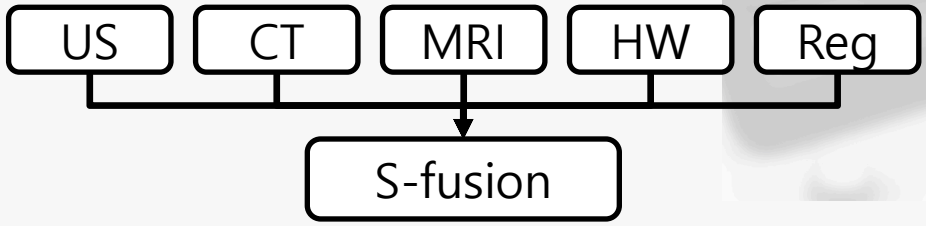
libdc-for-dirk
Forked from Subsurface-divelog/libdc
Only use for syncing with Dirk, don't use for anything else
● C ★ 43 ⑂ 18

2,240 contributions in the last year



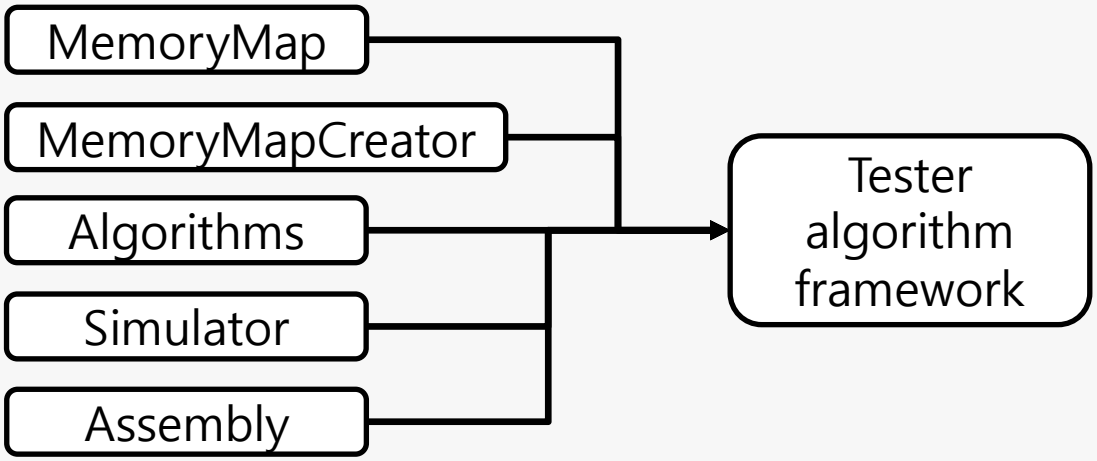
삼성 메디슨 S-Fusion 개발

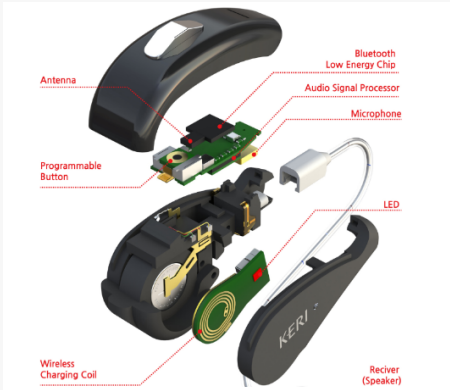
- 초음파, CT, MRI 실시간 정합
- 중재 시술용

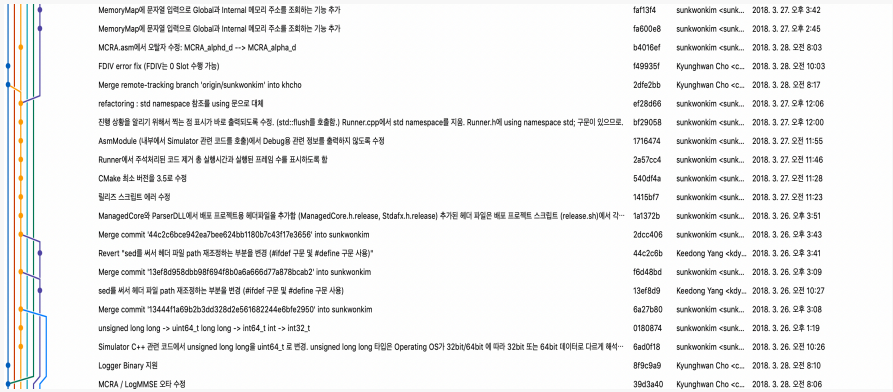


1. 배경설명 – Git을 이용한 연구/개발

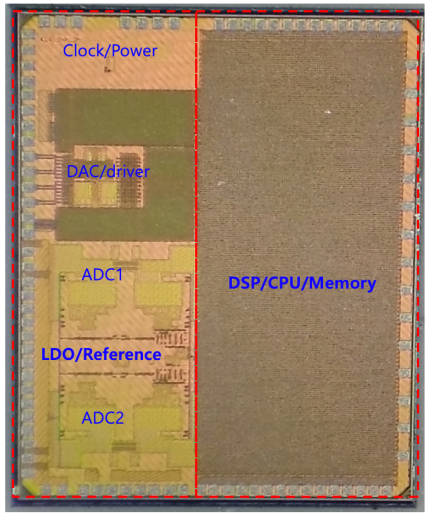
보청기 알고리즘 연구/개발 알고리즘 테스터 프레임워크







MemoryMap에 문자열 입력으로 Global과 Internal 메모리 주소를 조회하는 기능 추가	faf13f4	sunkwonkim <sunk...	2018. 3. 27. 오후 3:42
MemoryMap에 문자열 입력으로 Global과 Internal 메모리 주소를 조회하는 기능 추가	fa600e8	sunkwonkim <sunk...	2018. 3. 27. 오후 2:45
MCRA.asm에서 오탈자 수정: MCRA_alphd_d --> MCRA_alpha_d	b4016ef	sunkwonkim <sunk...	2018. 3. 28. 오전 8:03
FDIV error fix (FDIV는 0 Slot 수행 가능)	f49935f	Kyunghwan Cho <c...	2018. 3. 28. 오전 10:03
Merge remote-tracking branch 'origin/sunkwonkim' into khcho	2dfe2bb	Kyunghwan Cho <c...	2018. 3. 28. 오전 8:17
refactoring : std namespace 참조를 using 문으로 대체	ef28d66	sunkwonkim <sunk...	2018. 3. 27. 오후 12:06
진행 상황을 알리기 위해서 찍는 점 표시가 바로 출력되도록 수정. (std::flush를 호출함.) Runner.cpp에서 std namespace를 지움. Runner.h에 using namespace std; 구문이 있으므로.	bf29058	sunkwonkim <sunk...	2018. 3. 27. 오후 12:00
AsmModule (내부에서 Simulator 관련 코드를 호출)에서 Debug용 관련 정보를 출력하지 않도록 수정	1716474	sunkwonkim <sunk...	2018. 3. 27. 오전 11:55
Runner에서 주석처리된 코드 제거 총 실행시간과 실행된 프레임 수를 표시하도록 함	2a57cc4	sunkwonkim <sunk...	2018. 3. 27. 오전 11:46
CMake 최소 버전을 3.5로 수정	540df4a	sunkwonkim <sunk...	2018. 3. 27. 오전 11:28
릴리즈 스크립트 에러 수정	1415bf7	sunkwonkim <sunk...	2018. 3. 27. 오전 11:23
ManagedCore와 ParserDLL에서 배포 프로젝트용 헤더파일을 추가함 (ManagedCore.h.release, Stdafx.h.release) 추가된 헤더 파일은 배포 프로젝트 스크립트 (release.sh)에서 자...	1a1372b	sunkwonkim <sunk...	2018. 3. 26. 오후 3:51
Merge commit '44c2c8bce942ea7bee624bb1180b7c43f17e3656' into sunkwonkim	2dcc406	sunkwonkim <sunk...	2018. 3. 26. 오후 3:43
Revert "sed를 써서 헤더 파일 path 재조정하는 부분을 변경 (#ifdef 구문 및 #define 구문 사용)"	44c2c6b	Keedong Yang <kdy...	2018. 3. 26. 오후 3:41
Merge commit '13ef8d958dbb98f694f8b0a6a666d77a878bcab2' into sunkwonkim	f6d48bd	sunkwonkim <sunk...	2018. 3. 26. 오후 3:09
sed를 써서 헤더 파일 path 재조정하는 부분을 변경 (#ifdef 구문 및 #define 구문 사용)	13ef8d9	Keedong Yang <kdy...	2018. 3. 26. 오전 10:27
Merge commit '134441fa69b2b3dd328d2e561682244e6bfe2950' into sunkwonkim	6a27b80	sunkwonkim <sunk...	2018. 3. 26. 오후 3:08
unsigned long long -> uint64_t long long -> int64_t int -> int32_t	0180874	sunkwonkim <sunk...	2018. 3. 26. 오후 1:19
Simulator C++ 관련 코드에서 unsigned long long을 uint64_t 로 변경. unsigned long long 타입은 Operating OS가 32bit/64bit 에 따라 32bit 또는 64bit 데이터로 다르게 해석...	6ad0f18	sunkwonkim <sunk...	2018. 3. 26. 오전 10:26
Logger Binary 지원	8f9c9a9	Kyunghwan Cho <c...	2018. 3. 28. 오전 8:10
MCRA / LogMMSE 오타 수정	39d3a40	Kyunghwan Cho <c...	2018. 3. 28. 오전 8:06



필요성

2. GIT이 왜 좋은가?

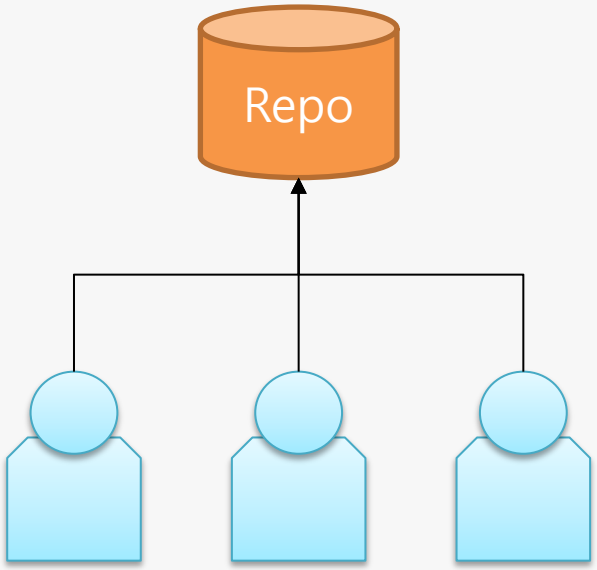
KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

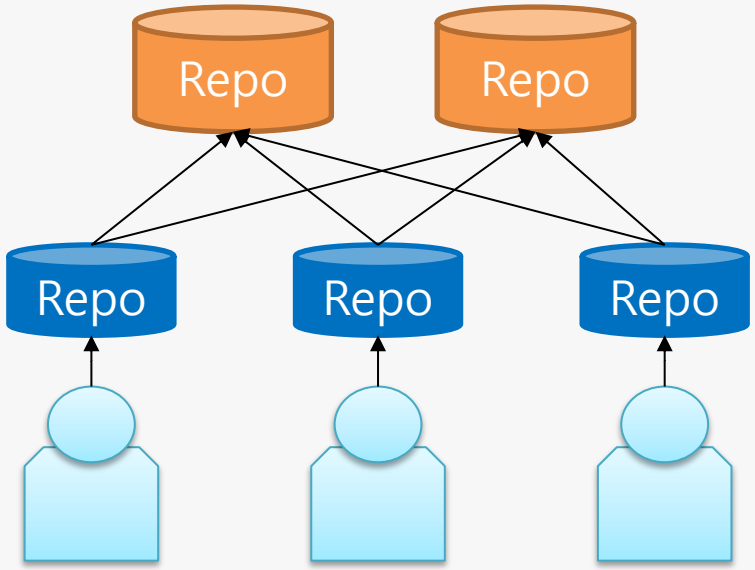
2. 기존의 툴과 비교

	CVS	SVN	Git
개발자	The CVS Team	콜랩넷	리누스 토발즈
종류	중앙집중식	중앙집중식	분산 버전 처리
운영체제	유닉스, 윈도우	크로스 플랫폼	크로스 플랫폼

CVS / SVN

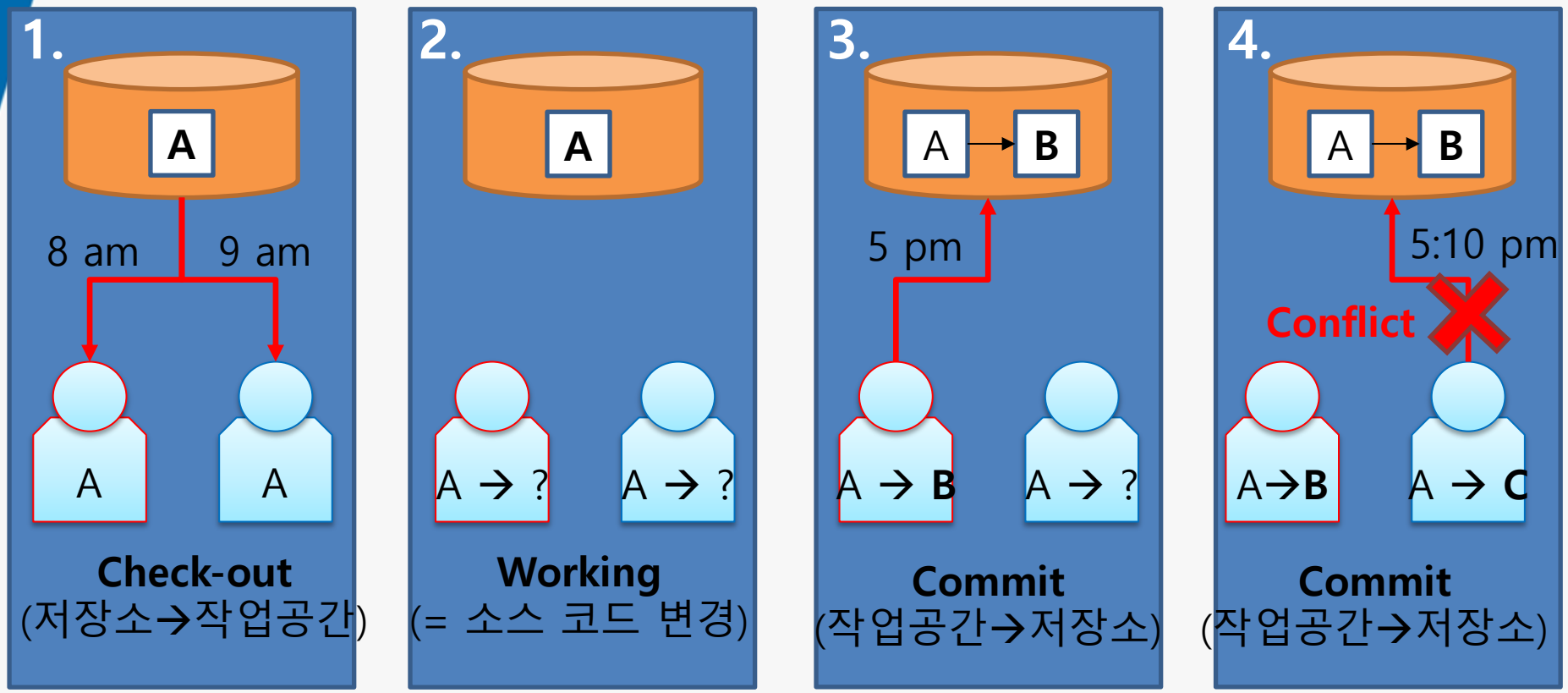


Git



2. 기존의 툴과 비교

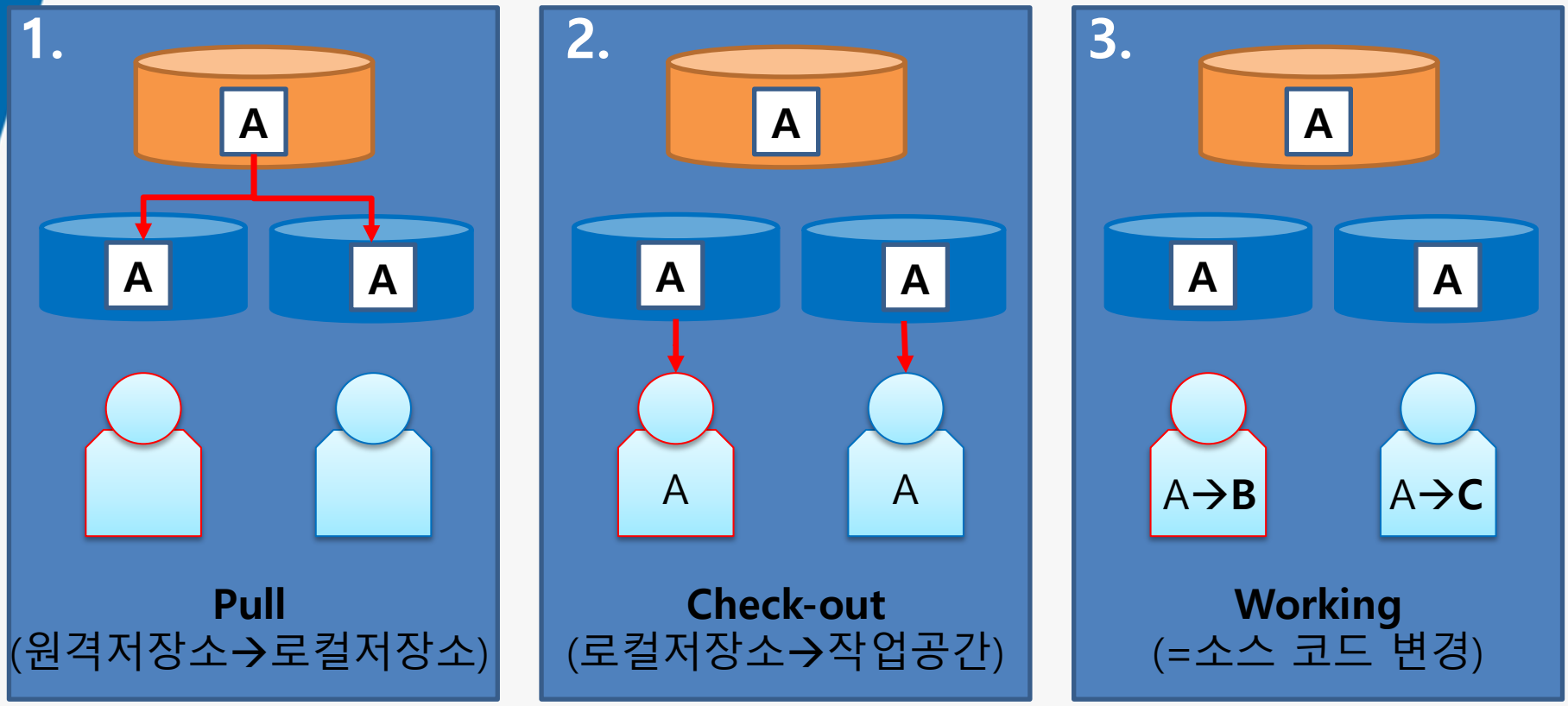
CVS / SVN



2. 기존의 툴과 비교

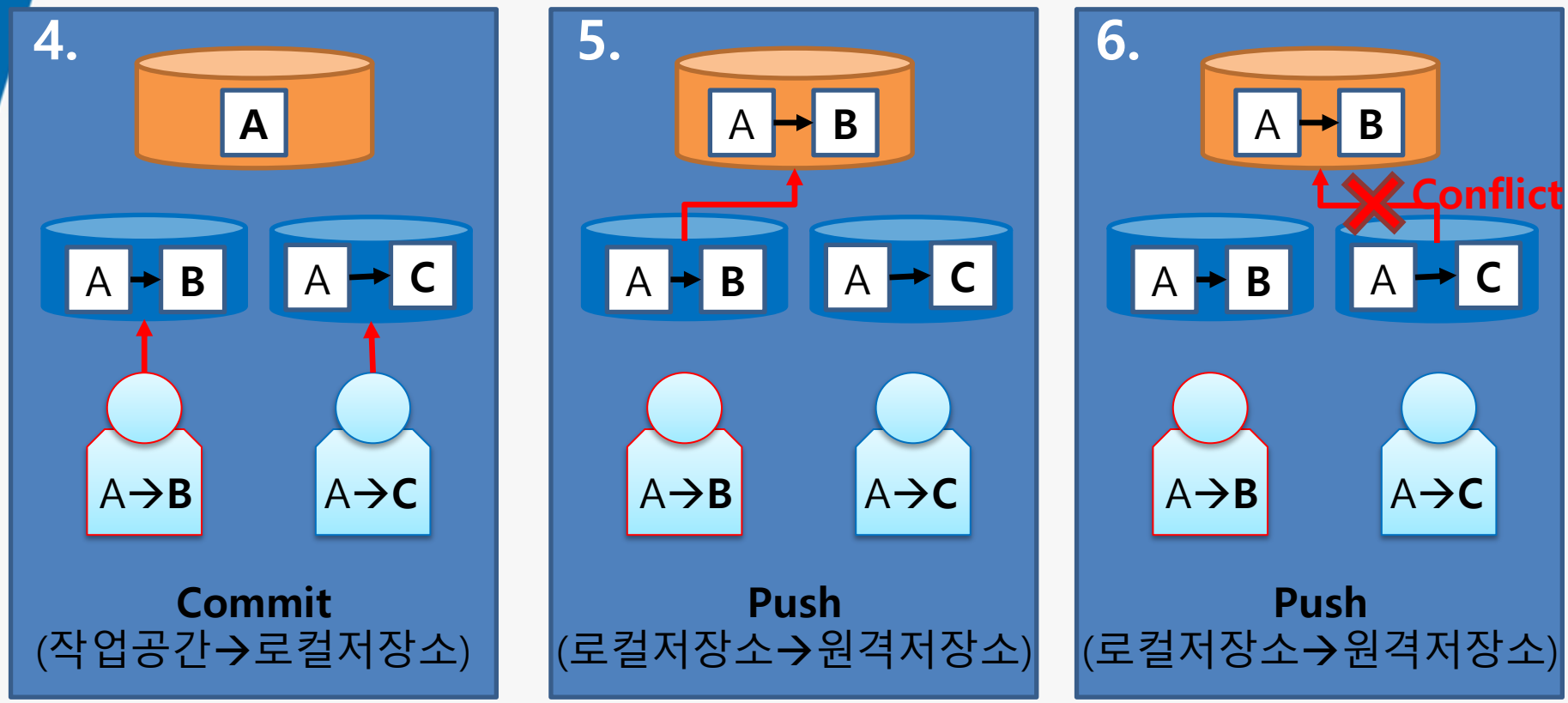
KERI

Git



2. 기존의 툴과 비교

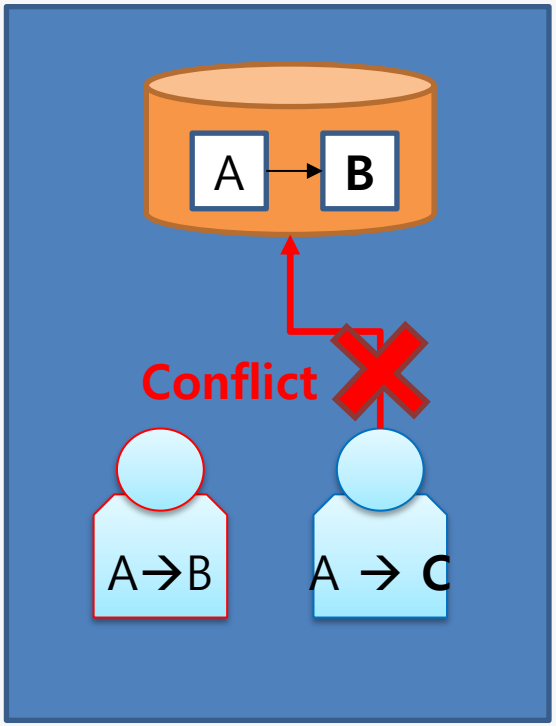
Git



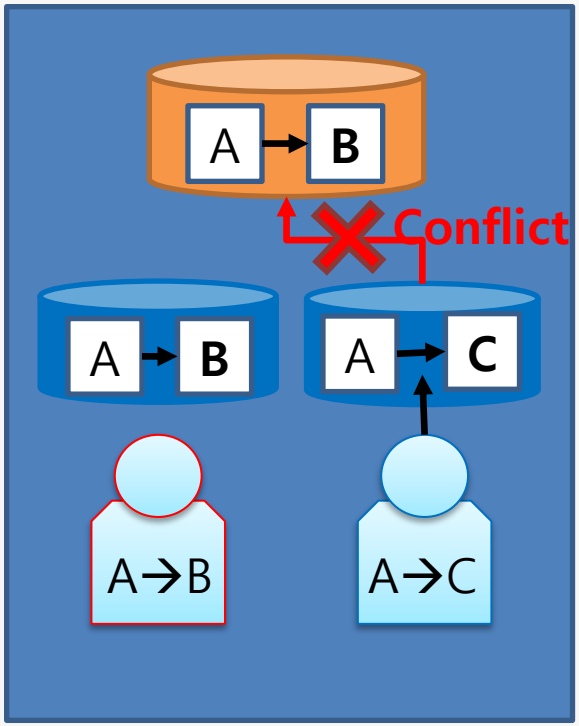
2. 장점 1

KERI

CVS / SVN



Git

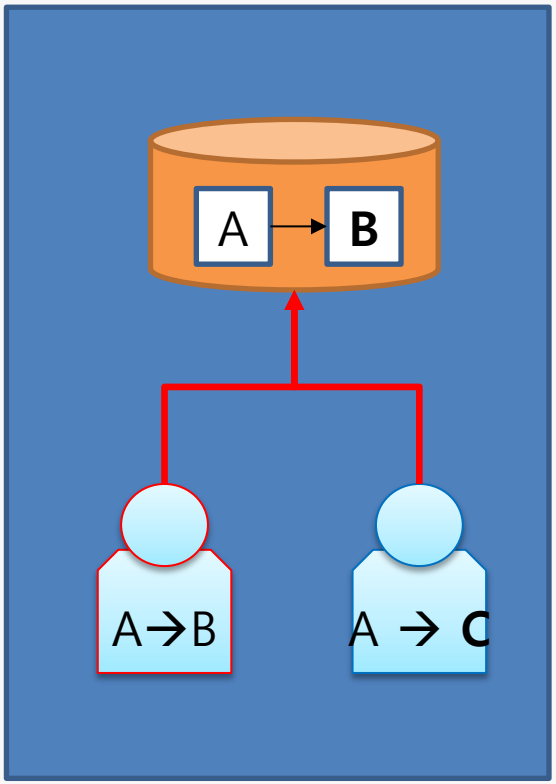


장점 1. Conflict을 해결하지 않으면 작업 내용을 버려야 한다. → 버리지 않아도 된다.

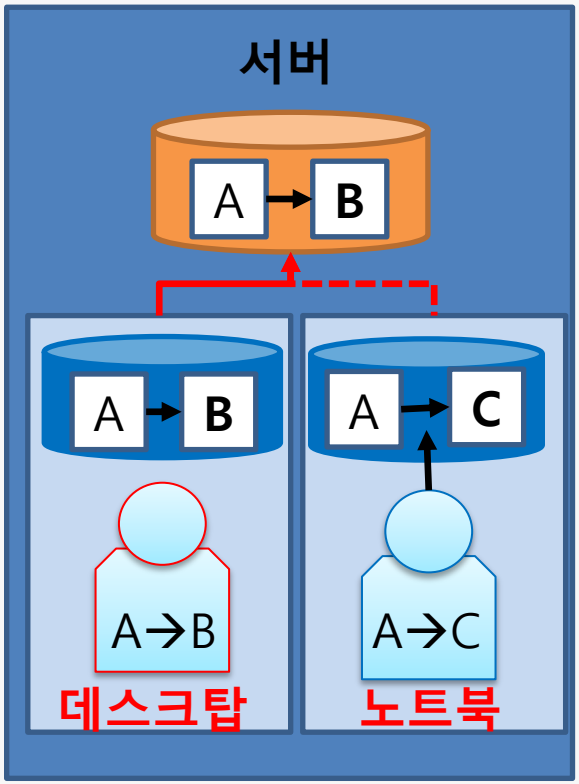
2. 장점 2

KERI

CVS / SVN



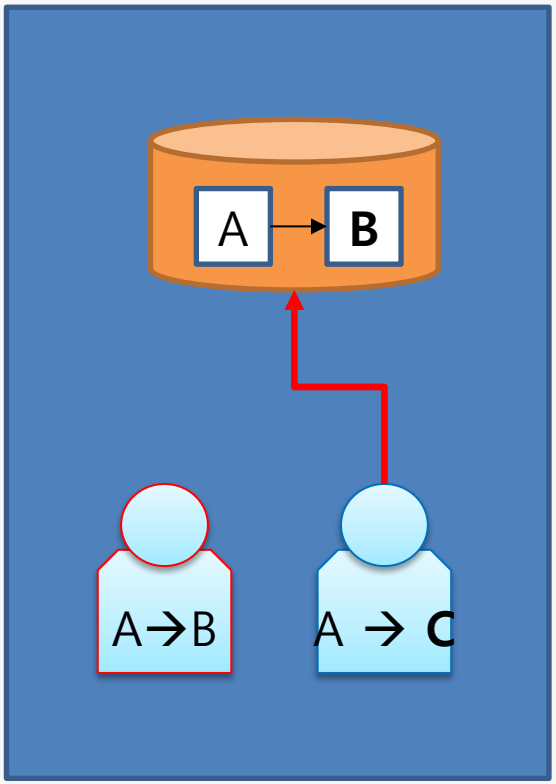
Git



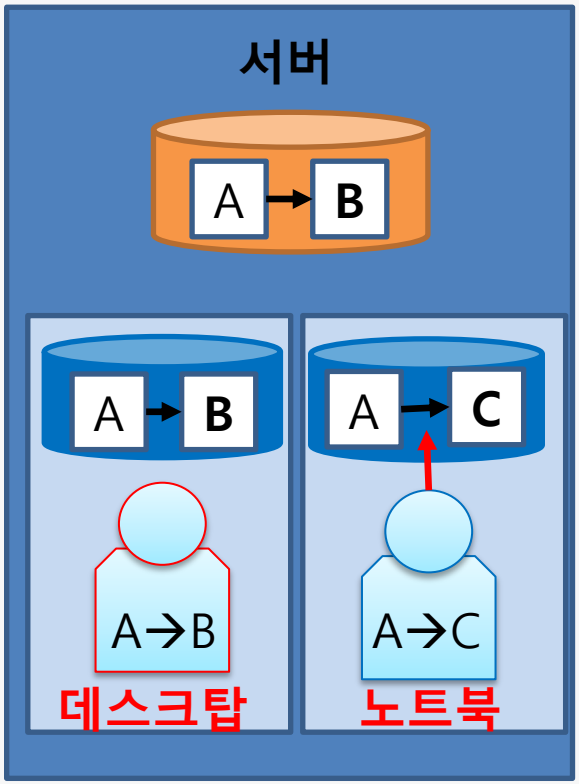
장점 2. 항상 저장소에 연결되어야 한다. → 오프라인에서도 작업이 가능하다.

2. 장점 3

CVS / SVN



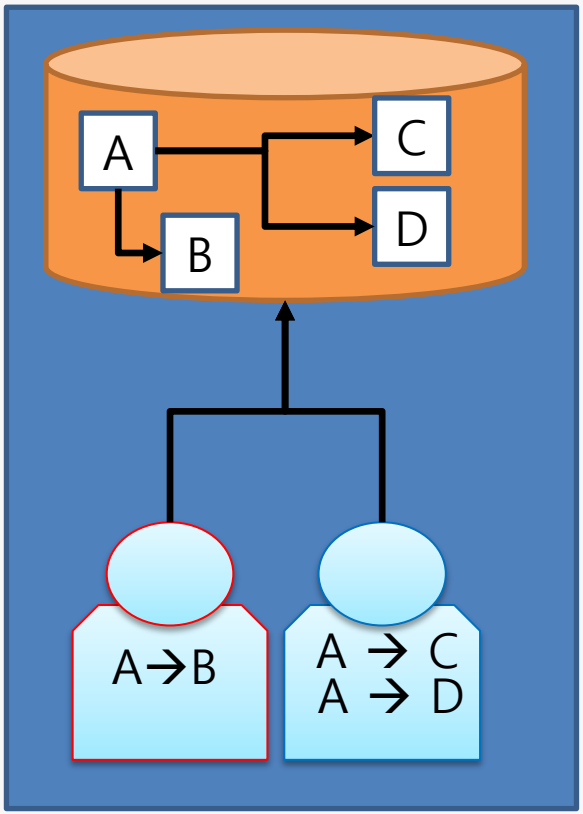
Git



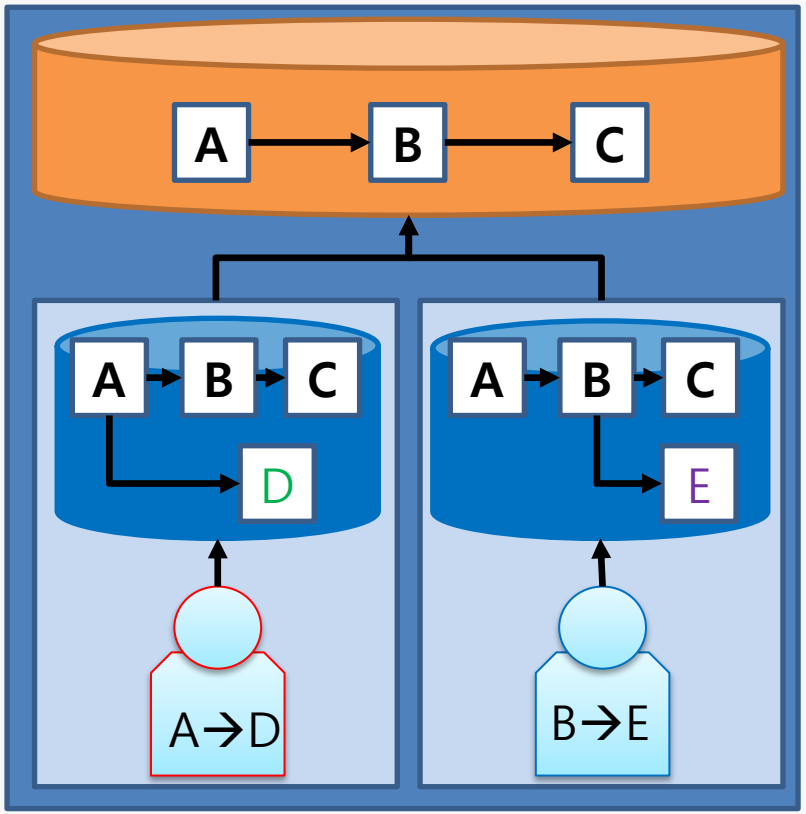
장점 3. 프로그램 규모와 인원이 늘어날수록 느리다 → 느려지지 않는다 (빠르다).

2. 장점 4

CVS / SVN



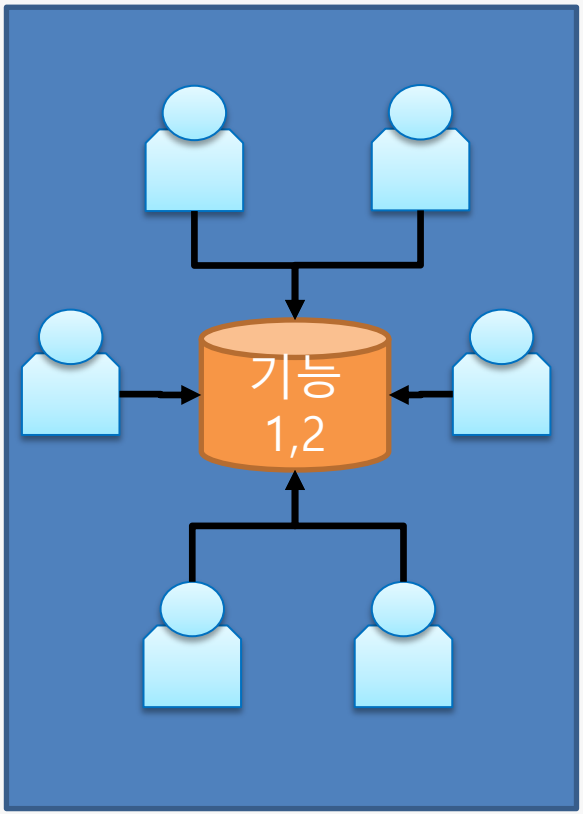
Git



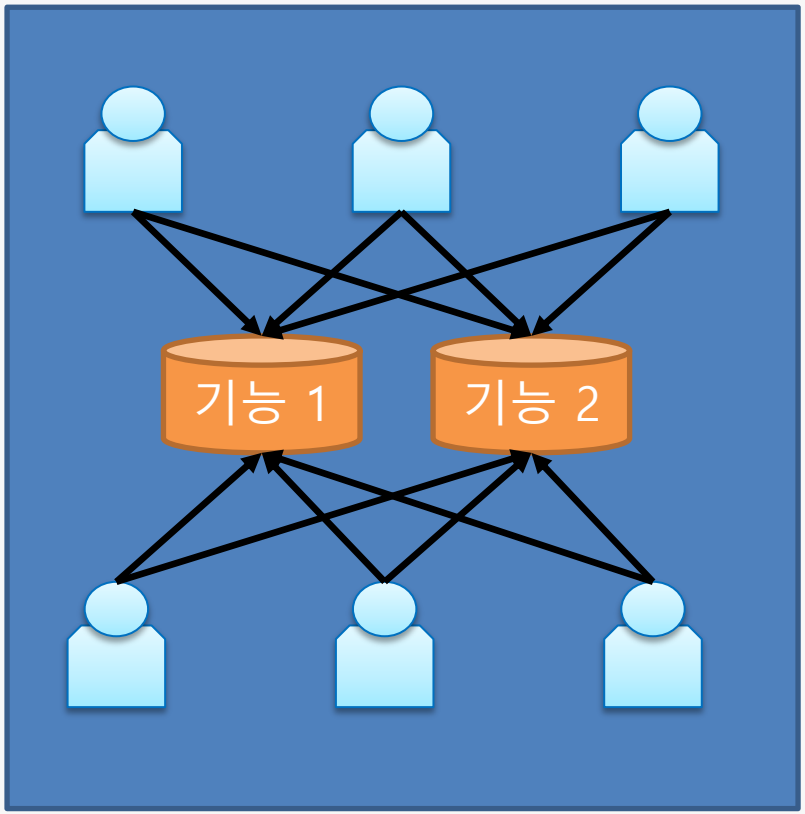
장점 4. 모든 작업 과정은 공개되어야 한다. → 여러 가지 아이디어를 테스트한 후 최종 안을 공개할 수 있다.

2. 장점 5

CVS / SVN



Git

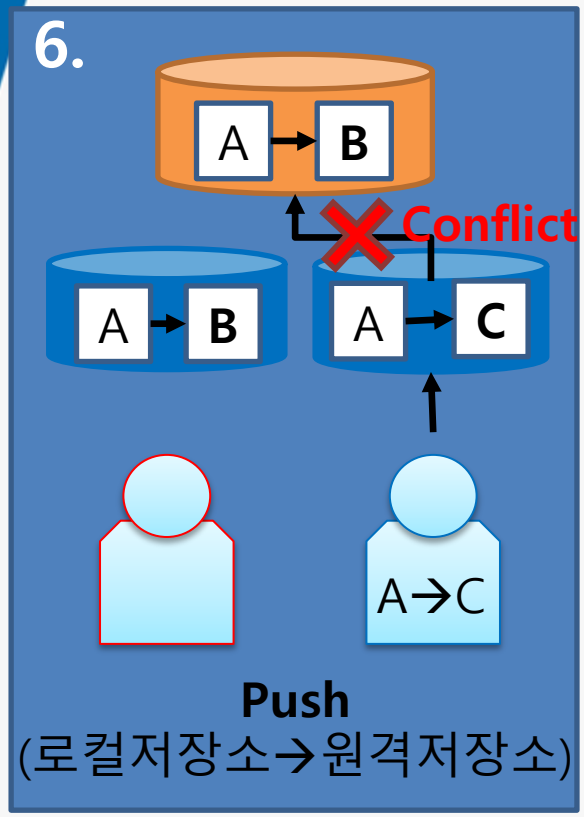


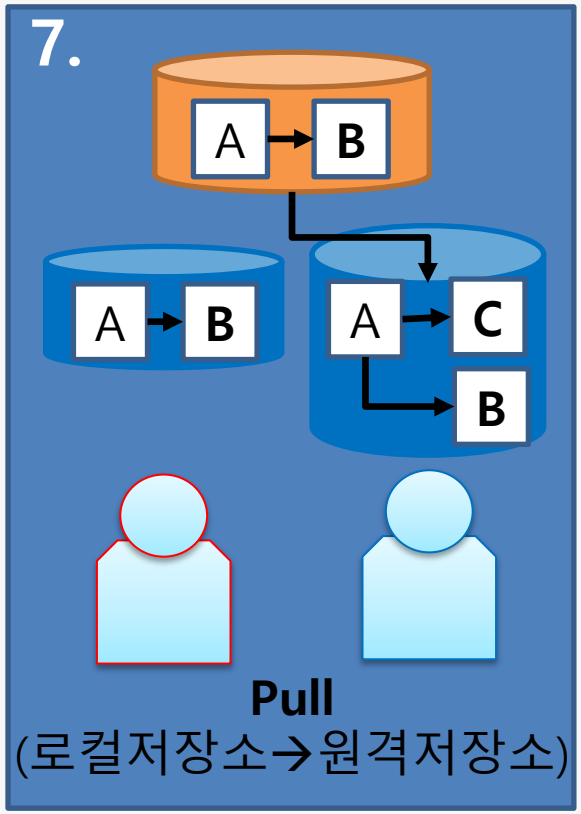
장점 5. 단일 저장소 → 다중 원격 저장소 (소스 코드의 분산 저장/관리가 가능)

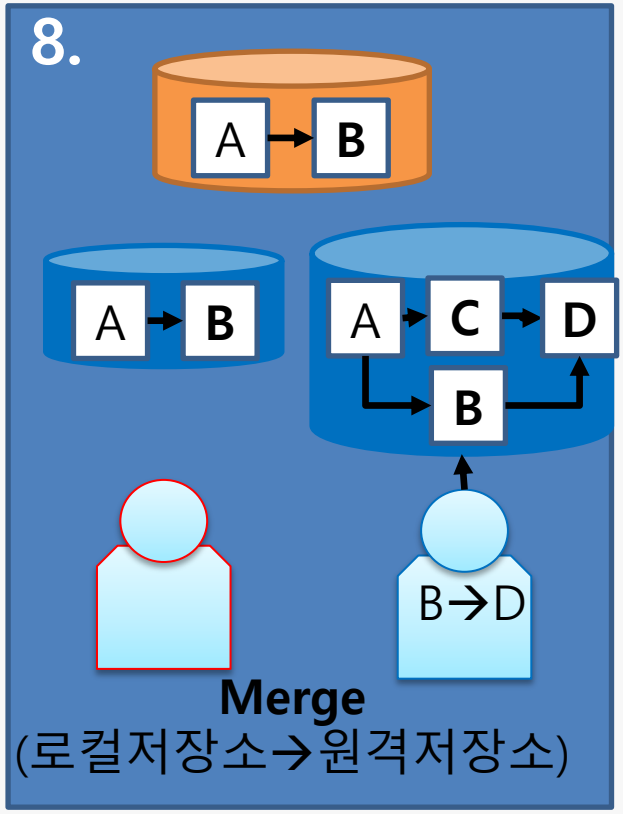
2. 기존의 툴과 비교

Git

* 다시 충돌(Conflict)이 발생했을 시점으로 돌아가 봅시다.

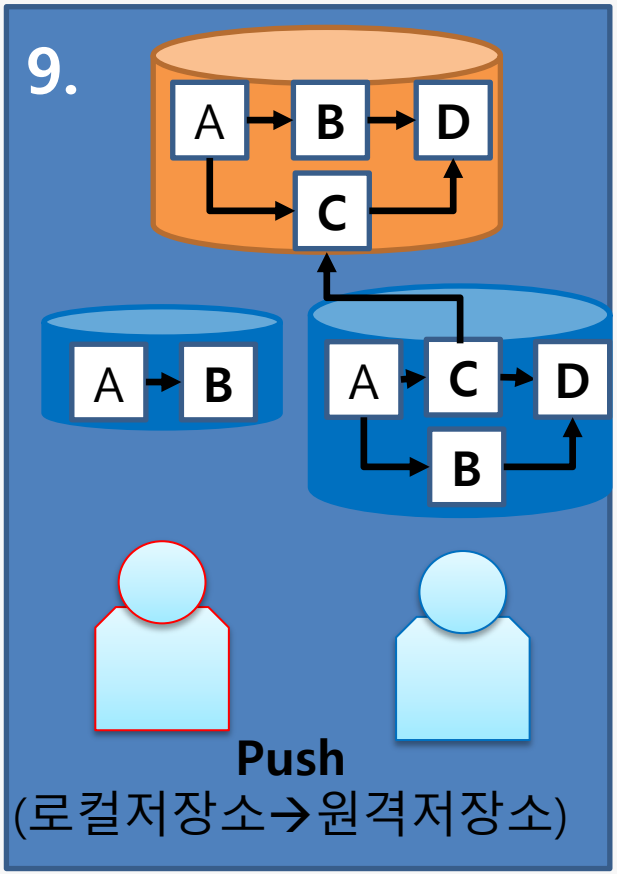


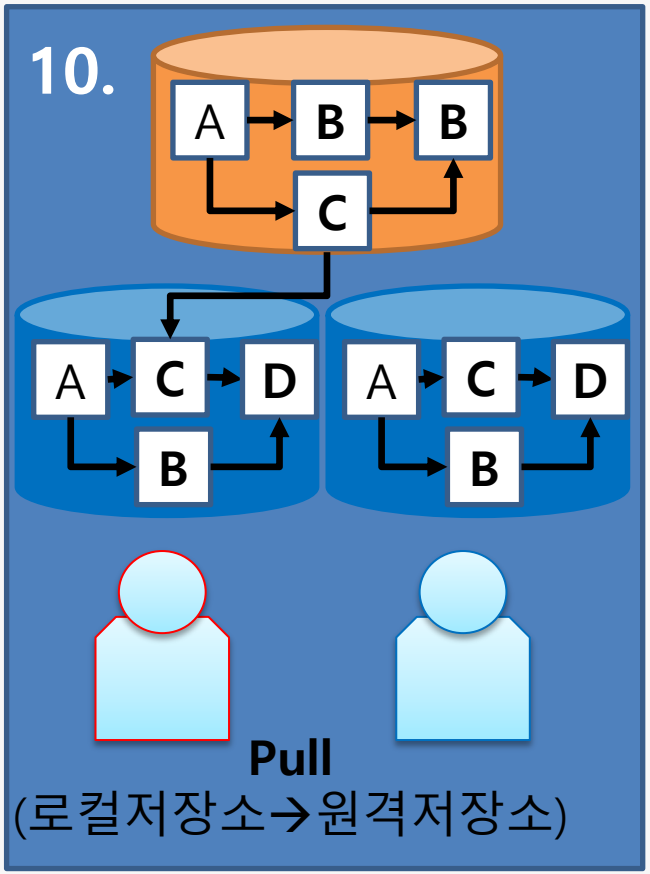




2. 기존의 툴과 비교

Git





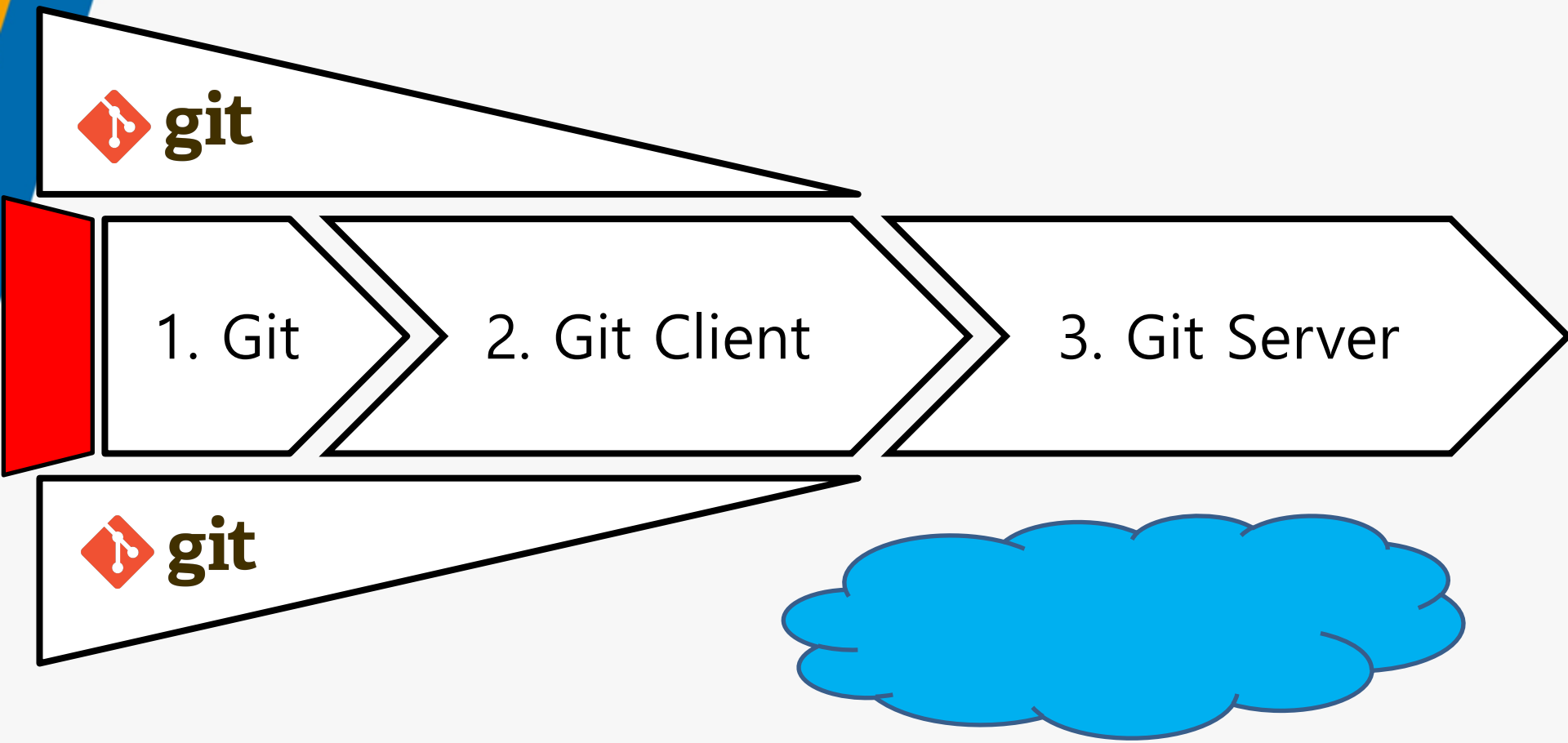
git

3. 시작하기

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

3. 시작하기



3. Git

1.1 Git 인스톨러를 이용한 설치

- 구글 검색 → 다운로드 → 설치
- https://git-scm.com

1.2 터미널에서 설치 (패키지 관리 툴 사용)

Ubuntu: `apt install git ⏎`

Red Hat: `yum install git ⏎`

Mac: `brew install git ⏎`

2. Git 초기 설정

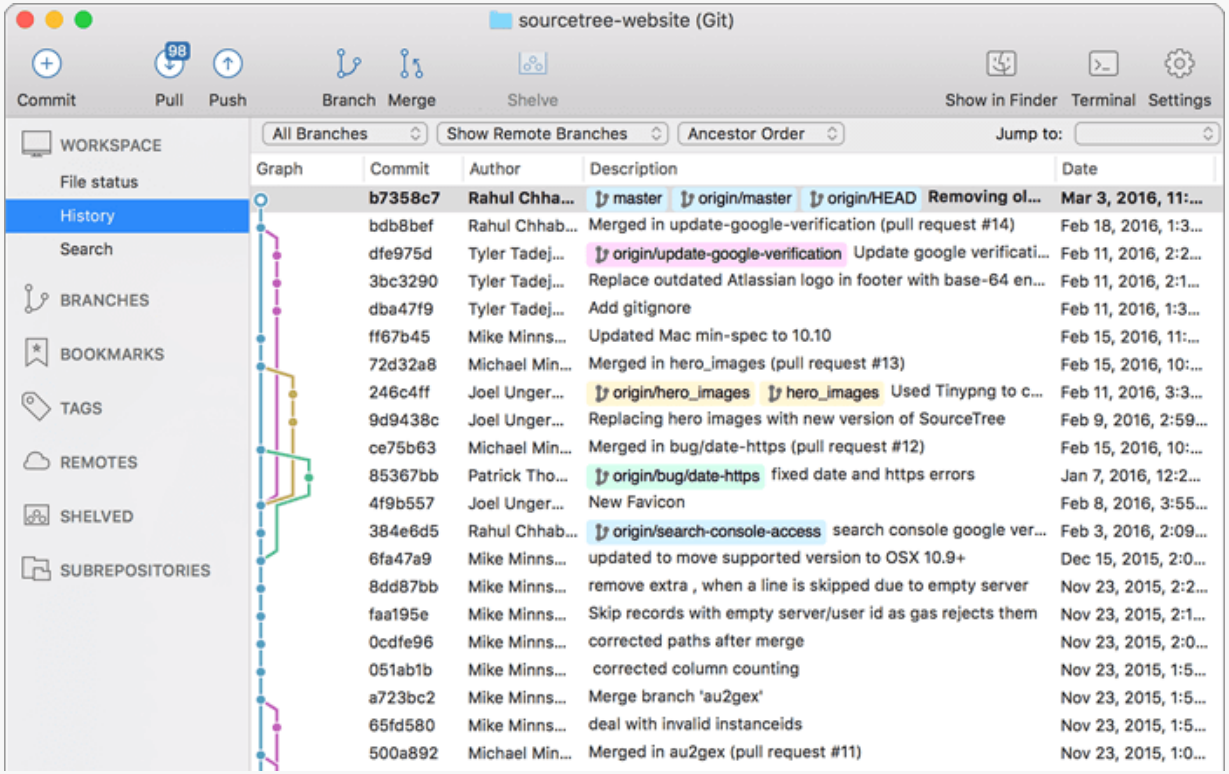
```
git config –global user.name "이름" ⏎
git config –global user.email "메일주소" ⏎
```

3. Git client

1. Sourcetree (Windows/Mac)

- https://www.sourcetreeapp.com/
- Completely free for any use

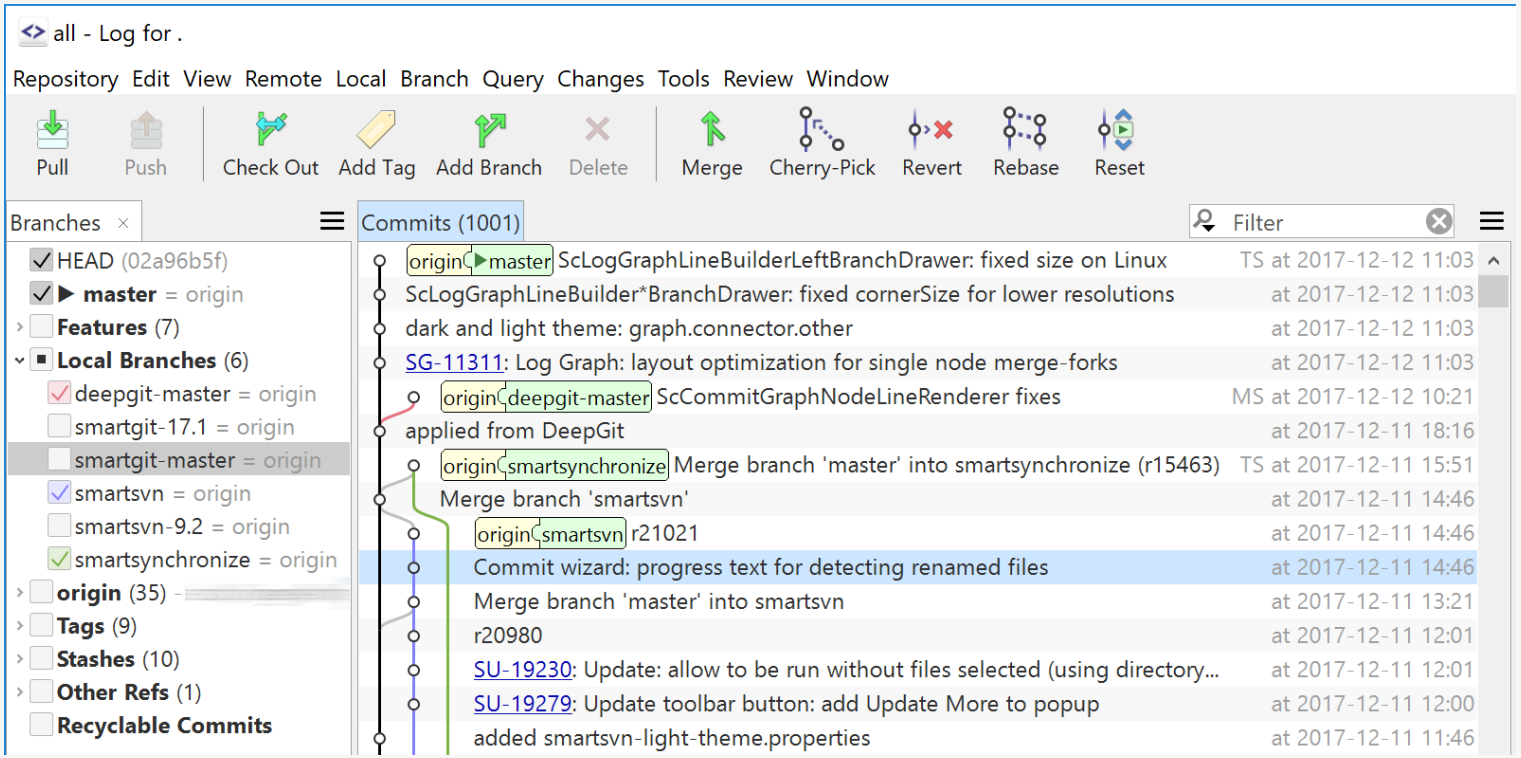
Sourcetree



3. Git client

2. Smart-git (Windows/Mac/Linux)

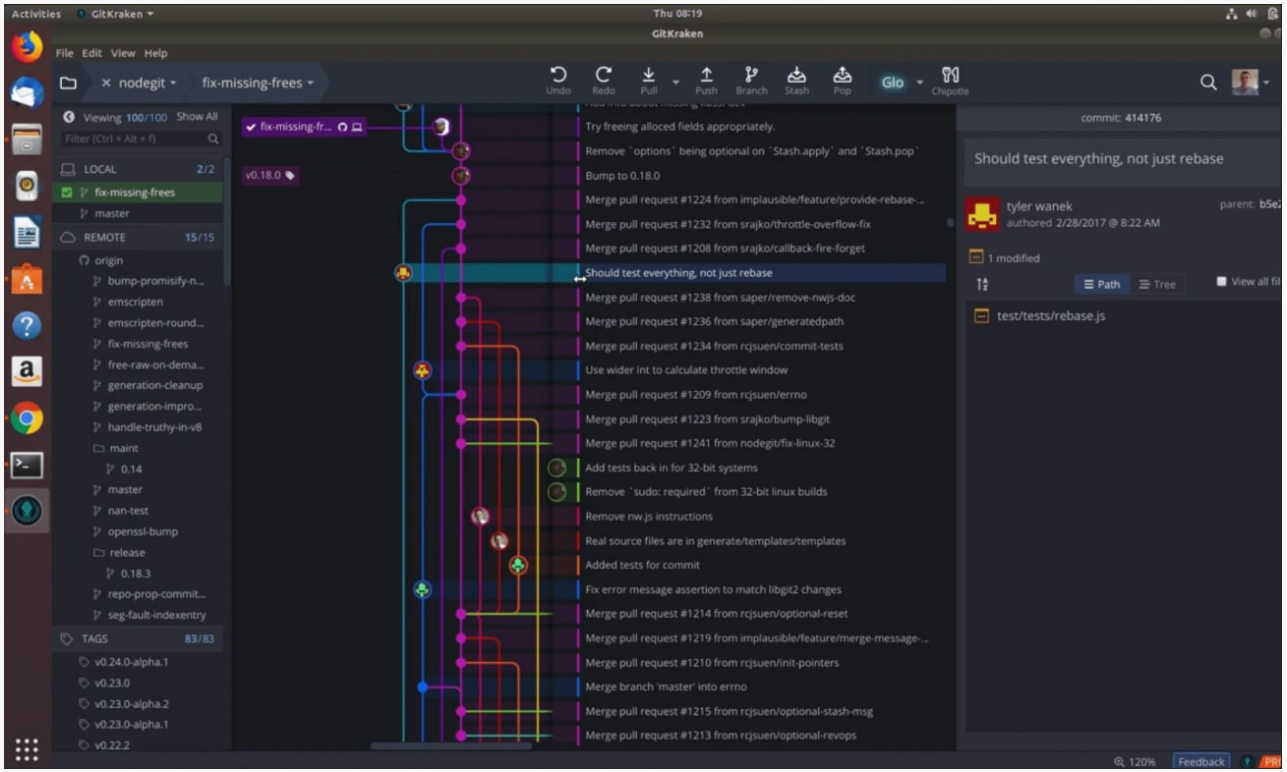
- https://www.syntevo.com/smartgit/
- Free for non-commercial use



3. Git client

3. Gitkraken (Windows/Mac/Linux)

- https://www.gitkraken.com/
- Free for non-commercial use

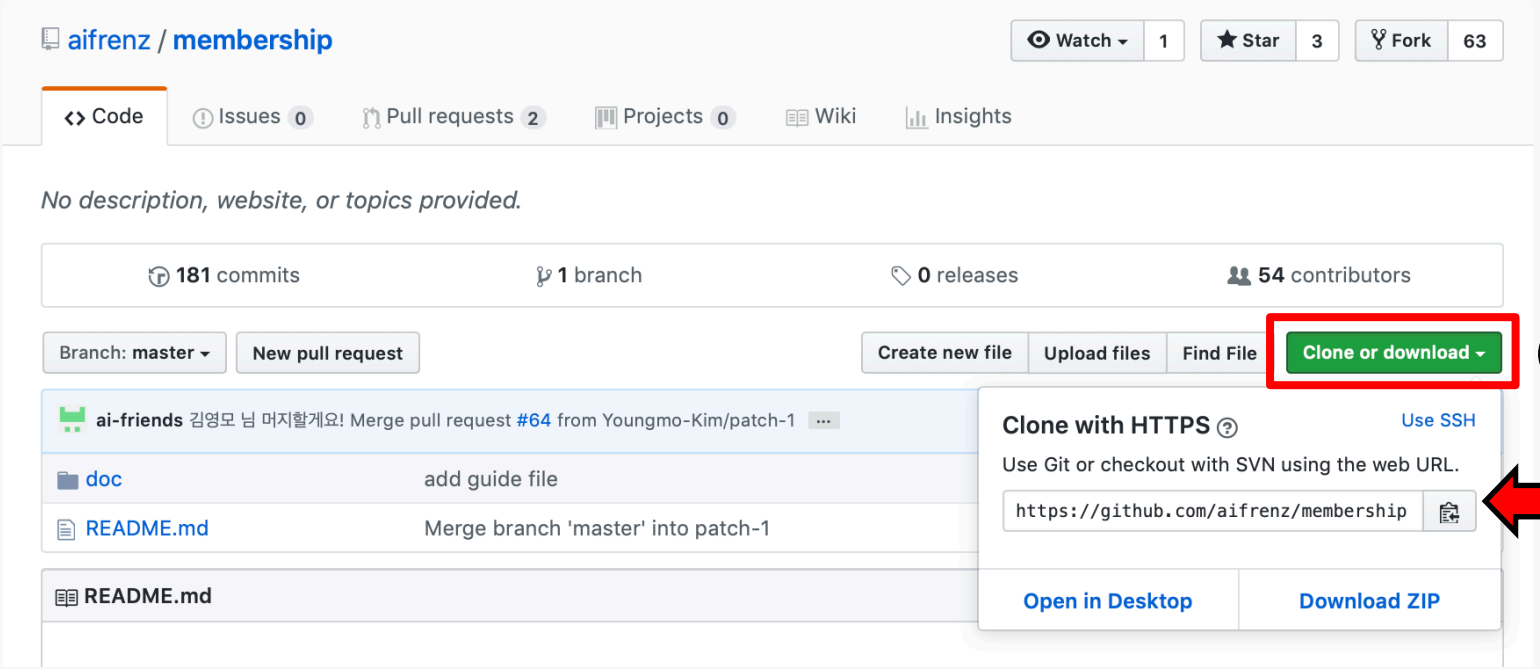


3. Git Server

1. GitHub
    - Free for private repositories (< 3 users)
    - https://github.com
    - Microsoft.
2. Bitbucket
    - Free for private repositories (< 5 users)
    - https://bitbucket.org
    - Atlassian
3. My server
    - Local storage (HDD, SDD)
    - Network attached storage (NAS)

3. Git Server 연결하기

1. Git Server 주소 얻기



2. Git Server와 연결하기

③ `git clone https://github.com/aifrenz/membership.git⏎`

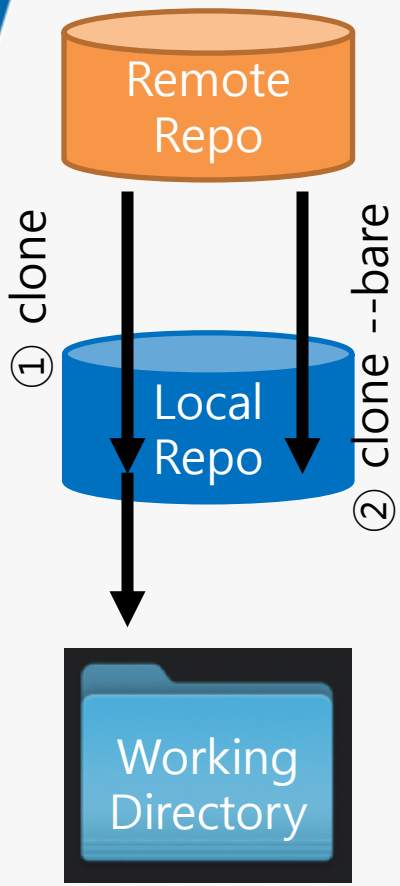
※ 원격 저장소를 로컬 저장 장치에 복사하는 방식으로 로컬 저장소를 생성

4. 사용하기

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

4. 사용하기



① 원격 저장소를 복사하면서 지역 저장소와 작업공간을 생성하기

```
git clone (원격 저장소 주소)⏎
```

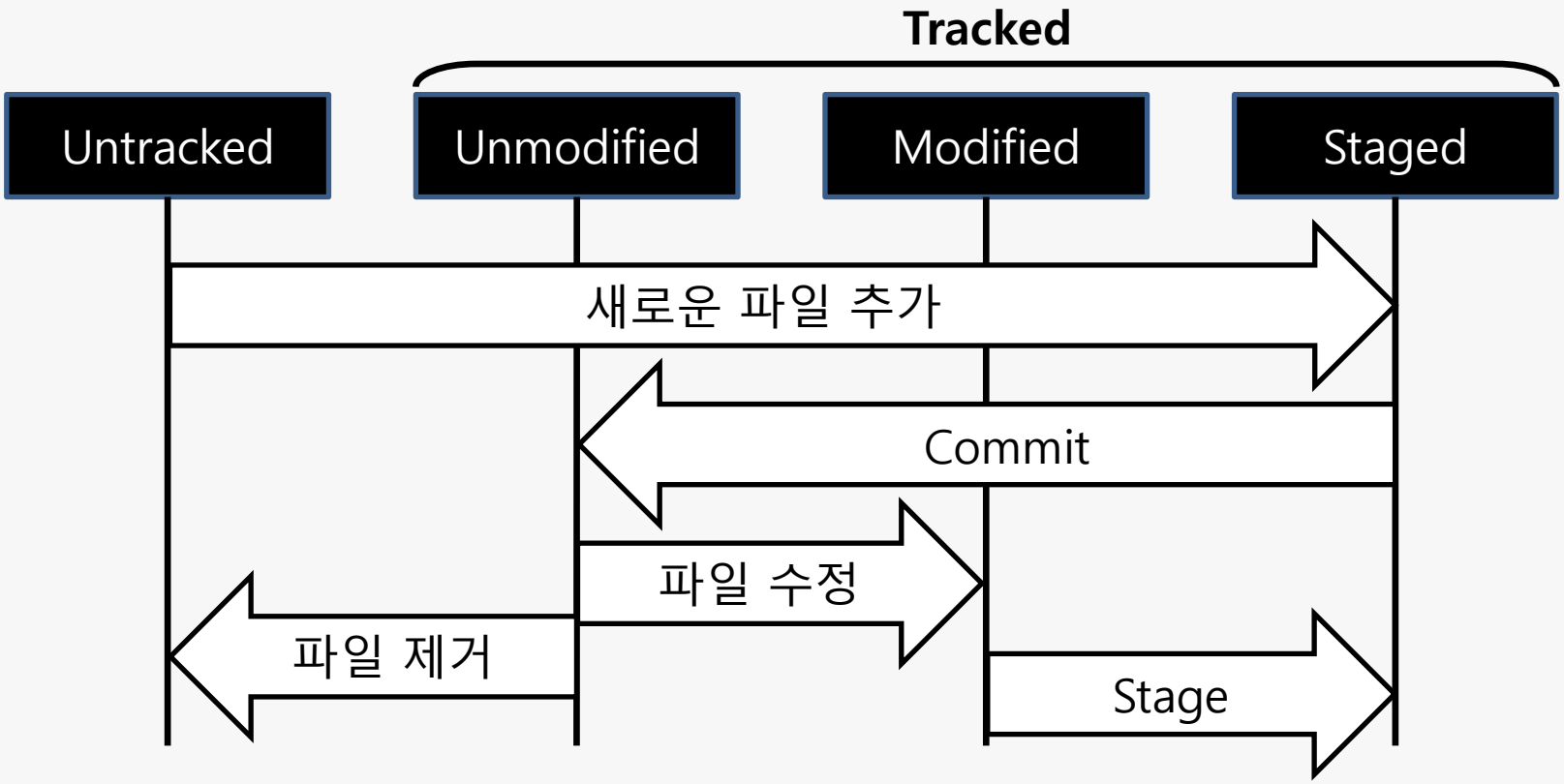
```
git clone https://github.com/aifrenz/membership.git⏎
```

② 원격 저장소를 복사해오면서 지역저장소만 생성하기

```
git clone (원격 저장소 주소) --bare⏎
```

```
git clone --bare https://github.com/aifrenz/membership.git⏎
```

4. 사용하기



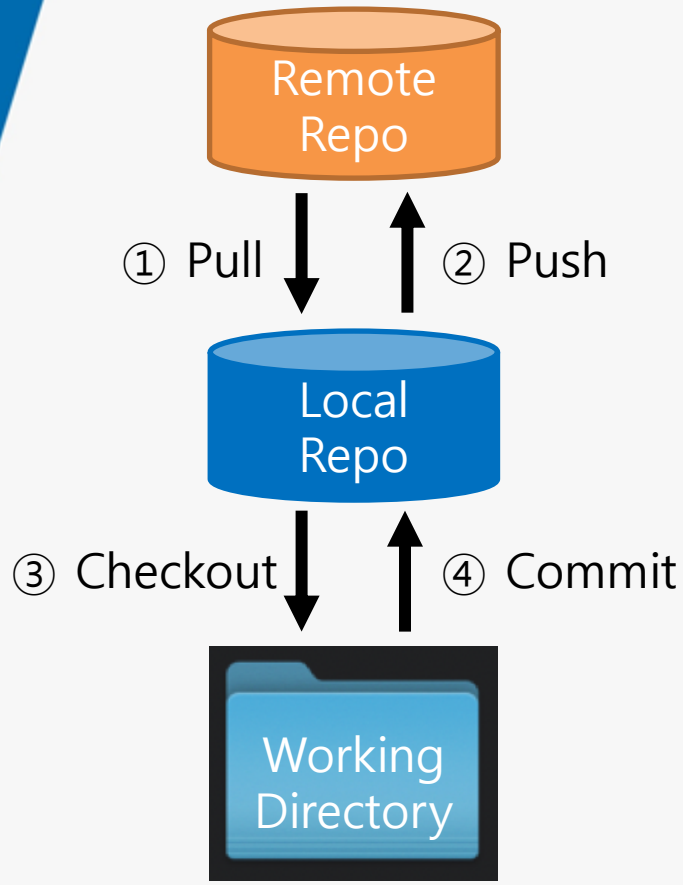
상태 출력: git status⏎

파일 추가: git add 파일명⏎

파일 제거: git rm 파일명⏎

파일 이동: git mv 파일명1 파일명2⏎

4. 사용하기



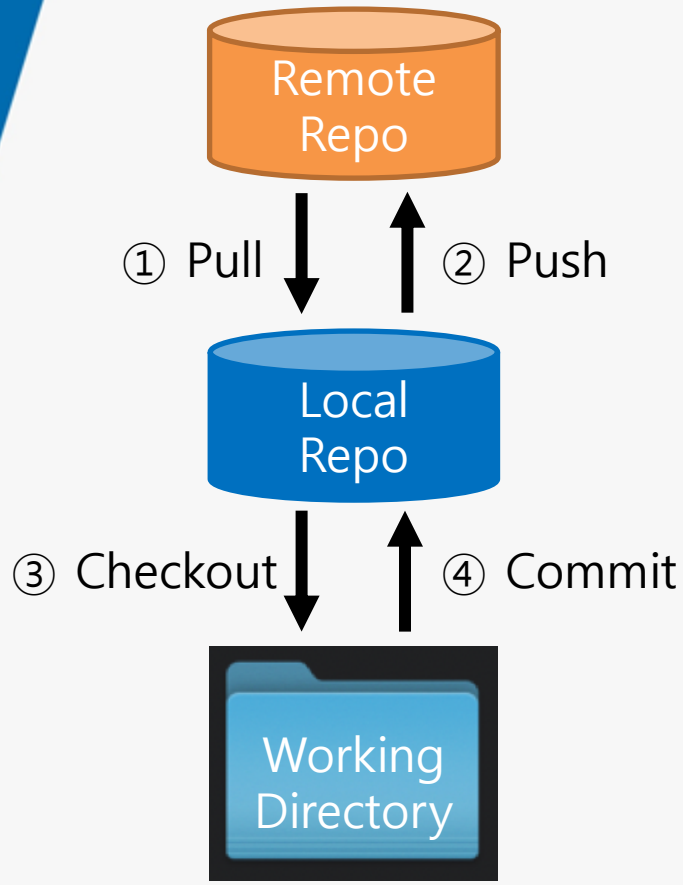
① 원격 저장소 → 지역 저장소

```
git pull⏎
git pull (remote)⏎
git pull (remote) (branch)⏎
```

② 지역 저장소 → 원격 저장소

```
git push⏎
git push (remote)⏎
git push (remote) (branch)⏎
```

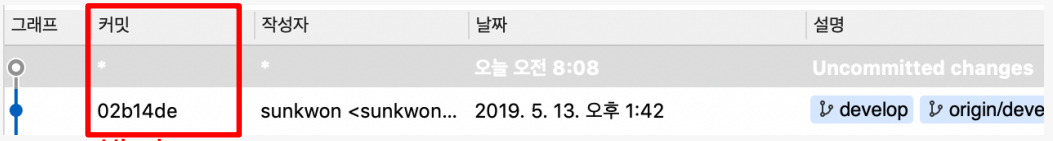
4. 사용하기



③ 지역 저장소 → 작업 공간

브랜치 이동: `git checkout (branch)⏎`

커밋 히스토리 이동: `git checkout (hashcode)⏎`



sourcetree

④ 작업 공간 → 지역 저장소

`git commit –m "설명/기록"⏎`

Source Tree (Git)의

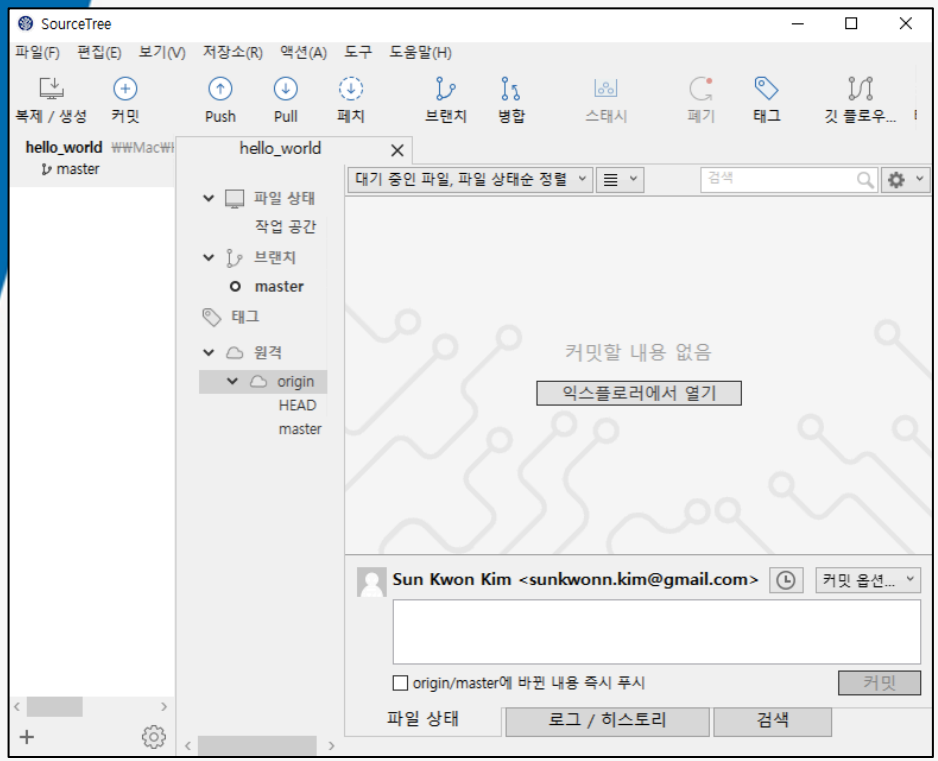
5. GUI 클라이언트

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

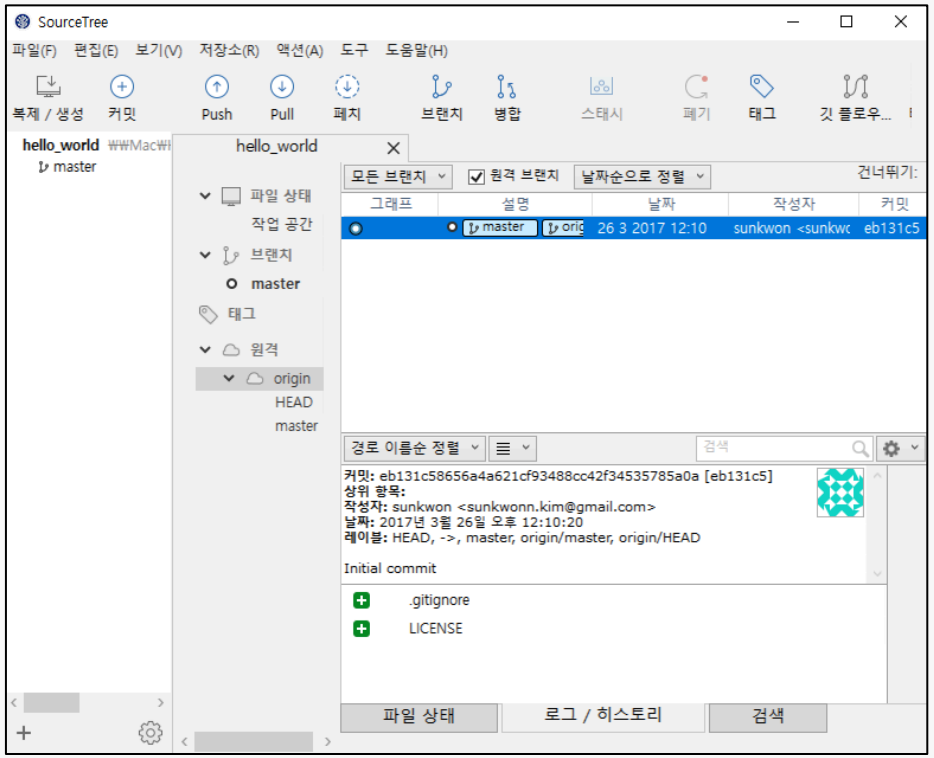
5. GUI 클라이언트

파일 상태 뷰(CTRL + 1)



- Commit하기 위해 필요한 기능이 모여 있음
- Commit할 내용 선택 및 코맨트 입력
- 소스 코드 충돌 (Conflict)시 해결하는 기능

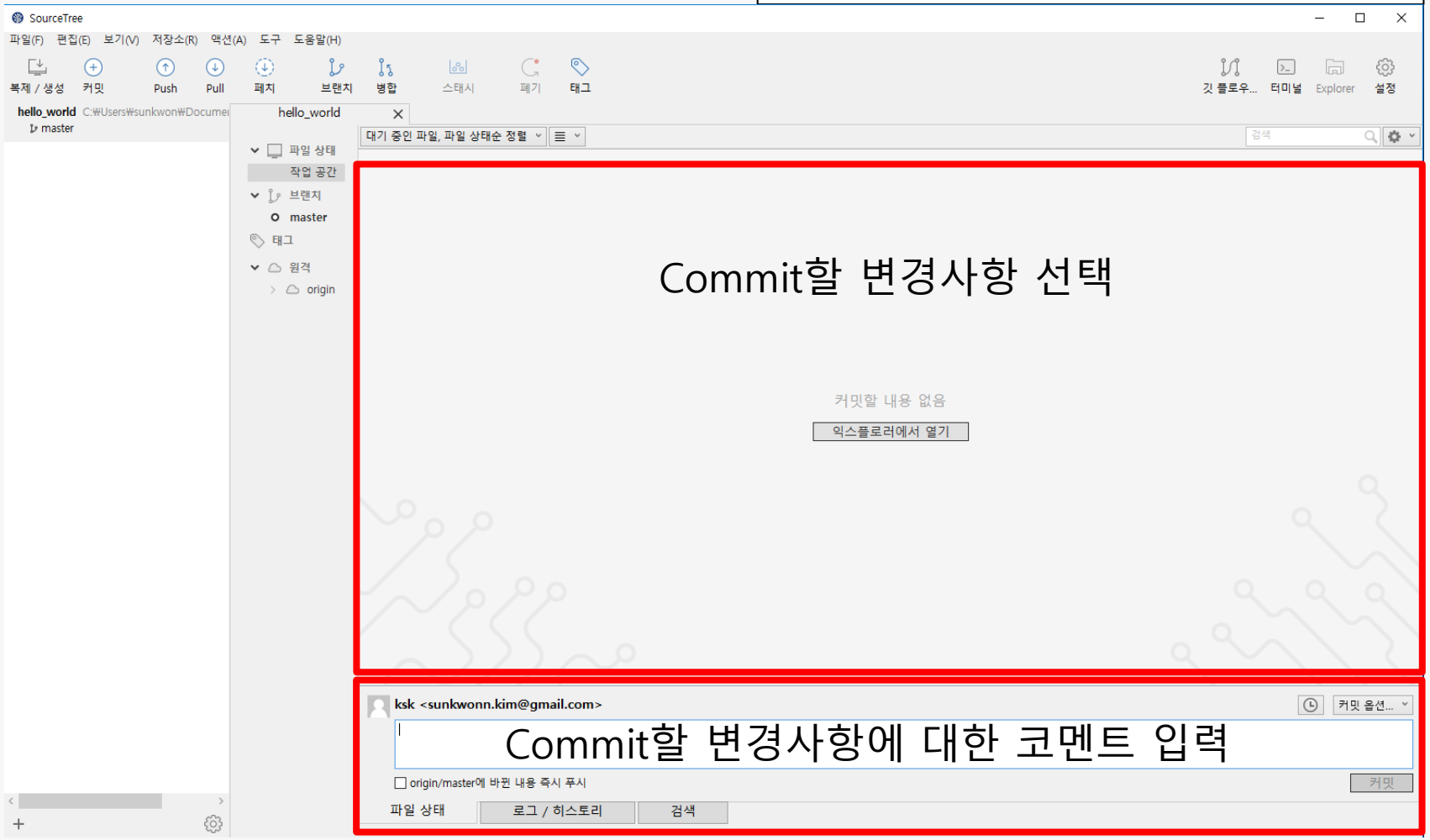
로그 보기 뷰(CTRL + 2)



- 소스 코드 히스토리를 조작하는 모든 기능
- 히스토리(로그) 조회/이동
- 브랜치 생성 / 합치기 (Merge)

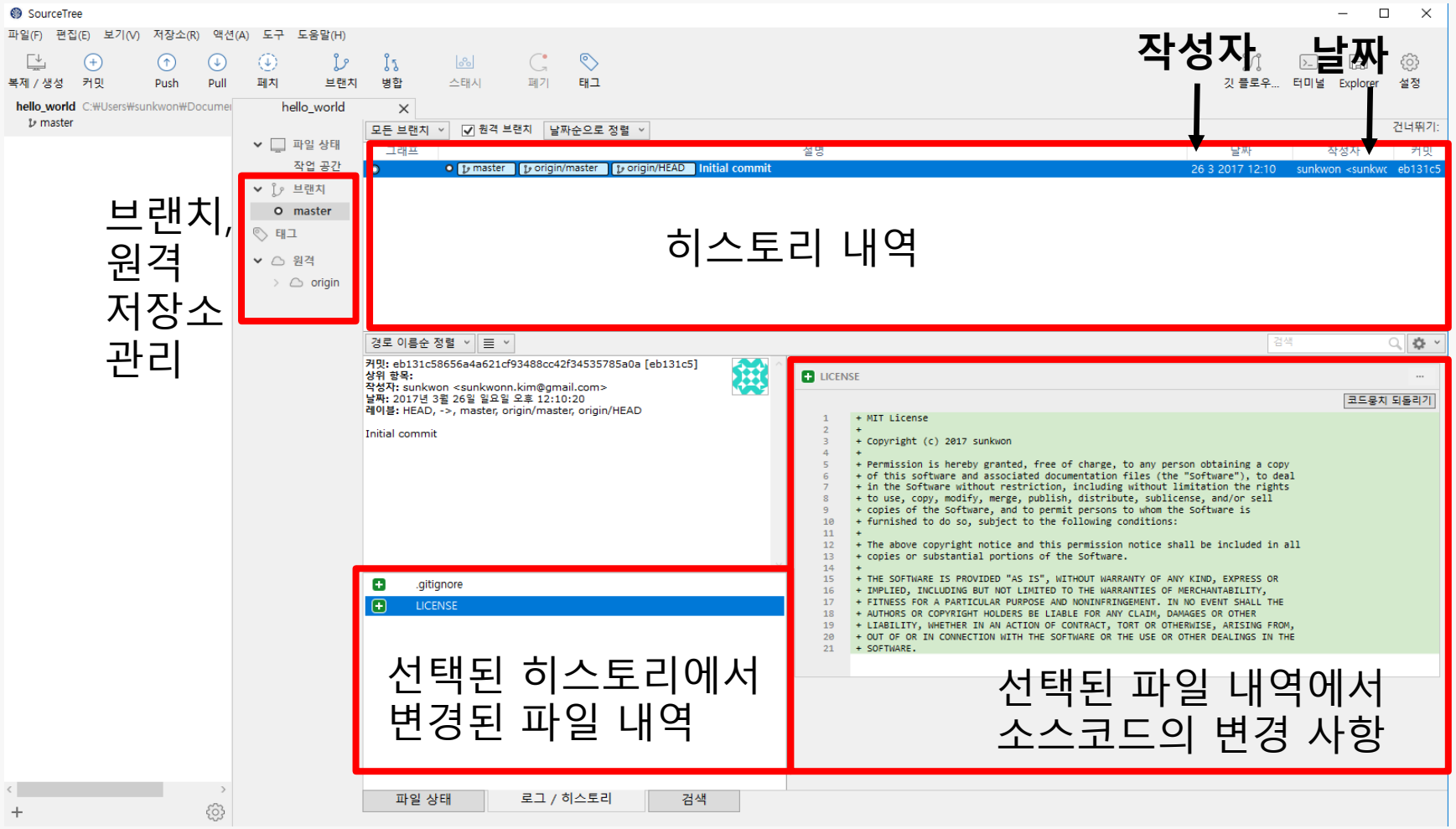
5. GUI 클라이언트

1. 파일 상태 뷰(CTRL + 1)



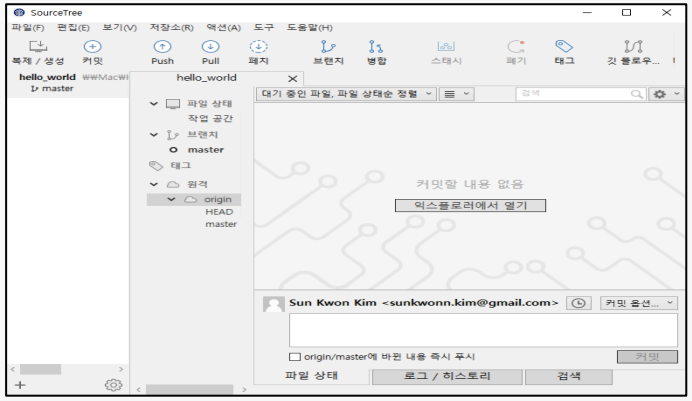
5. GUI 클라이언트

히스토리 뷰 (단축키: CTRL + 2)

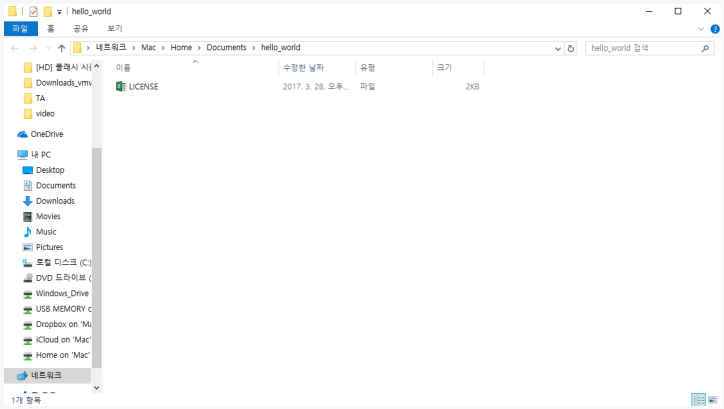


5. GUI 클라이언트

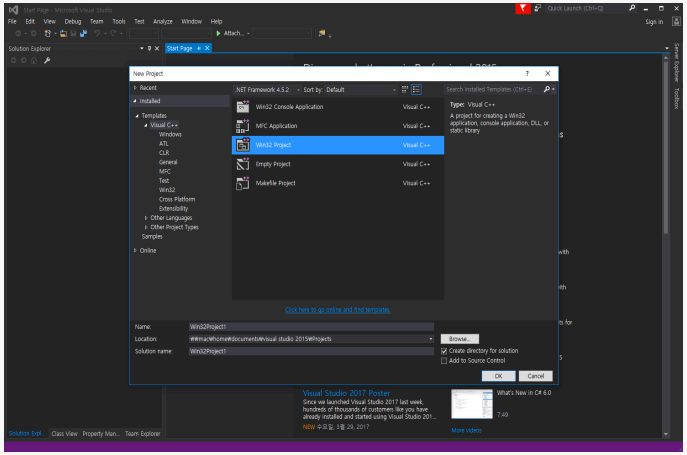
1. 익스플로어에서 열기 클릭



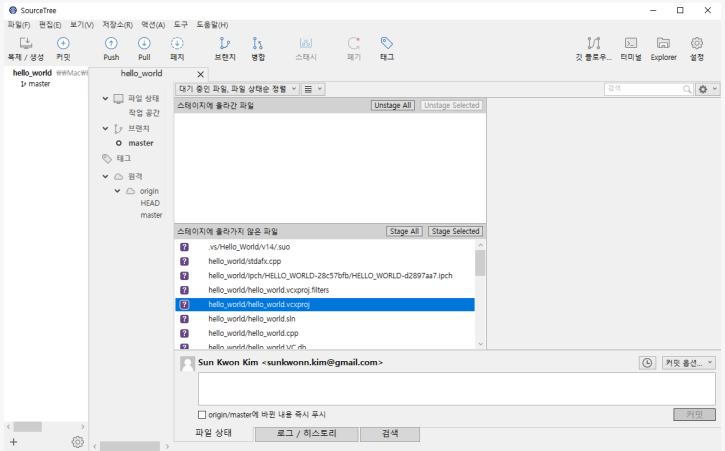
2. 작업 공간



3. 작업 공간에 프로젝트 생성



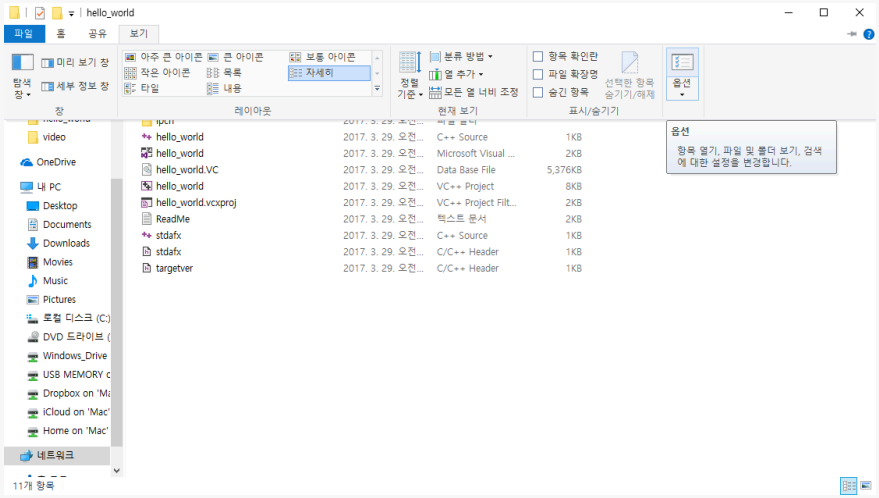
4. 파일 상태 뷰에서 변경사항 체크



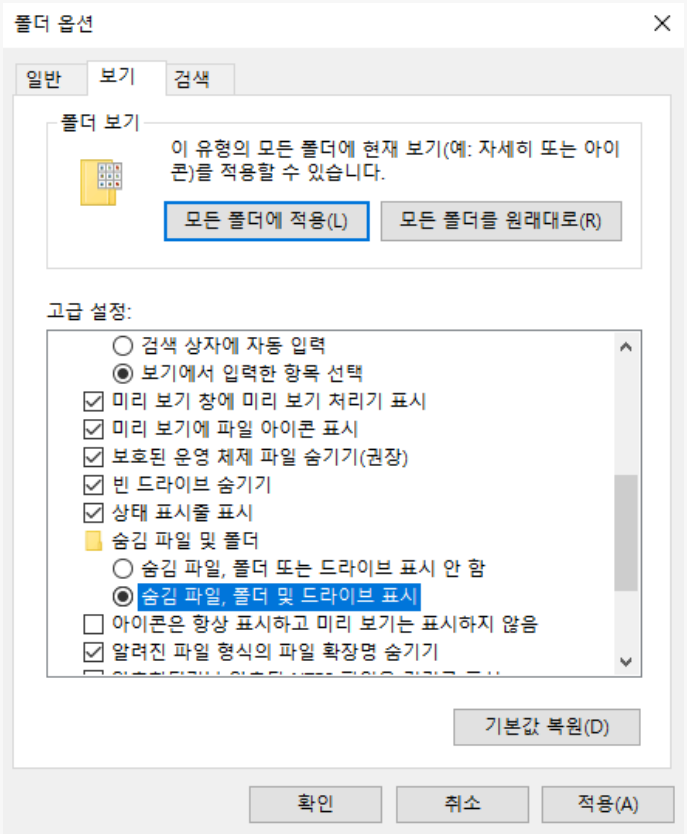
5. GUI 클라이언트

KERI

5. 탐색기 → 보기 → 옵션 선택

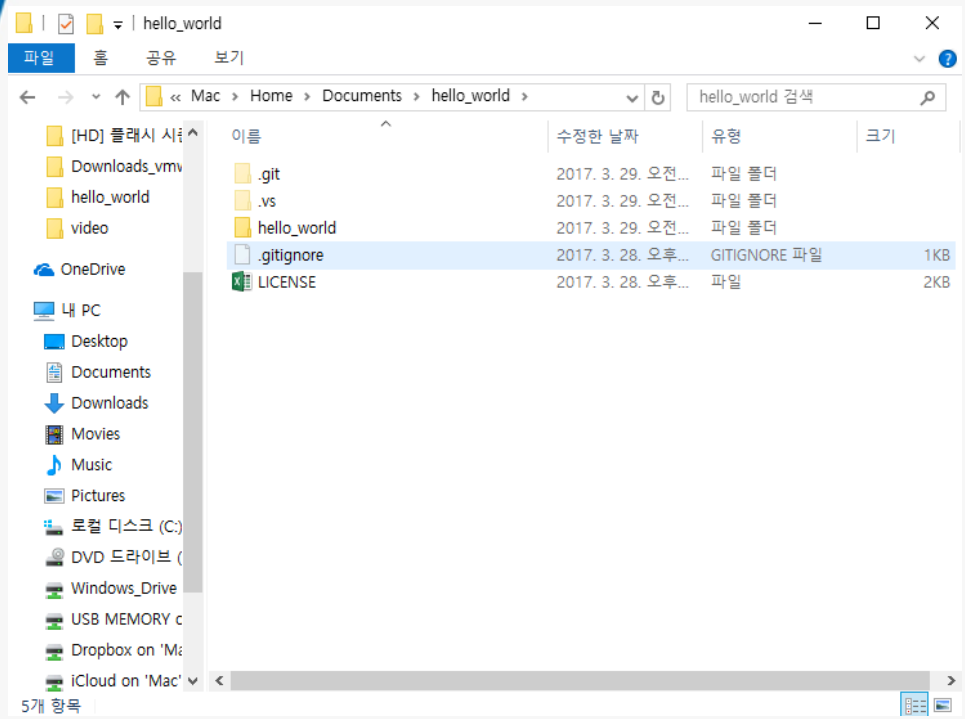


6. 숨김파일이 보이도록 변경



5. GUI 클라이언트

7. ".gitignore" 파일을 텍스트 편집기로 열기

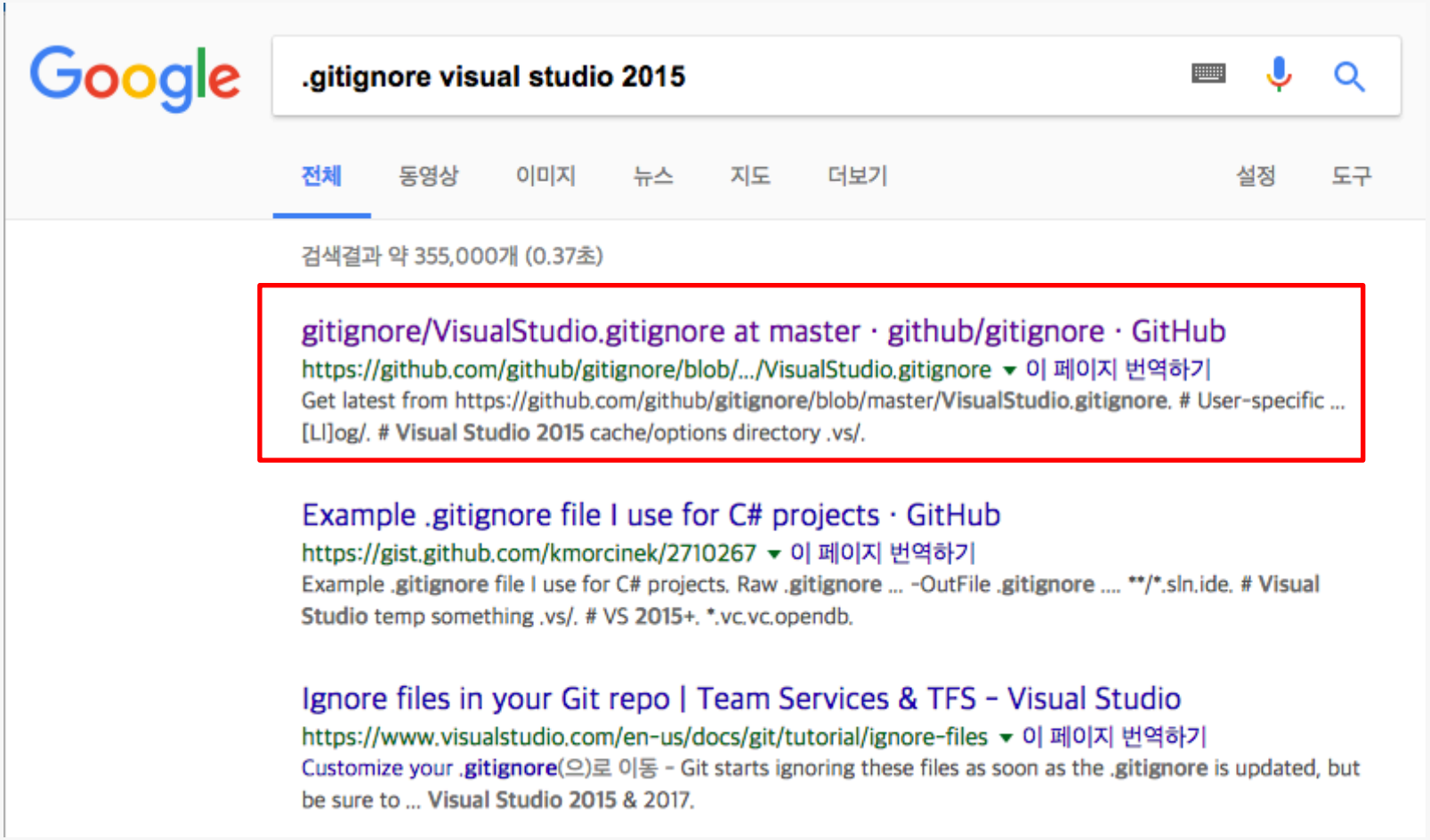


.gitignore - 메모장

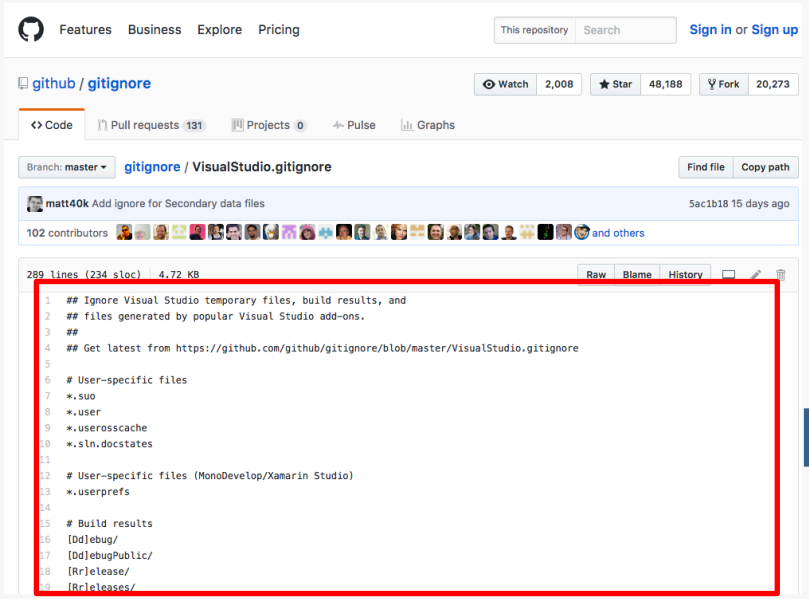
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

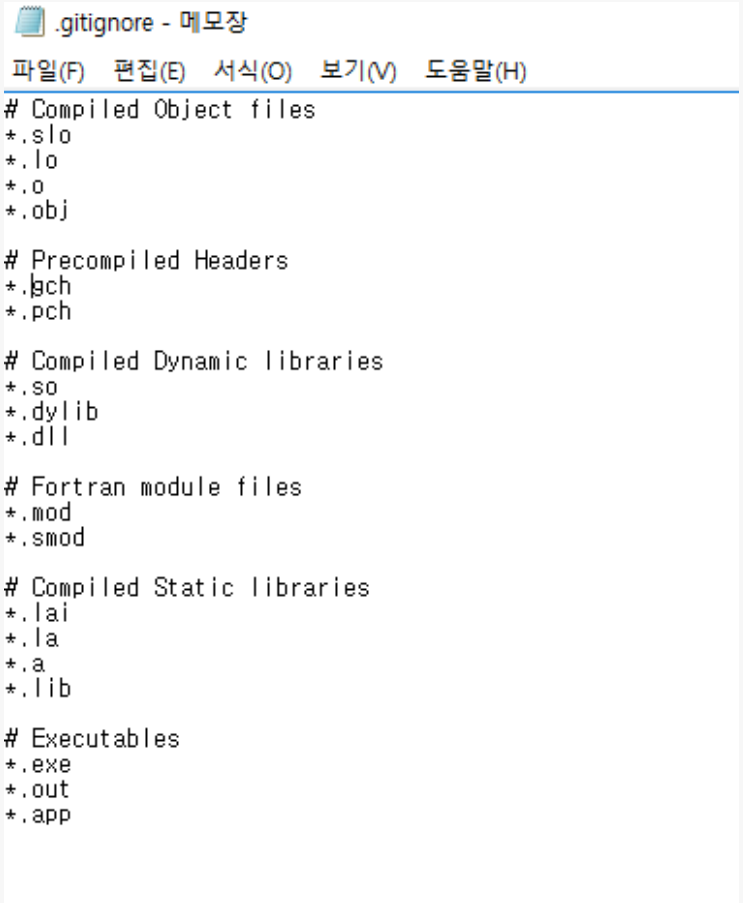
```
# Compiled Object files
*.slo
*.lo
*.o
*.obj

# Precompiled Headers
*.gch
*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll

# Fortran module files
*.mod
*.smod

# Compiled Static libraries
*.lai
*.la
*.a
*.lib

# Executables
*.exe
*.out
*.app
```

8. 구글에서 ".gitignore + 사용 프로그래밍 환경"을 검색

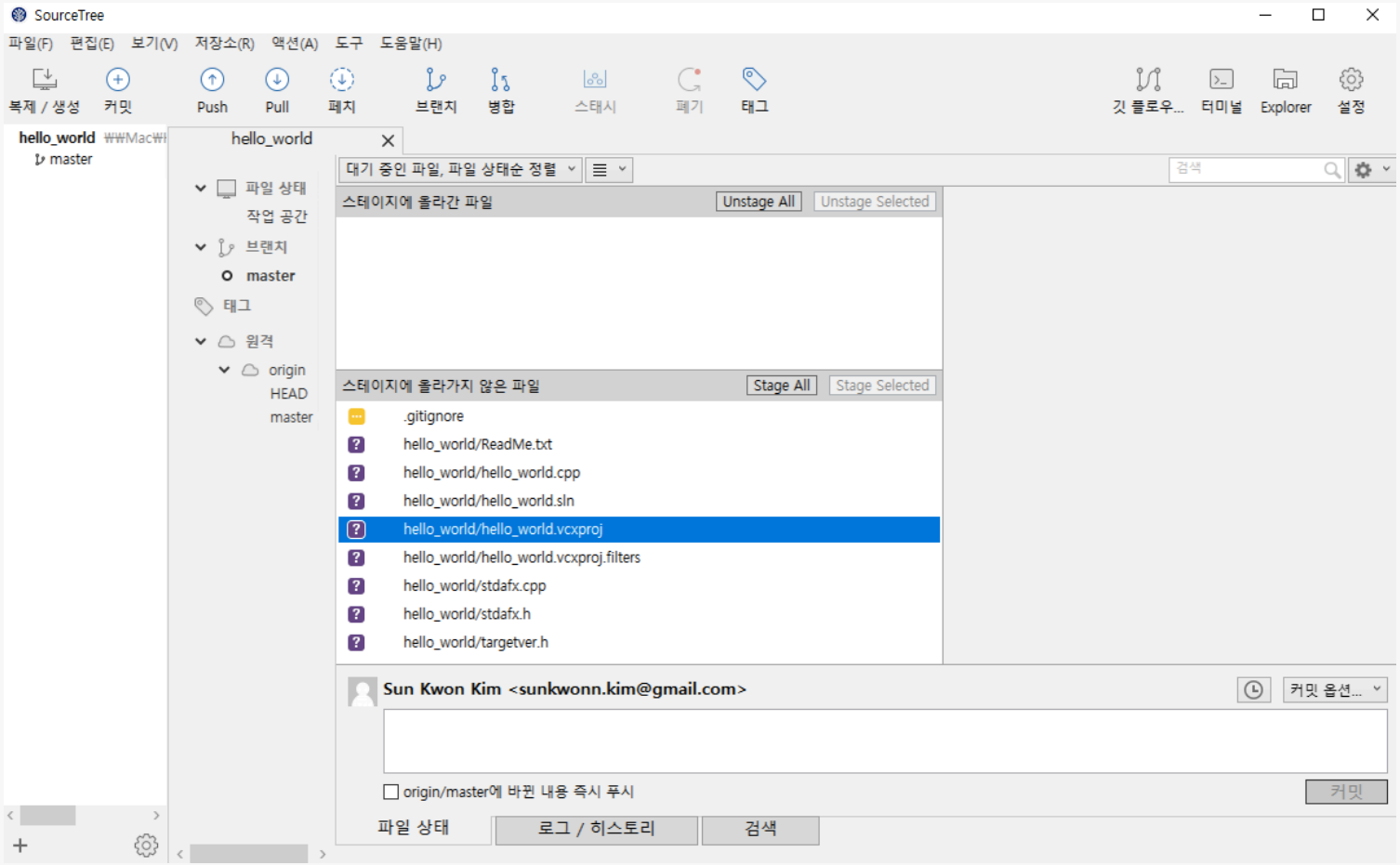


9. 검색된 내용을 ".gitignore" 파일에 붙여넣기



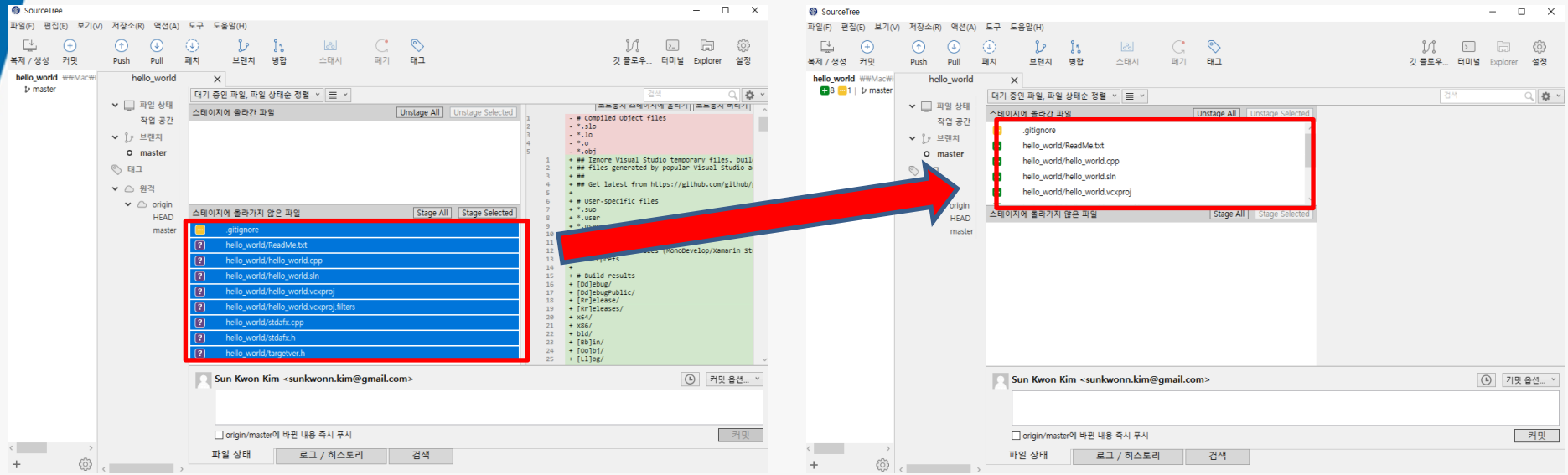


10. 저장소에 저장할 필요가 없는 불필요한 파일은 자동 무시됨



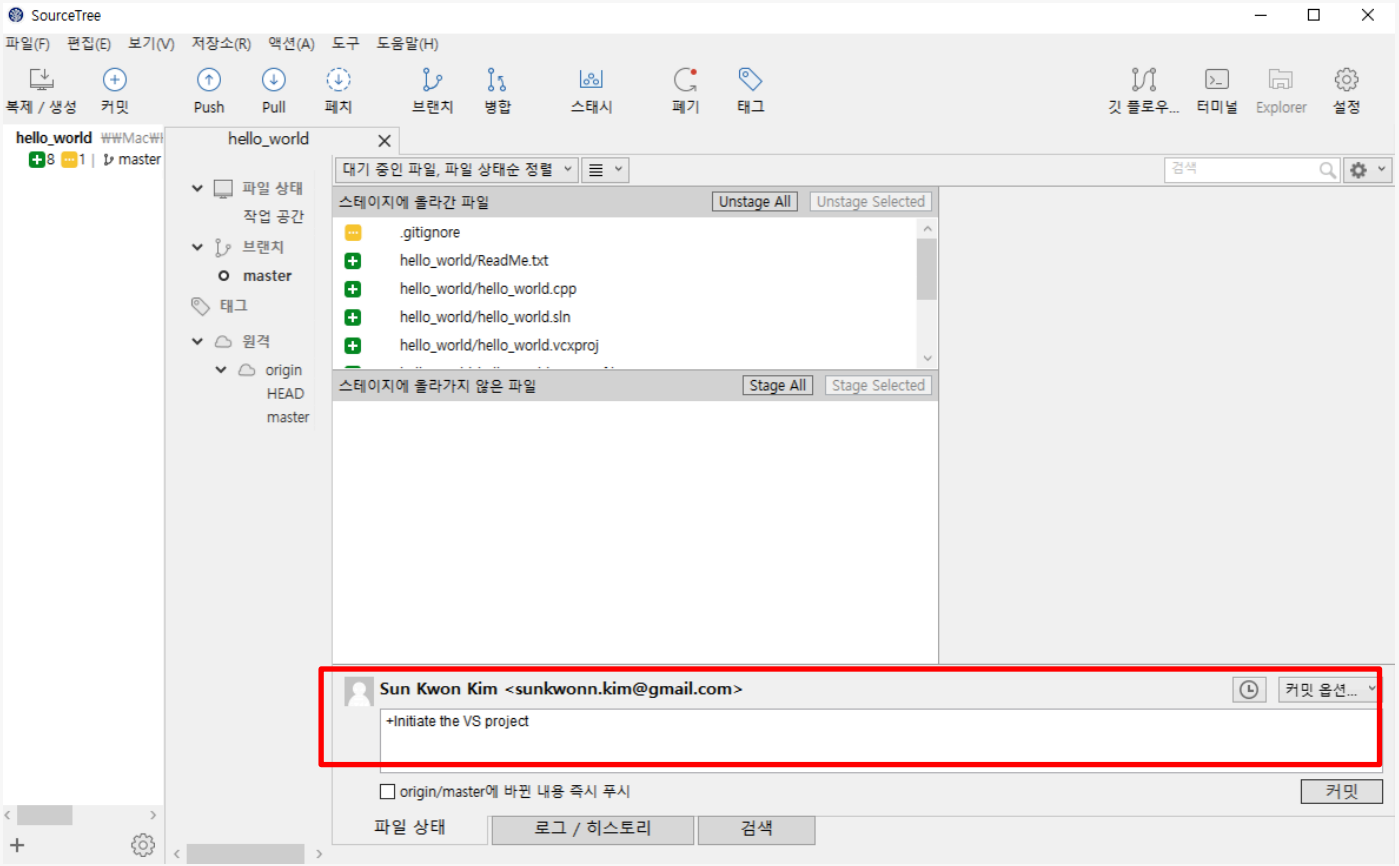
5. GUI 클라이언트

11. “Stage all” 또는 “Stage Selected” 버튼을 눌러서 로컬 저장소에 저장할 파일 상태(스테이지)로 넣음



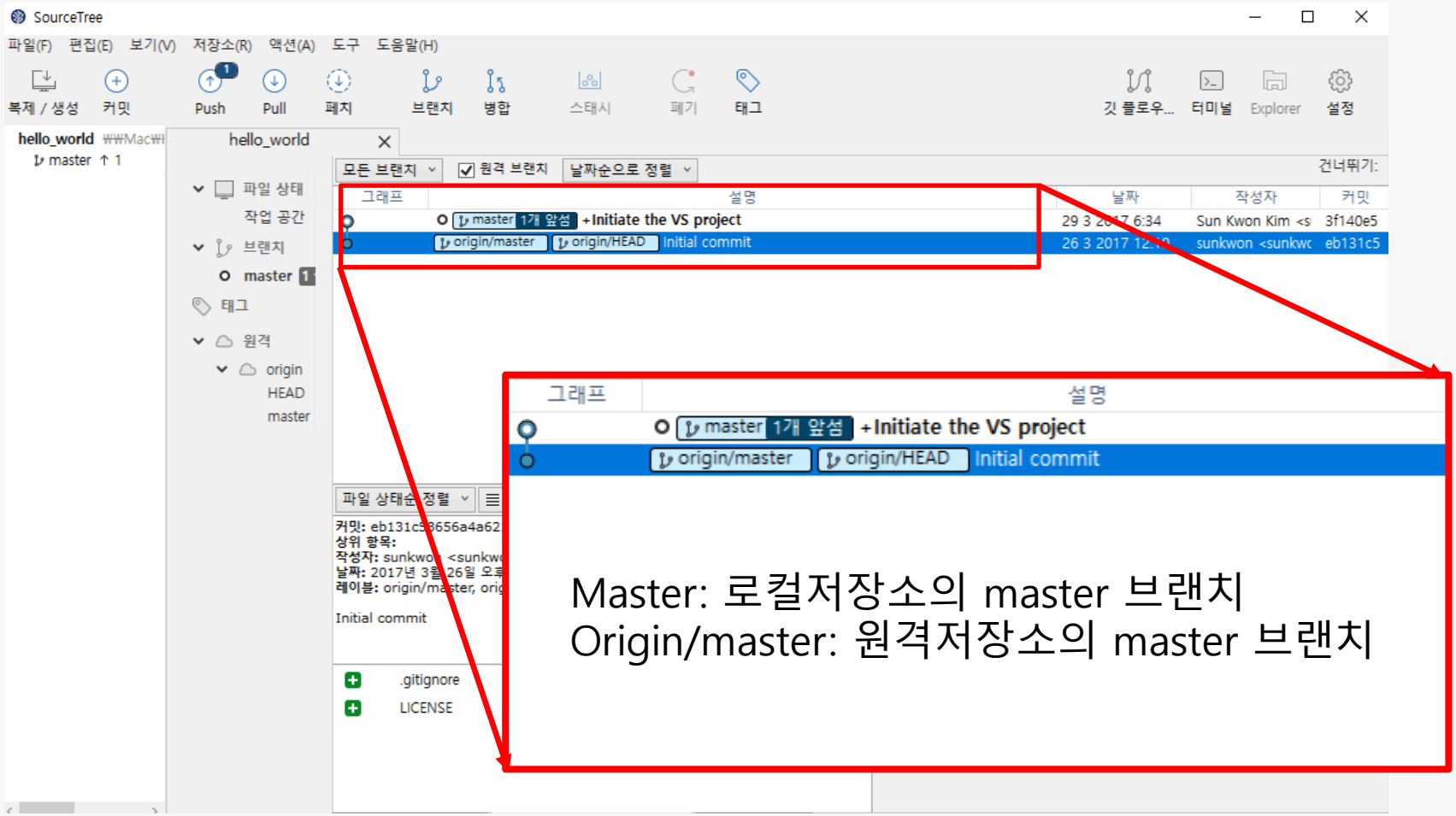
5. GUI 클라이언트

12. Commit할 내용에 대해서 기술한 다음 "Commit (커밋)" 버튼을 클릭

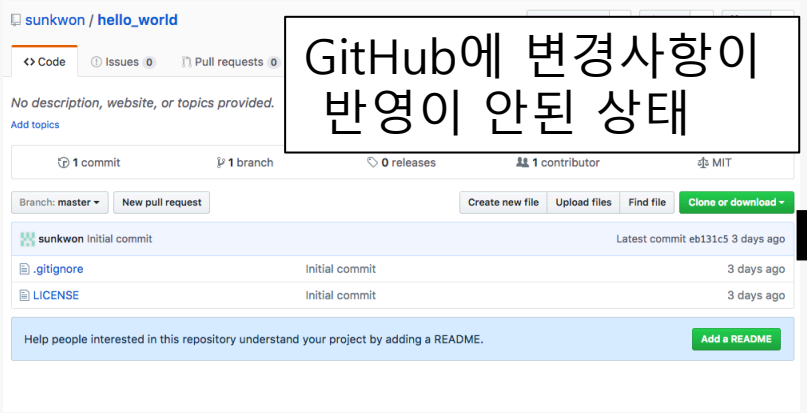


5. GUI 클라이언트

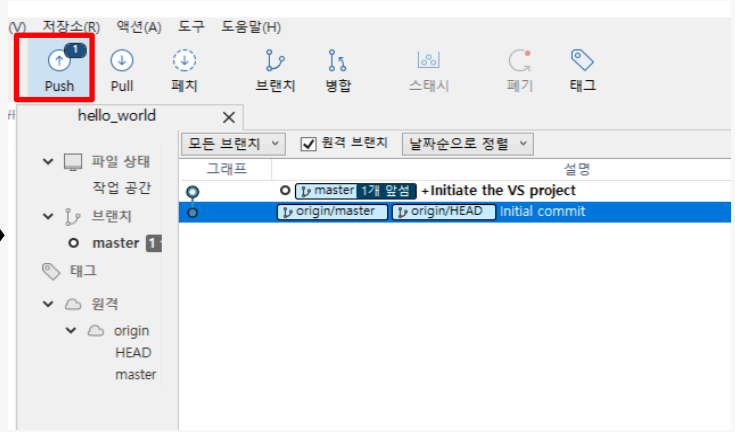
13. Commit할 내용에 대해서 기술한 다음 "Commit (커밋)" 버튼을 클릭

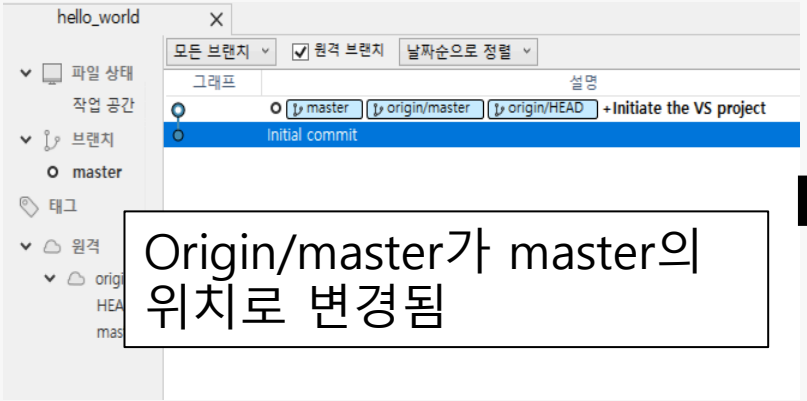


14. "Push (푸시)" 버튼을 눌러서 원격저장소 (GitHub)에 변경사항을 업로드

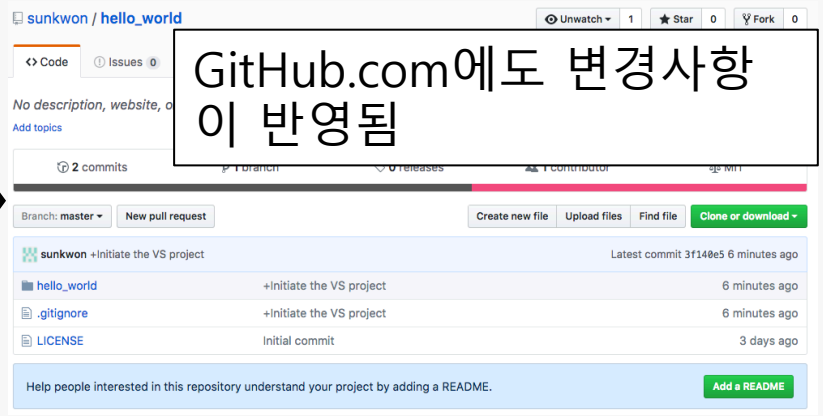


GitHub에 변경사항이 반영이 안된 상태





Origin/master가 master의 위치로 변경됨

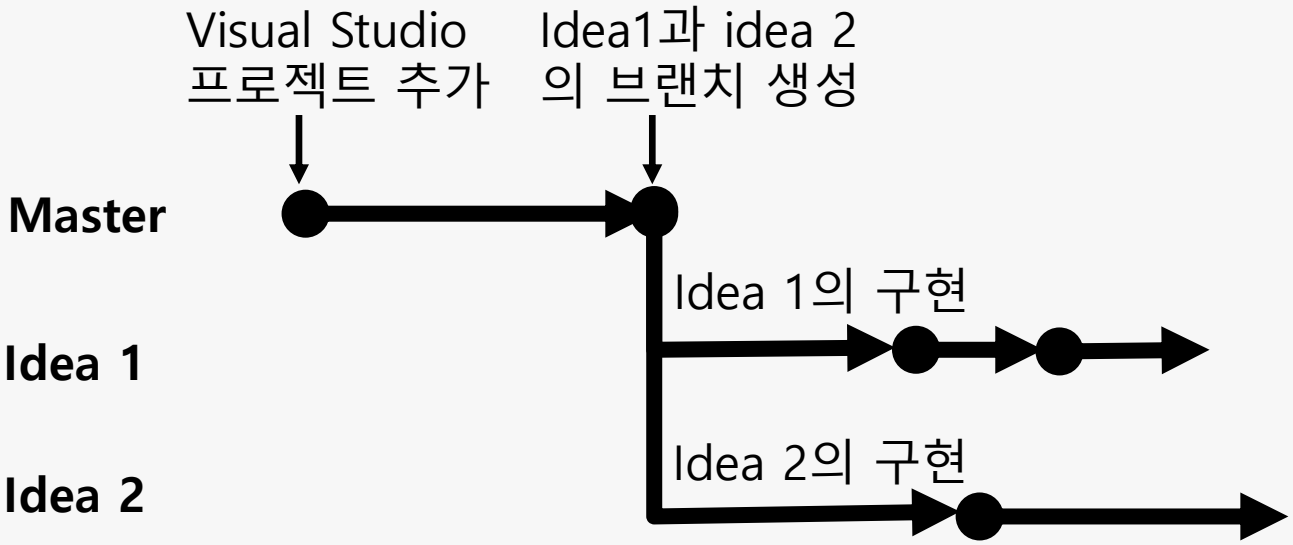


GitHub.com에도 변경사항이 반영됨

5. GUI 클라이언트: 브랜치 생성 (Branch)

- 브랜치 == 하나의 대표되는 흐름
- 새로운 아이디어나 기능을 구현해야할 때에는 별도의 브랜치를 생성하여 작업을 하자.

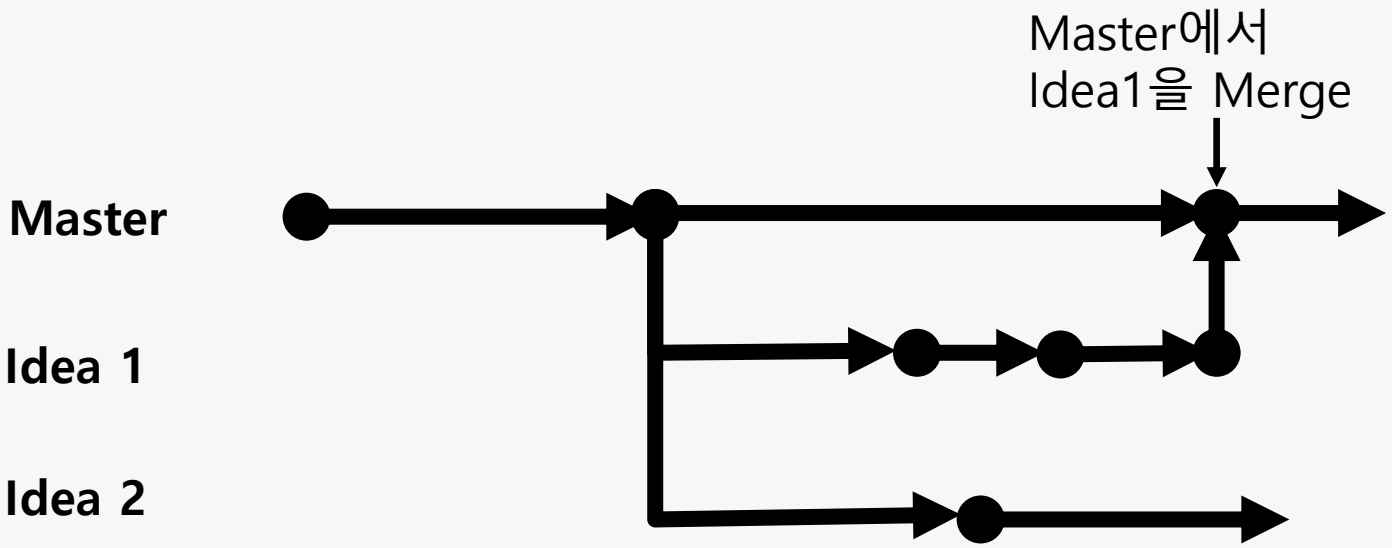
(예시) 어떤 문제를 해결하기 위하여 2가지의 아이디어가 도출되었고, 각각의 아이디어를 구현/검증해 보기로 결정함



5. GUI 클라이언트: 브랜치 합치기 (Merge)

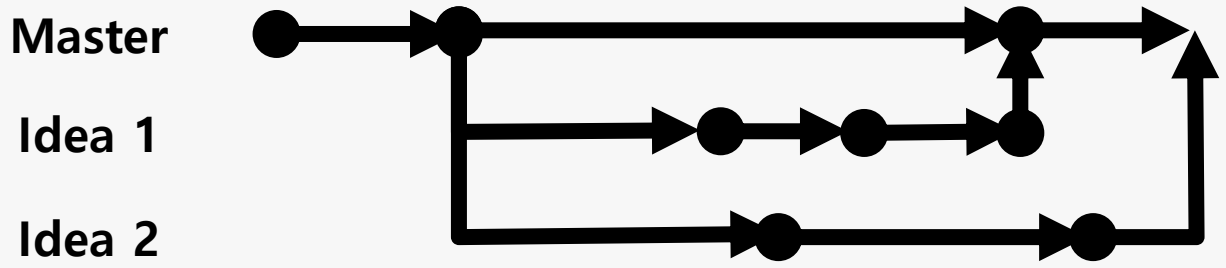
- 새로운 아이디어가 타당하여 본 프로젝트에 정식으로 편입시키고자 할 때 브랜치 합치기 (Merge)를 한다

(예시) Idea 1의 구현 결과가 성능이 좋아 본 프로젝트에 편입시키기로 결정함. Master 브랜치의 최신 히스토리로 이동(Check-out)한 후, Idea1을 Merge 한다.

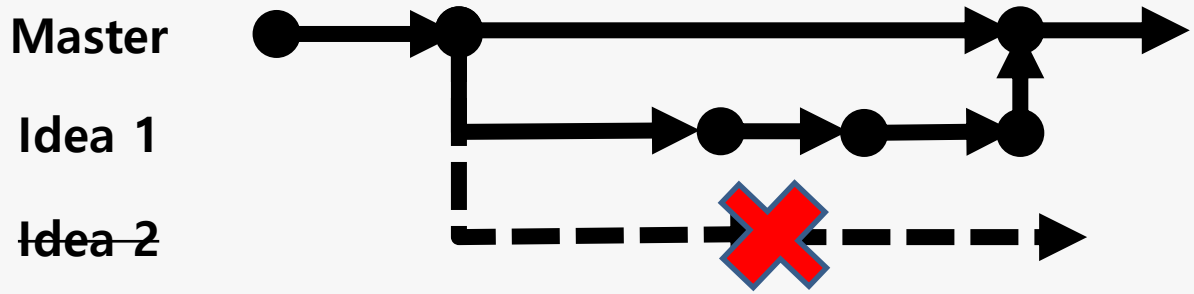


5. GUI 클라이언트: 브랜치 삭제 (Delete)

(예시) Idea 2의 결과 역시 만족스럽거나 별도의 기능이어서, 후에 본 프로젝트에 합치고자 하면 역시 같은 방법으로 Merge 하거나,



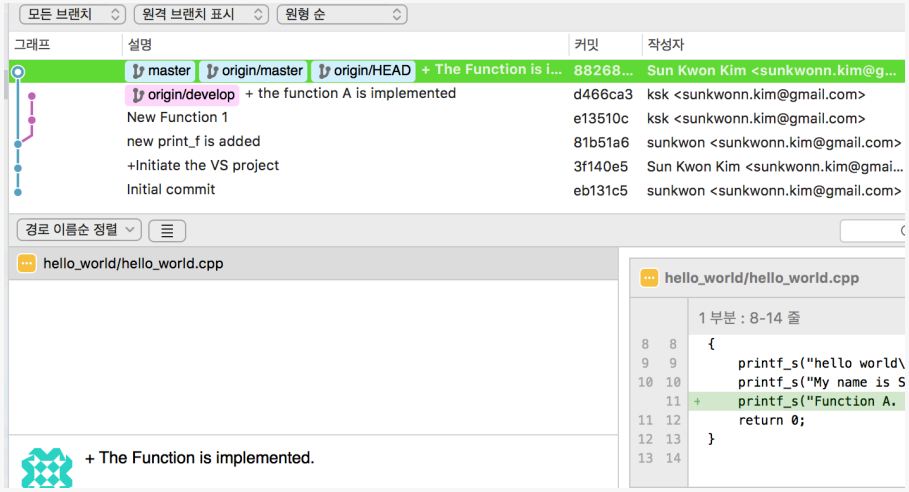
삭제할 수 있다. (Idea 2의 내용만 깔끔하게 삭제된다.)



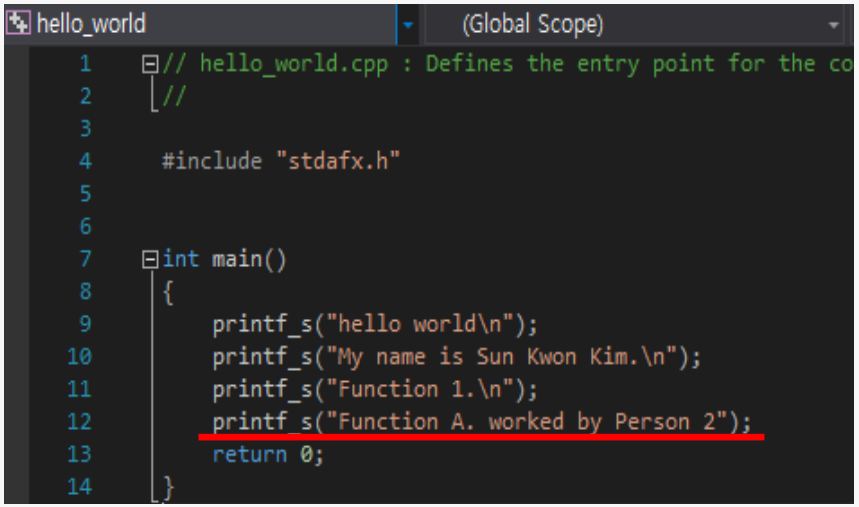
5. GUI 클라이언트: 충돌 해결 (Resolve Conflicts)

(예시) 작업자 A와 작업자 B가 동시에 동일한 부분(파일)을 수정함.

작업자 A

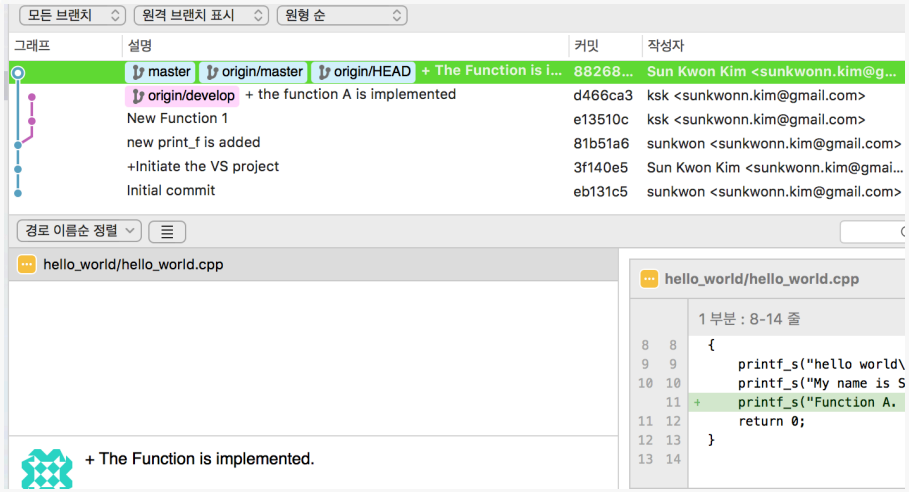


작업자 B



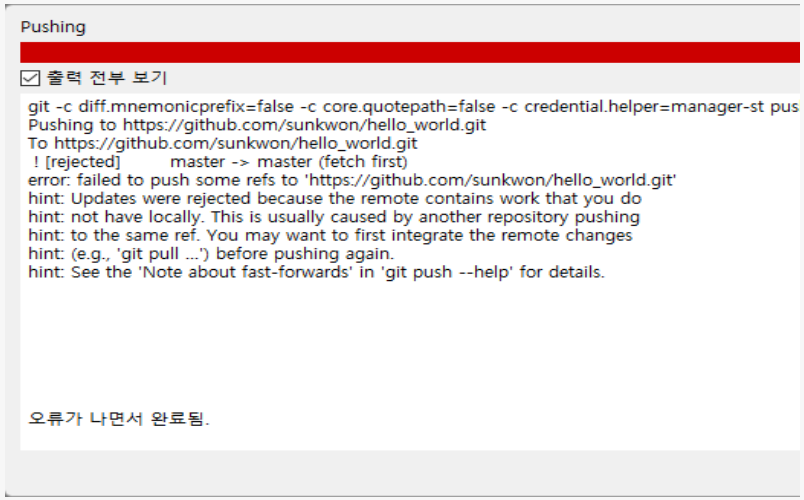
5. GUI 클라이언트: 충돌 해결 (Resolve Conflicts)

작업자 A



Commit과 Push 성공

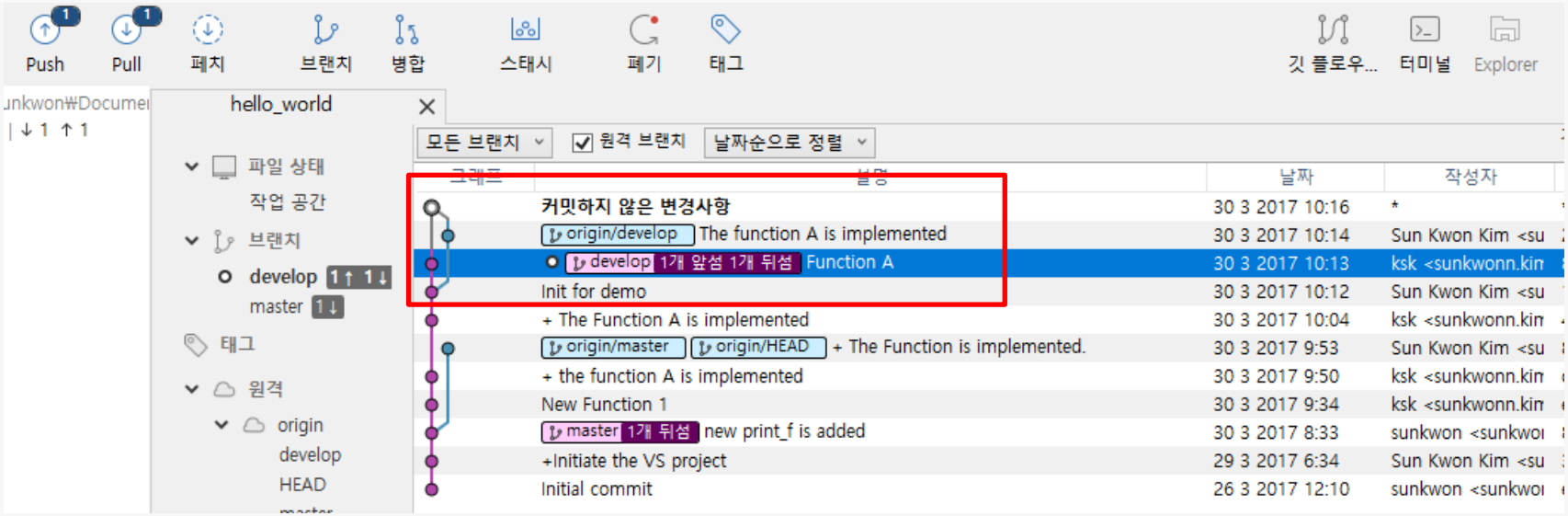
작업자 B



Commit은 성공하지만,
Push에서는 에러 발생

5. 사용 방법 : 충돌 해결 (Resolve Conflicts)

작업자 B

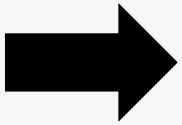


Pull 버튼을 눌러서 원격 저장소의 정보를 가져와 보면, 원격 저장소에 저장된 내용(작업자 A)와 현재 작업한 내용(작업자B)가 충돌이 발생했다는 것을 확인할 수 있음

5. GUI 클라이언트: 충돌 해결 (Resolve Conflicts)

사용하는 프로그래밍툴을 이용하여 충돌이 일어난 부분을 수정함

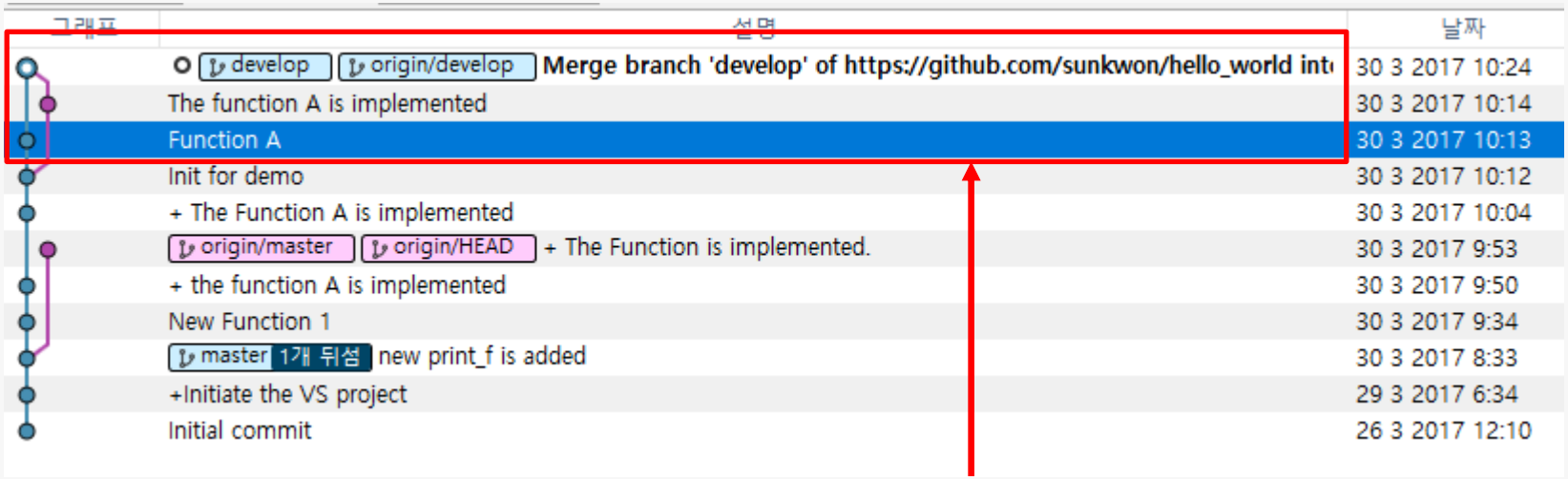
```
1   // hello_world.cpp : Defines the entry point for the console application.
2   //
3
4    #include "stdafx.h"
5
6
7   int main()
8   {
9       printf_s("hello world\n");
10      printf_s("My name is Sun Kwon Kim.\n");
11  <<<<<<< HEAD
12      printf_s("The Function A. implemented by Person B.\n");
13  =======
14      printf_s("The function is implemented. implemented by person A.\n");
15  >>>>>>> 201248c2d832fa4bc2f0bc724d6ef70f3a404631
16      return 0;
17  }
18
```



```
llo_world   (Global Scope)   main()
1   // hello_world.cpp : Defines the entry point for the console application.
2   //
3
4    #include "stdafx.h"
5
6
7   int main()
8   {
9       printf_s("hello world\n");
10      printf_s("My name is Sun Kwon Kim.\n");
11      printf_s("The Function A. implemented by Person A and B.\n");
12      return 0;
13  }
14
15
```

5. GUI 클라이언트: 충돌 해결 (Resolve Conflicts)

충돌 부분을 수정한 후 Commit과 Push 버튼을 누름



충돌이 해결되었음을 확인할 수 있음

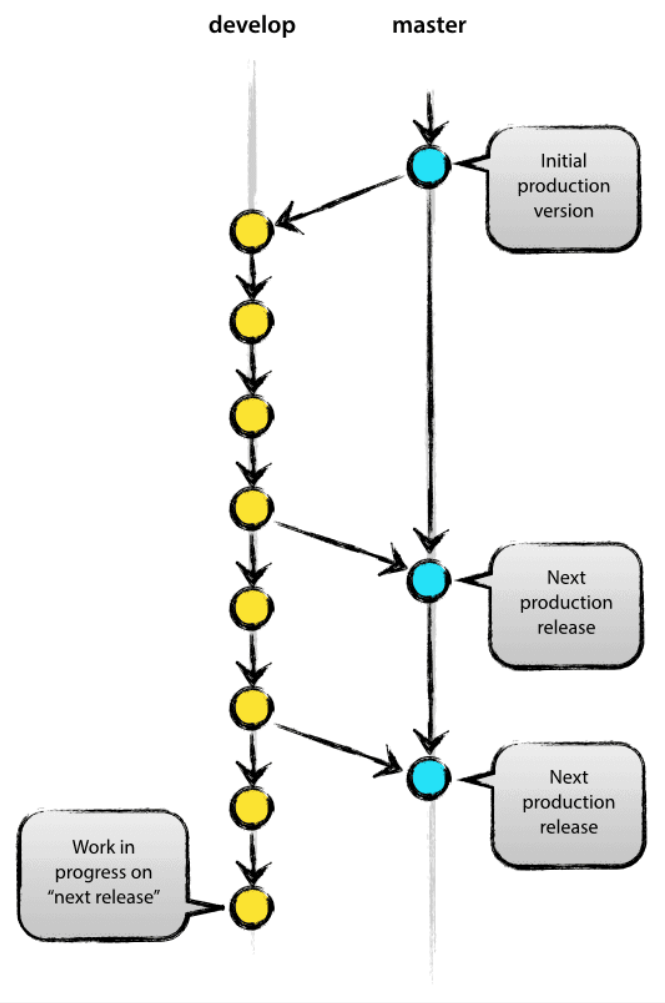
Git을 잘 사용하기 위한

6. 브랜치 관리 기법

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

6. 브랜치 관리 기법 : Simple



가장 간단한 브랜치 관리법

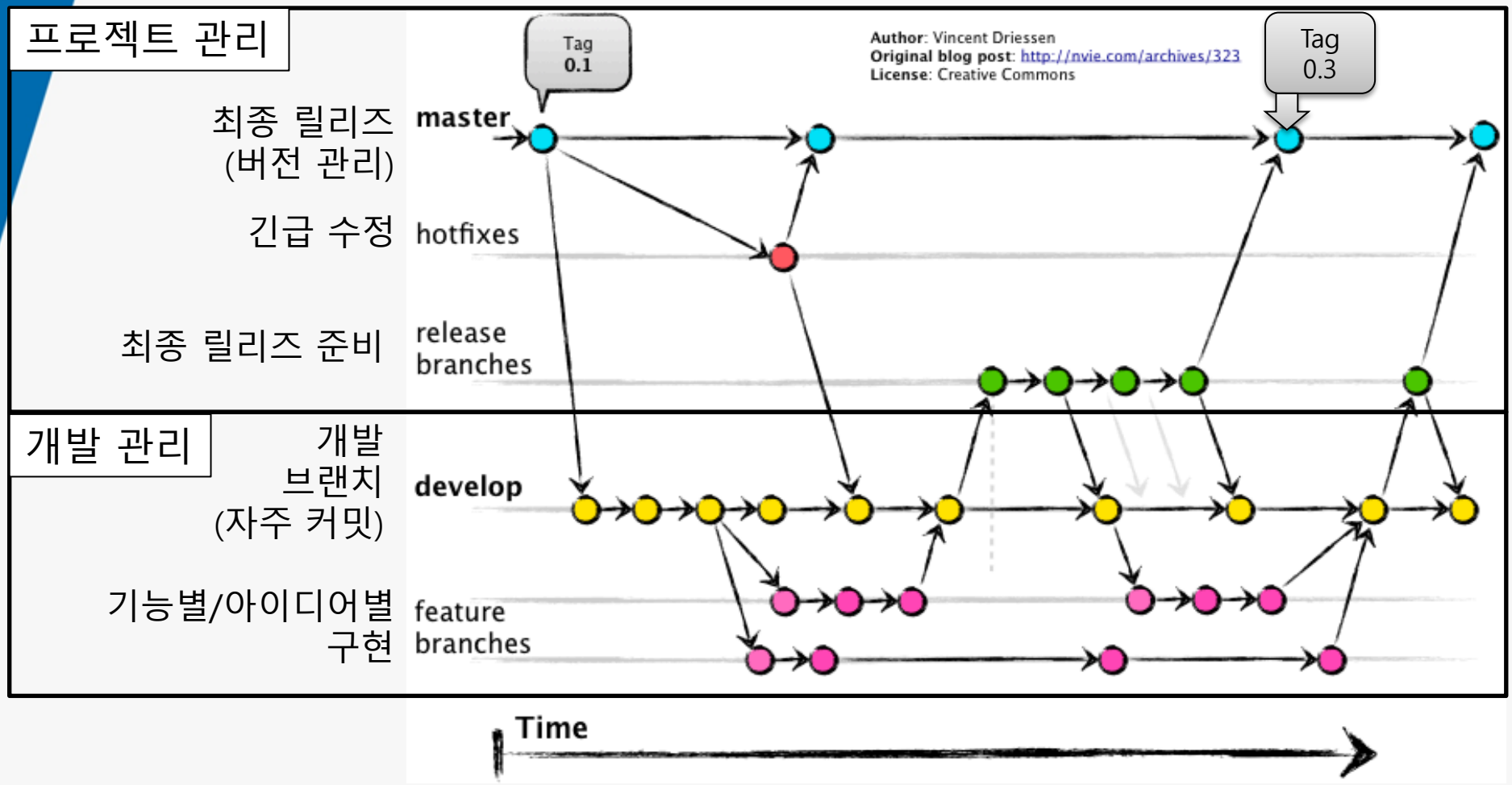
- Develop : 개발 브랜치
  - 평소에 커밋하는 브랜치
  - 여러 사람들의 작업들이 합쳐지는 브랜치
- Master : 릴리즈 브랜치
  - 릴리즈할 때마다 Develop 브랜치를 Merge
  - Release Version

- Tip:
  - Develop 브랜치는 자주 커밋을 해서 중간중간에 롤백이 가능한 포인트를 만드는 것이 좋음
  - Master 브랜치는 결과물이 출시되거나 팀 외부에 나갈 때마다 포인트를 만드는 것이 좋음

6. 브랜치 관리 기법 : Git-flow

KERI

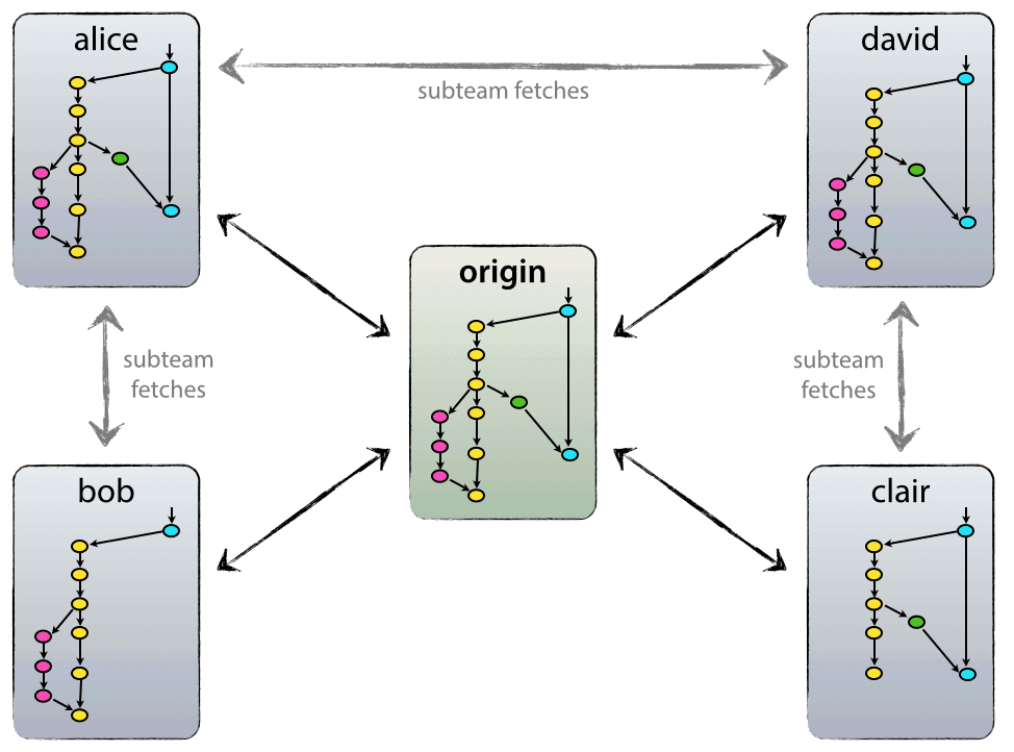
효과적인 브랜치 관리 기법으로 받아들여진 형태 : Git-flow



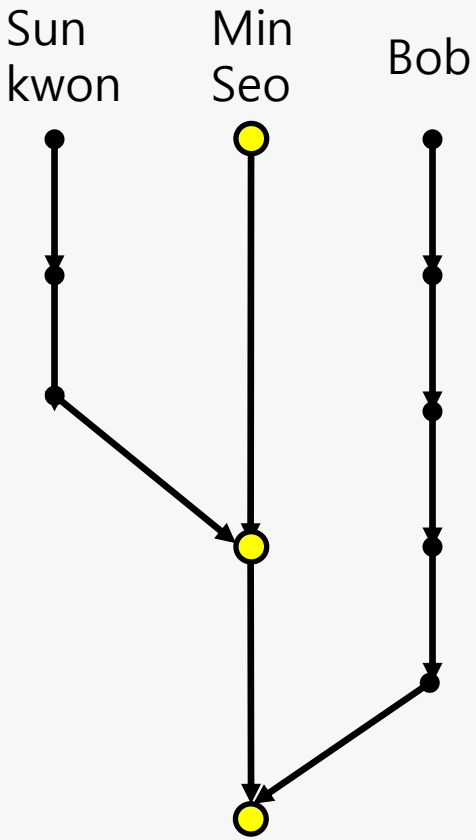
6. 브랜치 관리 기법 : Git-flow

KERI

Decentralized but centralized



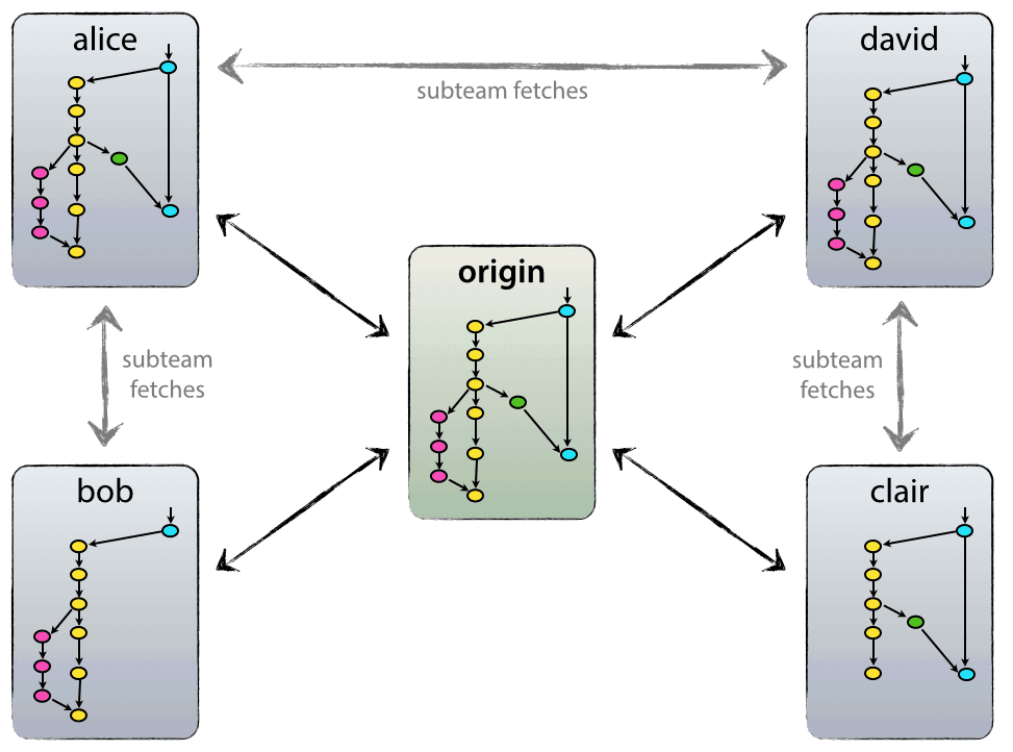
(예시) Git-flow의 정석은 아니지만...

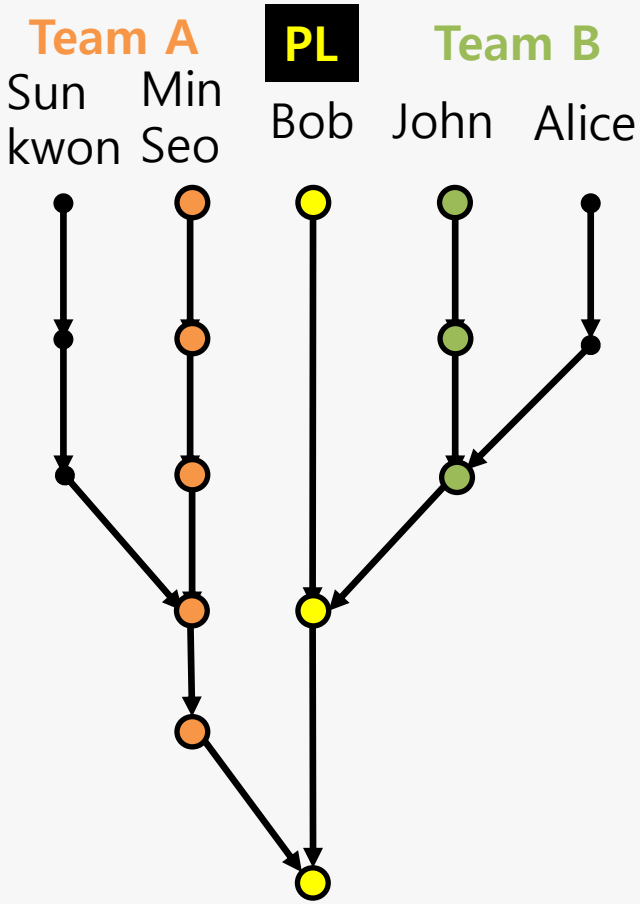


출처: A successful Git branching model (http://nvie.com/posts/a-successful-git-branching-model/)

6. 브랜치 관리 기법 : Git-flow

Decentralized but centralized

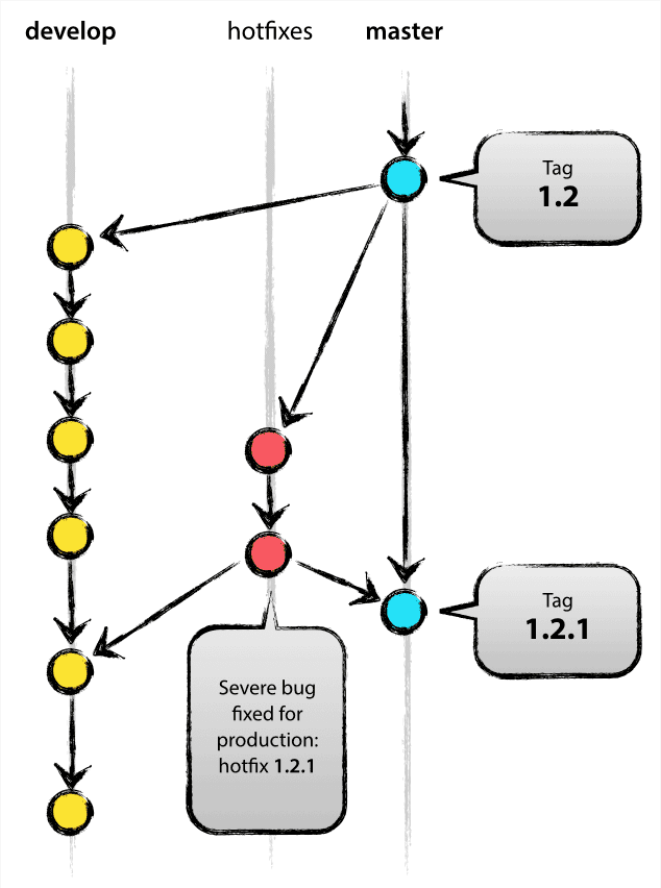




출처: A successful Git branching model (http://nvie.com/posts/a-successful-git-branching-model/)

6. 브랜치 관리 기법 : Git-flow

Hotfixes branches



버그를 수정한 내용들은 Develop 브랜치로 합치는 것이 일반적임

하지만 매우 긴급한 버그 수정 사항들(Hotfixes)은 Develop과 master 2개의 브랜치로 Merge 시키고 바로 공개(릴리즈)함

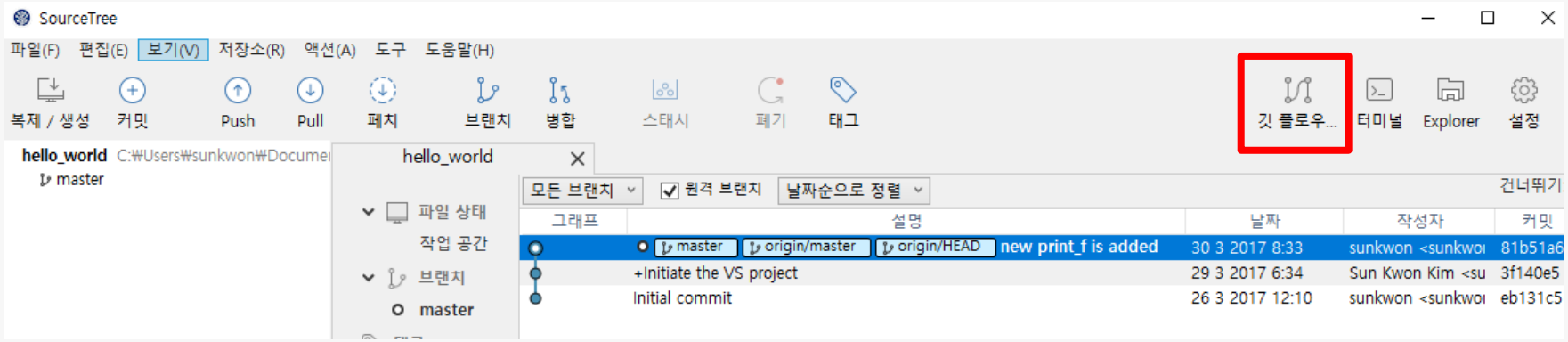
Tip. 릴리즈 버전을 매기는 규칙은 강제성이 없음. 편한대로 하면 좋지만, 일반적으로 (MajorRelease. MinorRelease. HotfixRelease)의 형태를 많이 사용함

출처: A successful Git branching model (http://nvie.com/posts/a-successful-git-branching-model/)

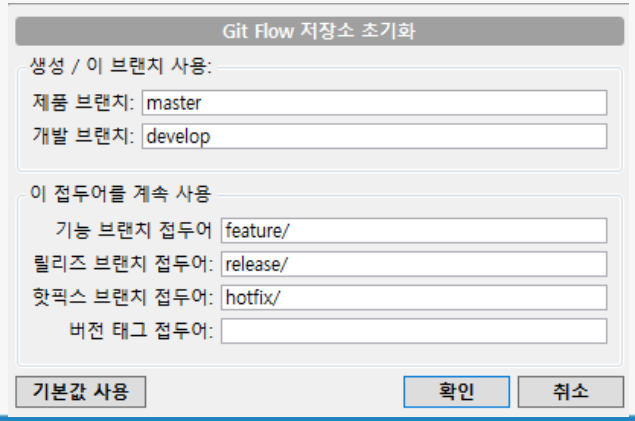
6. 브랜치 관리 기법 : Git-flow

대부분의 (모든) Git Client에서 Git-flow 관리를 도와주는 메뉴가 존재함.

1. 깃 플로우 아이콘을 선택한다.
   (또는 메뉴 → 저장소 → 깃 플로우 → 저장소 초기화를 선택한다.)



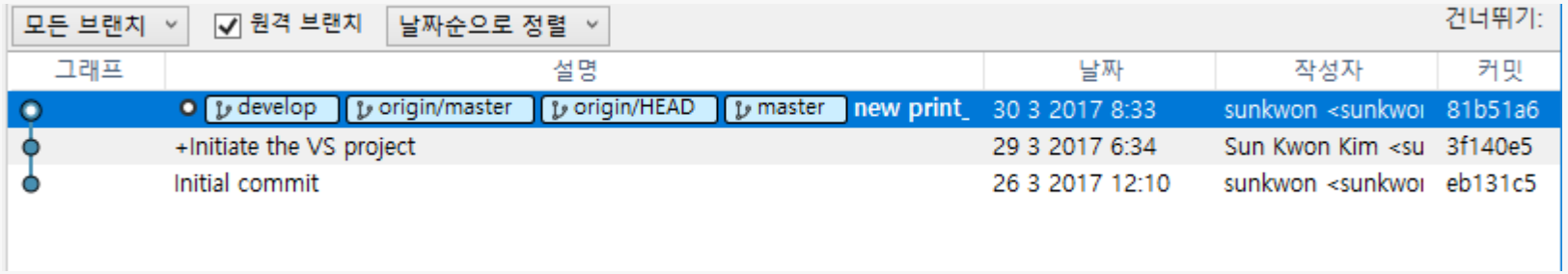
2. 확인을 버튼을 누른다.



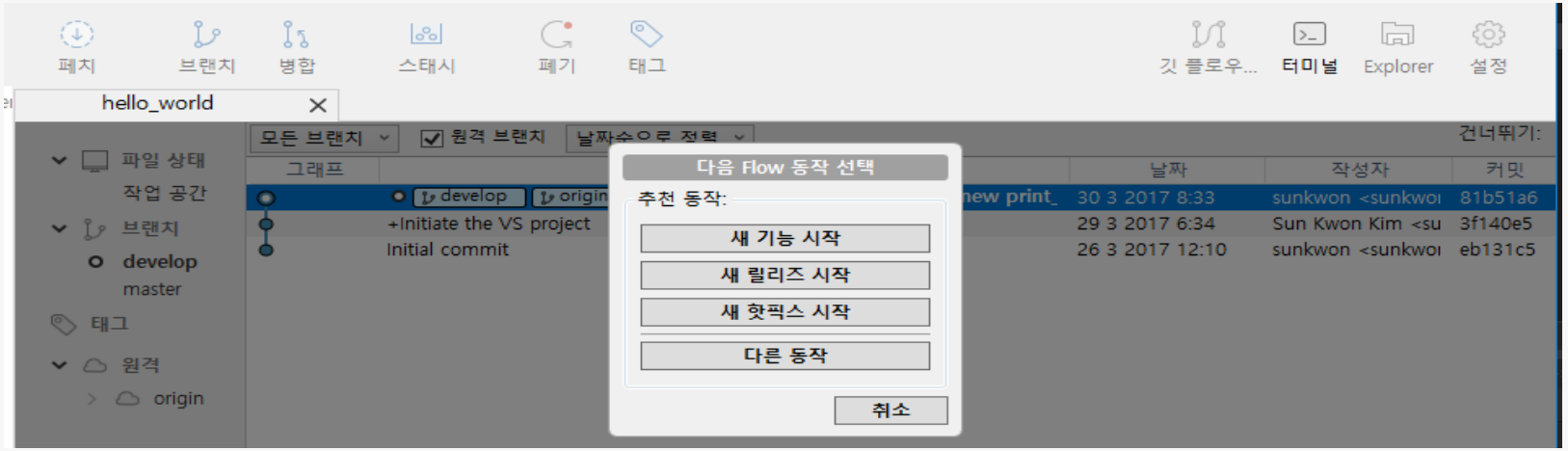
6. 브랜치 관리 기법 : Git-flow

대부분의 (모든) Git Client에서 Git-flow 관리를 도와주는 메뉴가 존재함.

3. 자동으로 Develop가 생성되었음



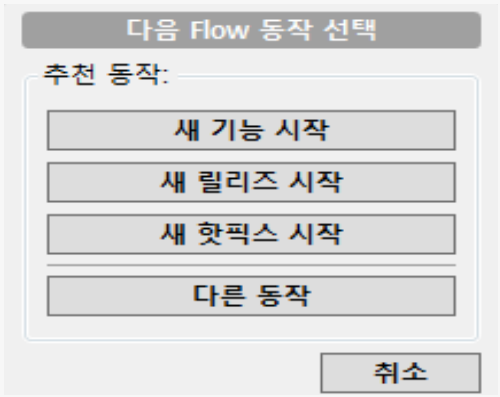
4. 다시 깃 플로우 버튼을 누르면, Git-flow 정책에 따른 메뉴가 나타남

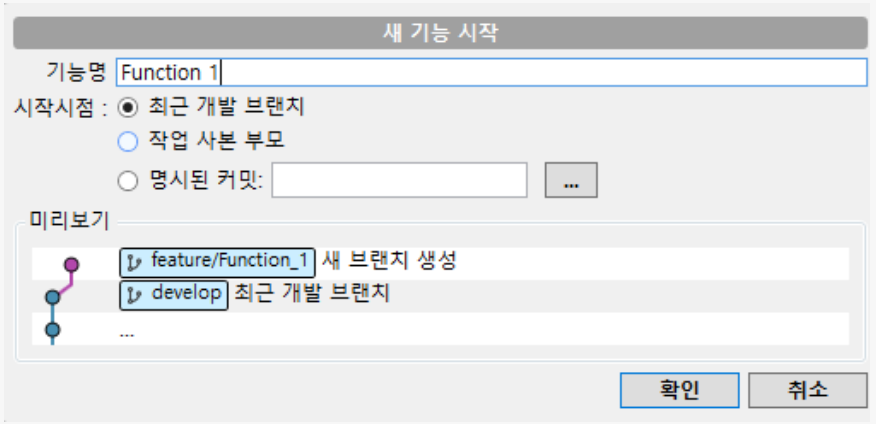


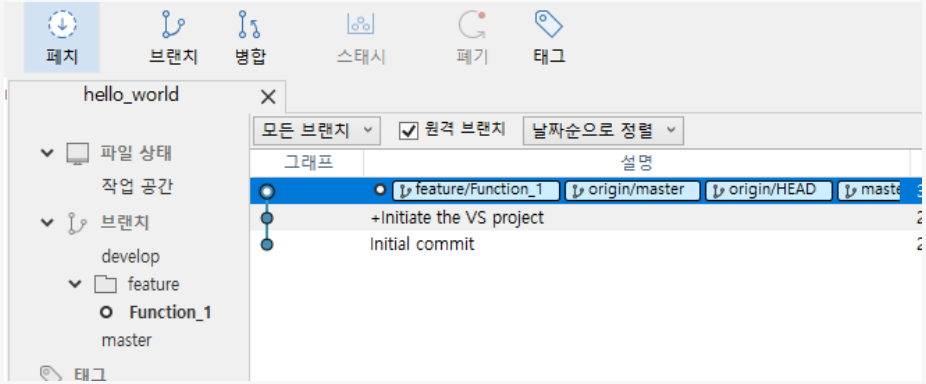
6. 브랜치 관리 기법 : Git-flow

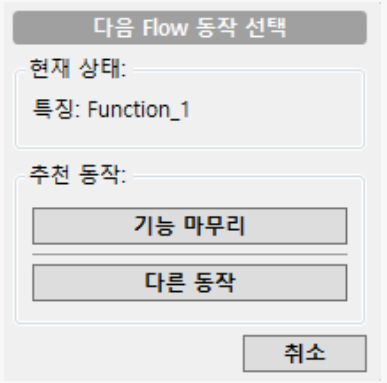
KERI

Feature branches



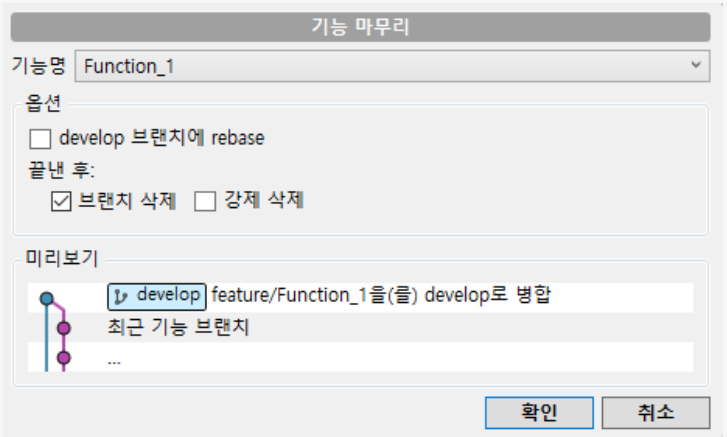


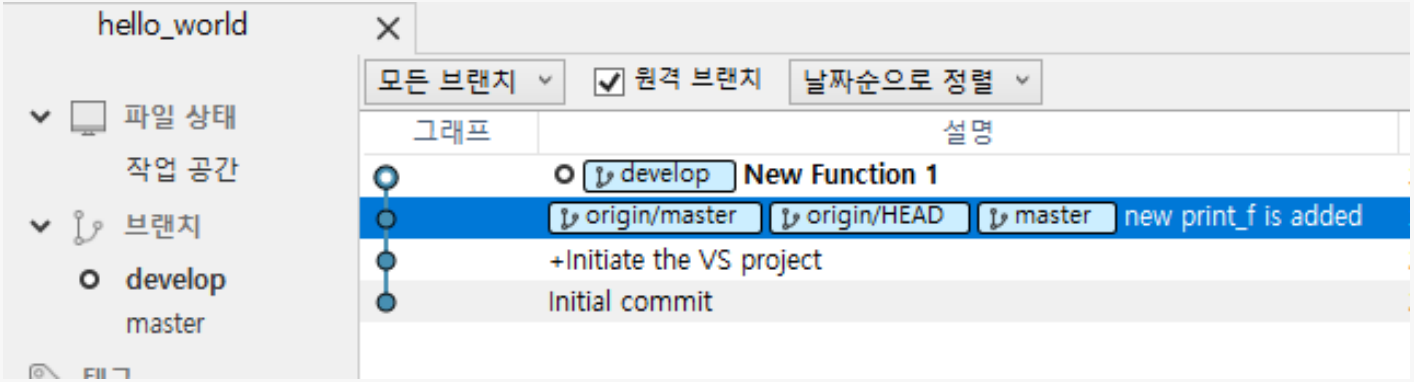




6. 브랜치 관리 기법 : Git-flow

Feature branches





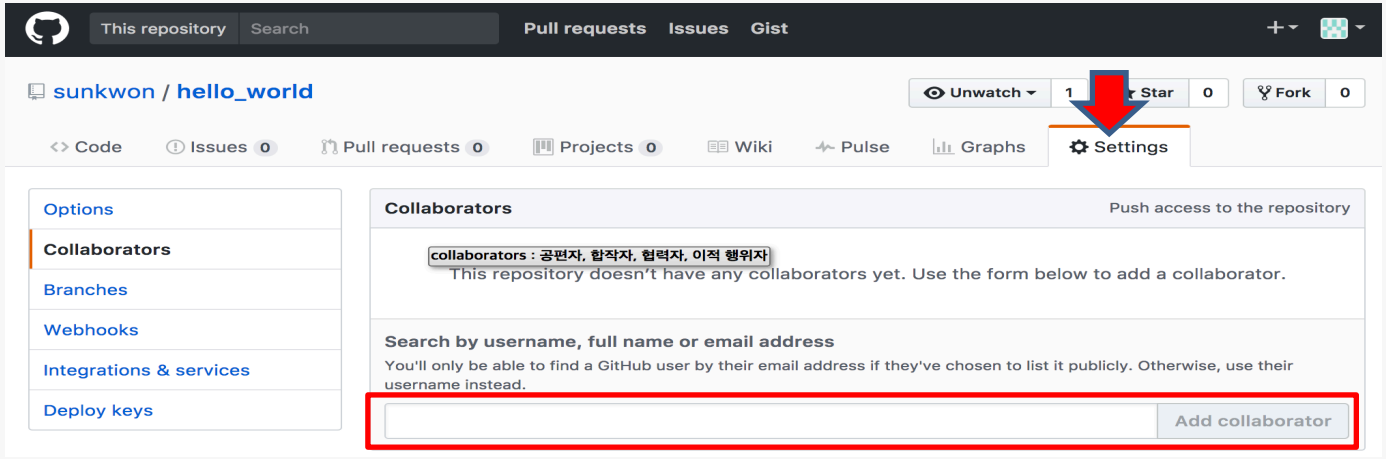
7. GITHUB

KERI
한국전기연구원

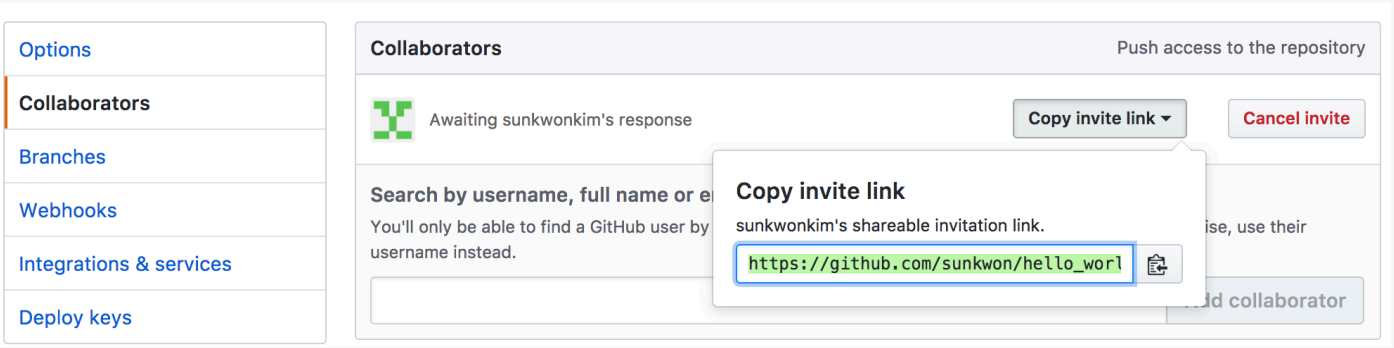
KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

7. GitHub : 협업하기

1. 생성한 프로젝트 → Settings → Collaborators → Add collaborator

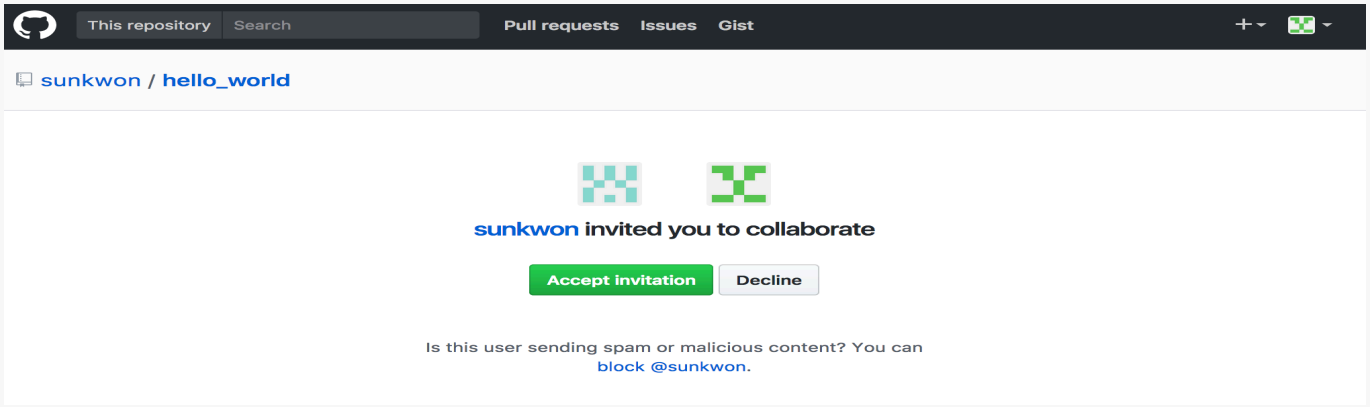


2. Invite Link 주소를 초청하고자 하는 사람에게 보내기

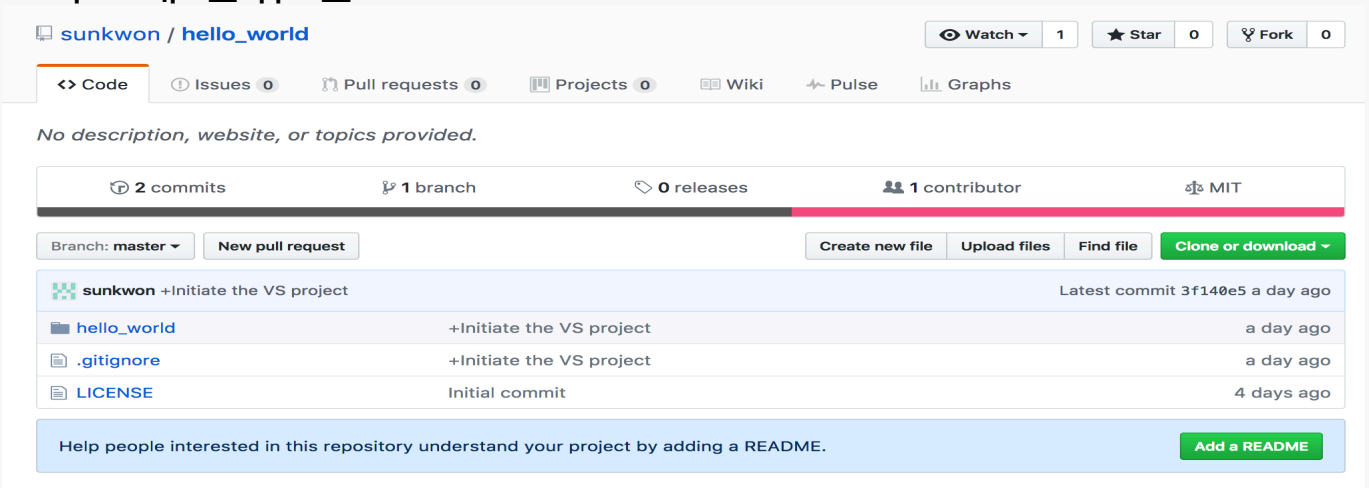


7. GitHub : 협업하기

3. 초청받은 사람이 Link 주소에 접속해서 "Accept Invitation" 버튼을 누른다.



4. 프로젝트에 합류 완료



7. GitHub : 협업하기

8. 초청받은 사람이 Git 저장소로부터 소스코드를 다운받기
(다른 환경을 가정: 리눅스 Ubuntu, Command 작업환경)

```
sunkwon@HP-Z840-Workstation: ~/Documents
sunkwon@HP-Z840-Workstation:~/Documents$ git clone https://github.com/sunkwon/hello_world.git
```

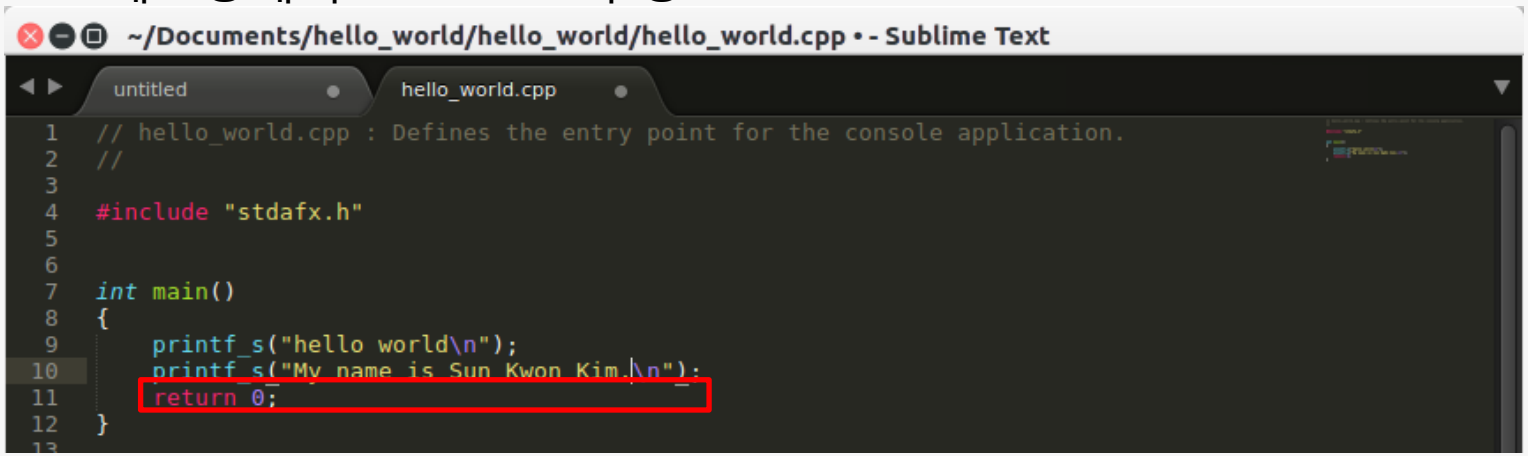
9. 소스코드 다운로드 완료

```
sunkwon@HP-Z840-Workstation: ~/Documents
sunkwon@HP-Z840-Workstation:~/Documents$ git clone https://github.com/sunkwon/hello_world.git
Cloning into 'hello_world'...
remote: Counting objects: 16, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 16 (delta 0), reused 12 (delta 0), pack-reused 0
Unpacking objects: 100% (16/16), done.
Checking connectivity... done.
sunkwon@HP-Z840-Workstation:~/Documents$
```

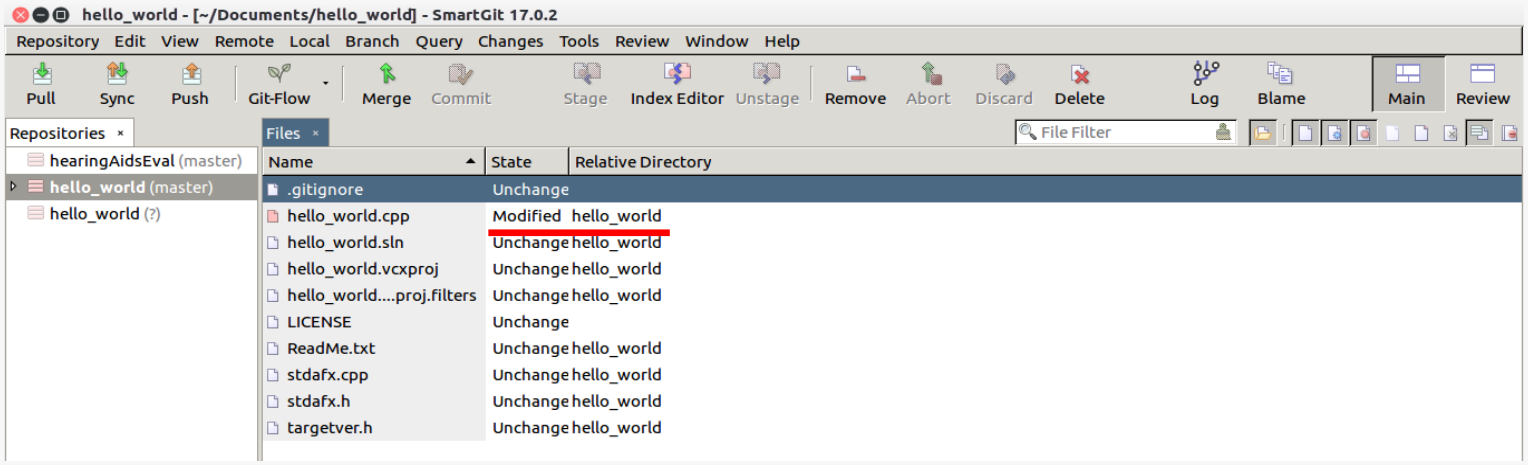
*위 작업은 source tree에서도 앞서 설명한 방식으로 할 수 있습니다. 이 예시는 다른 OS 환경에서 다른 툴로 작업하는 것을 예시로 보여드리는 것입니다.

7. GitHub : 협업하기

10. 메모장에서 소스 코드 수정

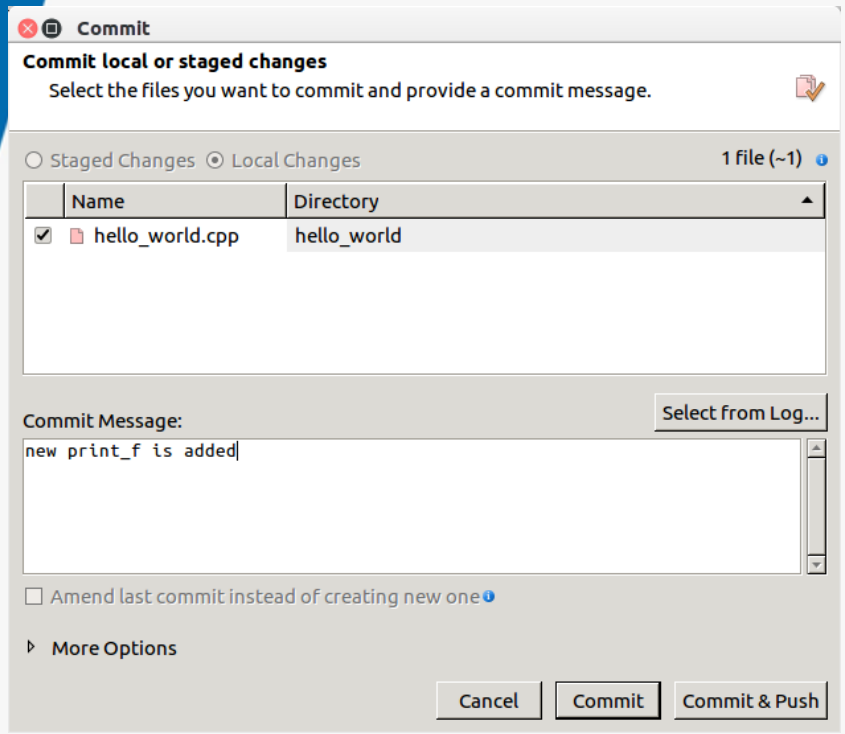


11. Git Client 에서 소스 변경을 확인

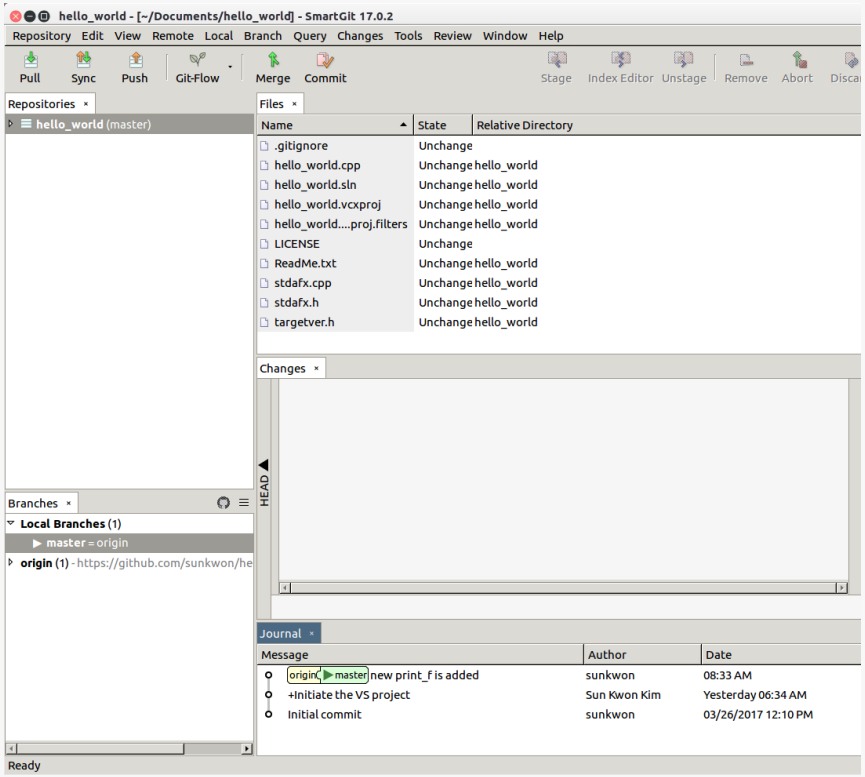


7. GitHub : 협업하기

11. Commit과 Push 하기

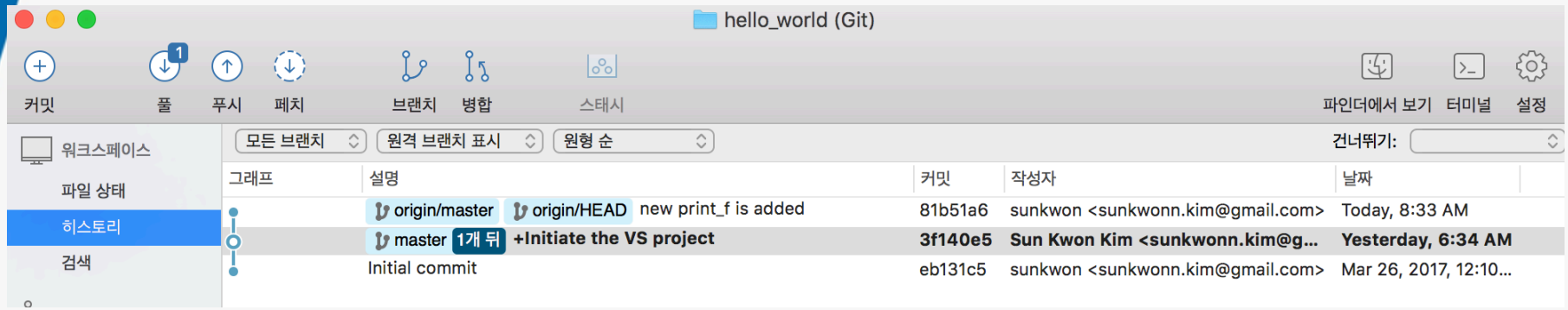


12. 새로운 history (log)가 생김

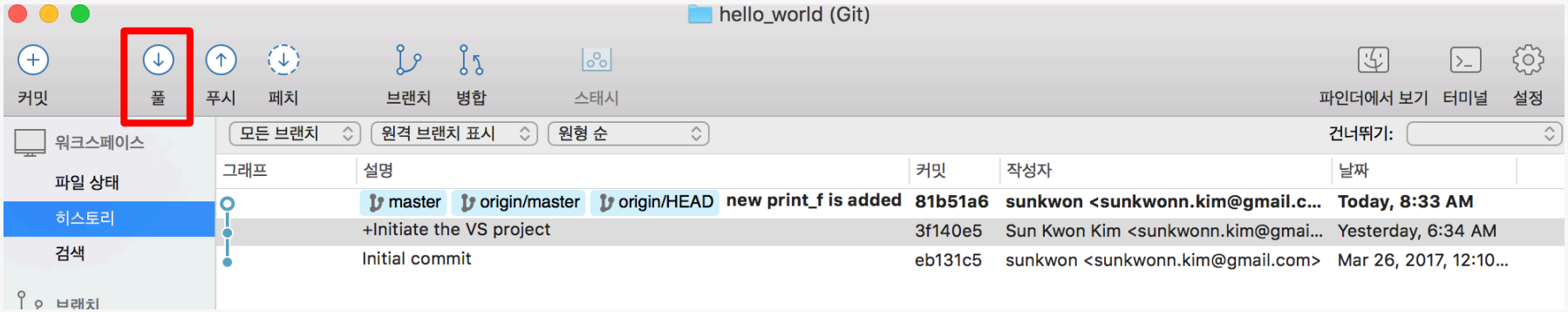


7. GitHub : 협업하기

13. 프로젝트 생성자의 Source Tree 화면에 원격 저장소("origin", GitHub)에 새로운 작업 히스토리가 생겼음을 볼 수 있음.



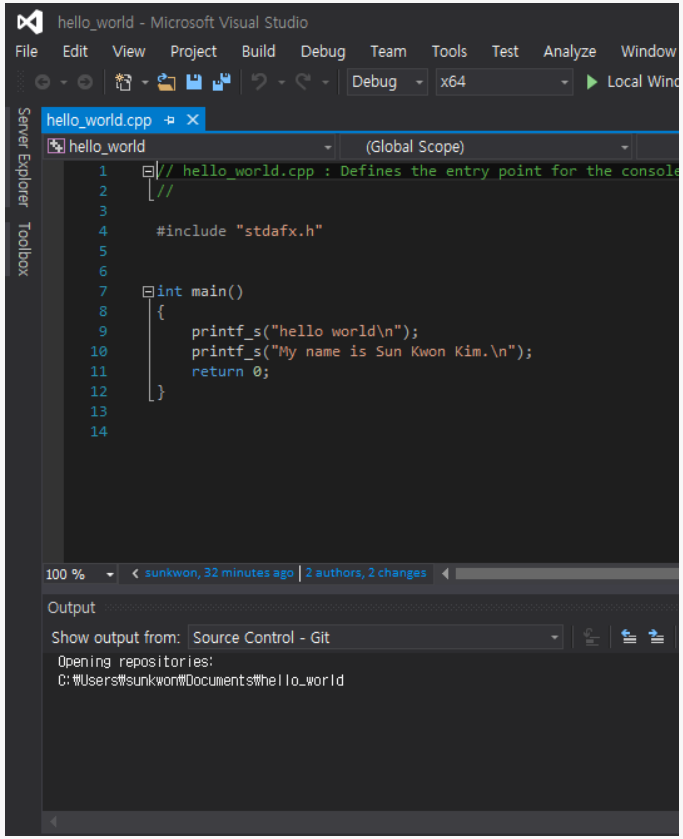
14. Pull 버튼을 눌러서 원격저장소의 새로운 내용을 다운받음



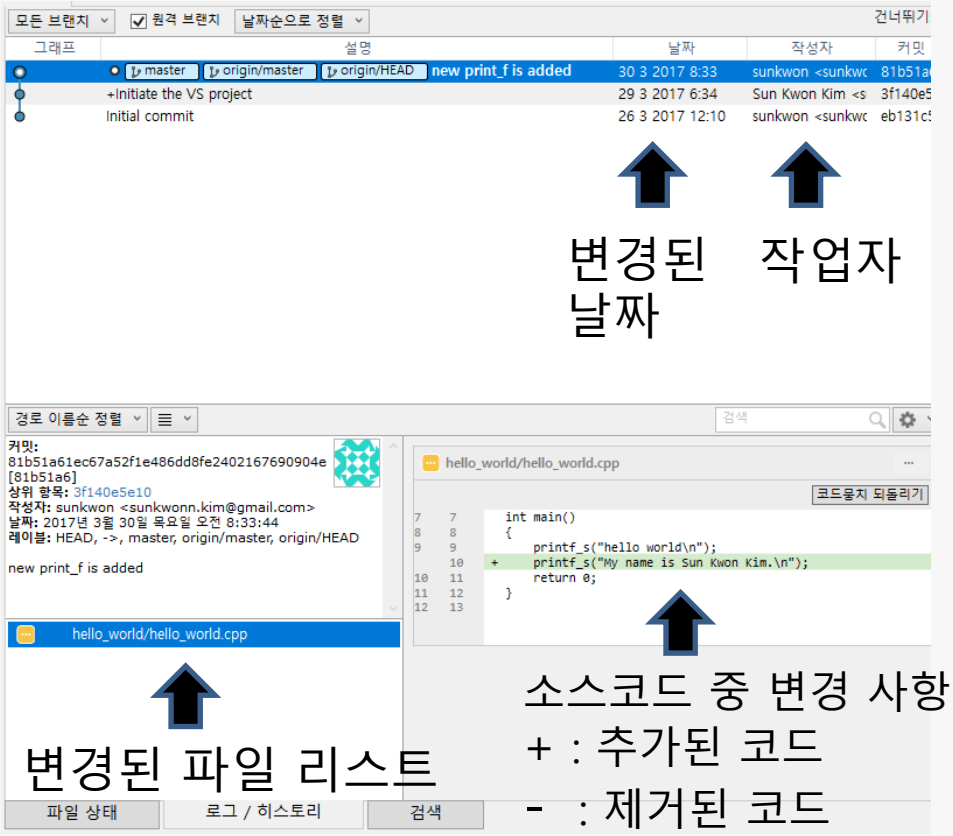
7. GitHub : 협업하기

15. 프로젝트 생성자의 작업 환경에 소스코드 변경 사항이 적용 되었음.

작업환경 (예: Visual Studio)



Git Client (예: Source Tree)



지금까지 자세히 설명하지 않은

8. 기타

KERI
한국전기연구원

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

8. 기타

- 스태시 (Stash)
  - 임시로 저장하기

- 리베이스 (Rebase)
  - 결과는 병합 (Merge)와 비슷함.
  - Merge는 합치는 2개의 작업 플로우가 그대로 남지만, Rebase는 하나의 플로우로 시간순으로 재배치하여 정리함.

8. 기타

매일 자동으로 프로젝트 실행 여부를 검사하기

Linux 예약 실행 스케쥴러: **crontab**

1. 스케쥴 목록 확인 : crontab –l
2. 스케쥴 등록/수정/확인: crontab -e

```
  .---------------- 분 - 범위 : 0-59
  | .------------- 시 - 범위 : 0-23 (자정이 0)
  | |  .---------- 일 - 범위 : 1-31
  | |  |  .------- 달 - 범위 : 1-12 또는 jan,feb, mar, apr ... 축약형 영문 월 표시
  | |  |  | .---  요일 - 범위 : 0-6 (일요일은 0 또는 7) 또는 sun, mon, tue, wed thu, fri, sat
  | |  |  | |
# *  *  *  *  * [실행될 명령어]
```

3. 스케쥴 서비스 시작(재실행): service cron restart

8. 기타

/etc/test_project.sh 파일 생성

```
cd ~/(Git 작업 디렉토리)
Git pull
make
./run_test
```

Crontab 서비스에 매일 오전 1시에 test_project.sh 파일을 실행하도록 등록

```
crontab –e
* 1 * * * sh ~/test_project.sh
```

감사합니다.

가치있는 대형 연구성과 창출의 산실

KOREA ELECTROTECHNOLOGY RESEARCH INSTITUTE

KERI