세미나 - Git

이병현 연구원

2025 / 09 / 26

Agenda

- 코드 형상 관리의 목적
- Git 핵심 명령어
- SVN vs Git
- 원격저장소(GitHub)

코드 형상 관리의 목적

코드 형상 관리의 목적

- **코드 형상 관리 : 소프트웨어 개발 과정**에서 발생하는 **소스 코드, 문서 등 결과물의 버전을 체계적으로 추적하고 통제**하여 일관성과 추적 가능성을 보장하는 활동

- **프로젝트의 변경 사항을 기록하는 '디지털 작업 일지'**

- **원격 저장소 : 코드를 공유하고 백업하기 위한 인터넷상의 안전한 '공용 작업 폴더'**
  - **GitHub, GitLab 등**

코드 형상 관리의 목적

- 코드를 잘못 저장하였을 때
- config.txt를 잘못 수정한 상황
- Git 을 이용하여 복구

```
PS C:\Users\leebh\my_project> echo  'OOPS! CONFIG DELETED!' > config.txt
PS C:\Users\leebh\my_project> git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   config.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\leebh\my_project> type .\config.txt
OOPS! CONFIG DELETED!
PS C:\Users\leebh\my_project> git restore config.txt
PS C:\Users\leebh\my_project> git status
On branch main
nothing to commit, working tree clean
PS C:\Users\leebh\my_project> type .\config.txt
DB_HOST=127.0.0.1
PS C:\Users\leebh\my_project>
```

코드 형상 관리의 목적

- 2명 이상이 코드를 수정해 충돌이 날 때
- 같은 코드를 A와 B가 수정
- Git에서 수정 과정에 충돌이 있음을 알림

A

B

A

B

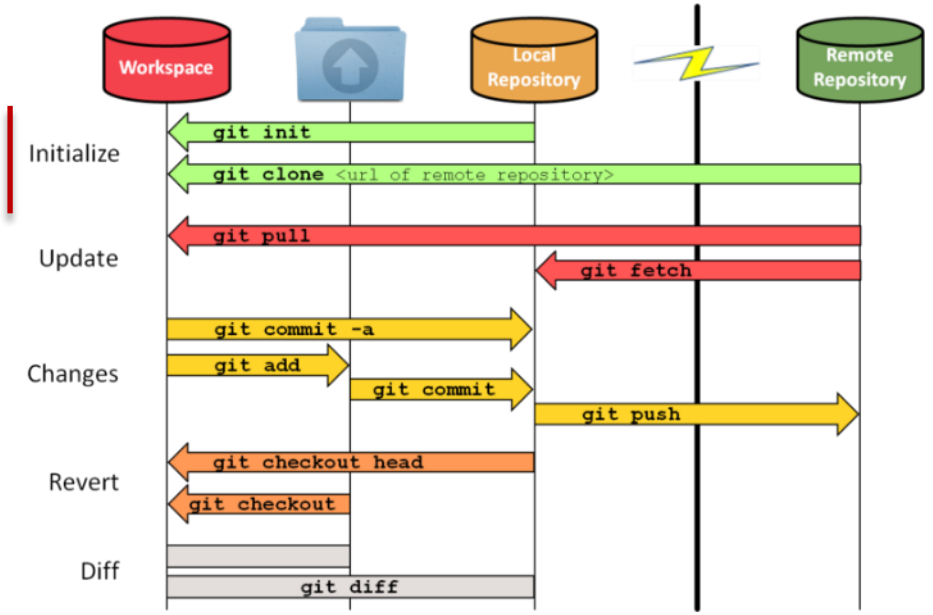
```
PS C:\Users\leebh\git_test> git init
Initialized empty Git repository in C:/Users/leebh/git_test/.git/
PS C:\Users\leebh\git_test> echo public class UserService { } > UserService.java
PS C:\Users\leebh\git_test> git add .
PS C:\Users\leebh\git_test> git commit -m "Create UserService base class"
[main (root-commit) b44c53c] Create UserService base class
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 UserService.java
PS C:\Users\leebh\git_test> git switch -c feature/login
Switched to a new branch 'feature/login'
PS C:\Users\leebh\git_test> git add .
PS C:\Users\leebh\git_test> git commit -m "Feat: Implement login function"
[feature/login 8f7f43b] Feat: Implement login function
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\leebh\git_test> git switch main
Switched to branch 'main'
PS C:\Users\leebh\git_test> git switch -c feature/signup
Switched to a new branch 'feature/signup'
PS C:\Users\leebh\git_test> git add .
PS C:\Users\leebh\git_test> git commit -m "Feat: Implement signup function"
[feature/signup 3d576dd] Feat: Implement signup function
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\leebh\git_test> git switch main
Switched to branch 'main'
PS C:\Users\leebh\git_test> git merge feature/login
Updating b44c53c..8f7f43b
Fast-forward
 UserService.java | Bin 64 -> 172 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\leebh\git_test> git merge feature/signup
warning: Cannot merge binary files: UserService.java (HEAD vs. feature/signup)
Auto-merging UserService.java
CONFLICT (content): Merge conflict in UserService.java
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\leebh\git_test>
```

Git 핵심 명령어

Git 핵심 명령어 - Initialize

- **git init**
  - 현재 폴더를 Git이 관리하는 공간(로컬 저장소)으로 만듭니다. 프로젝트 시작 시 최초 한 번만 사용

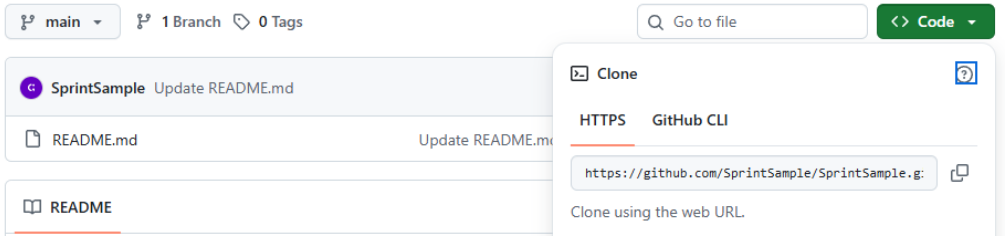
```
leebh@DESKTOP-TQ1GMAN MINGW64 ~
$ cd a_project/

leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git init
Initialized empty Git repository in C:/Users/leebh/a_project/.git/
```



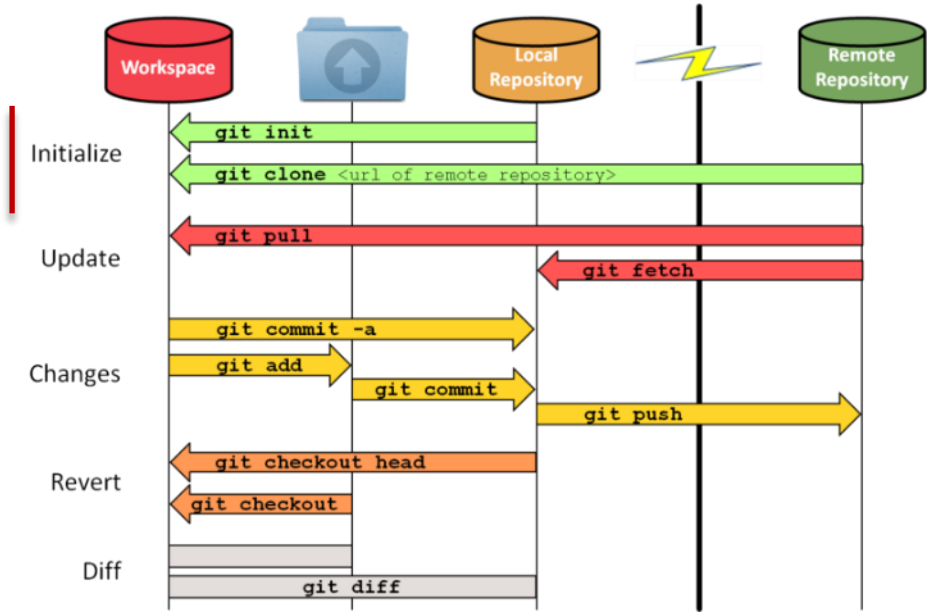
  - .git 폴더 : 숨겨진 폴더로 생성되고 프로젝트 기록과 정보를 담고 있는 핵심 데이터베이스

Git 핵심 명령어 - Initialize

- **git clone [URL]**
  - GitHub 같은 원격 저장소에 있는 프로젝트를 내 컴퓨터로 그대로 복제해 가져와 사용




```
leebh@DESKTOP-TQ1GMAN MINGW64 ~
$ git clone https://github.com/SprintSample/SprintSample.git
Cloning into 'SprintSample'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
```

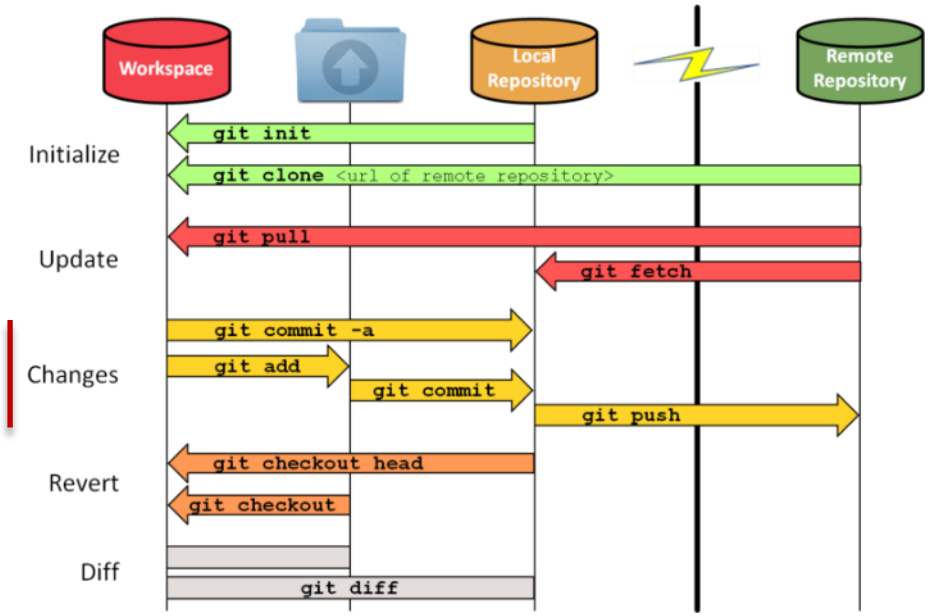


Git 핵심 명령어 – Changes

- git add [파일명]
  - 새로 작성 혹은 수정한 파일의 변경 사항을 저장할 준비가 됐다고 알림
  - [파일명] 대신에 .을 할 경우 폴더 내 모든 파일 add

* git status
  - 현재 작업 폴더의 상태를 보여줍니다. 어떤 파일이 수정됐는지, 어떤 파일이 add 되었는지 등 모든 상황을 알려주는 명령어

```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git add .

leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   project_1.txt
```



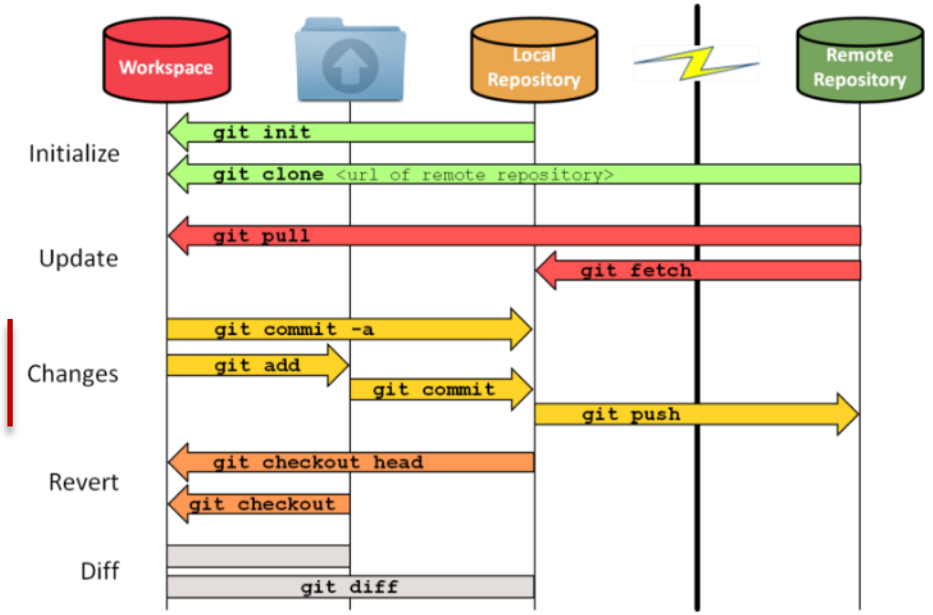
- 빈 폴더에서 project_1.txt 생성 후 add 하여 저장할 준비 완료, status를 통해 확인

Git 핵심 명령어 – Changes

- git commit –m “커밋 메시지 "
  - add 한 파일들을 하나의 의미 있는버전(커밋)으로 저장
  - -a 옵션을 추가해 add 없이 바로 커밋

- add한 project_1.txt 를 commit하여 기록

```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   project_1.txt


leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git commit -m "my first commit"
[main (root-commit) 4630375] my first commit
 1 file changed, 1 insertion(+)
 create mode 100644 project_1.txt

leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git status
On branch main
nothing to commit, working tree clean
```



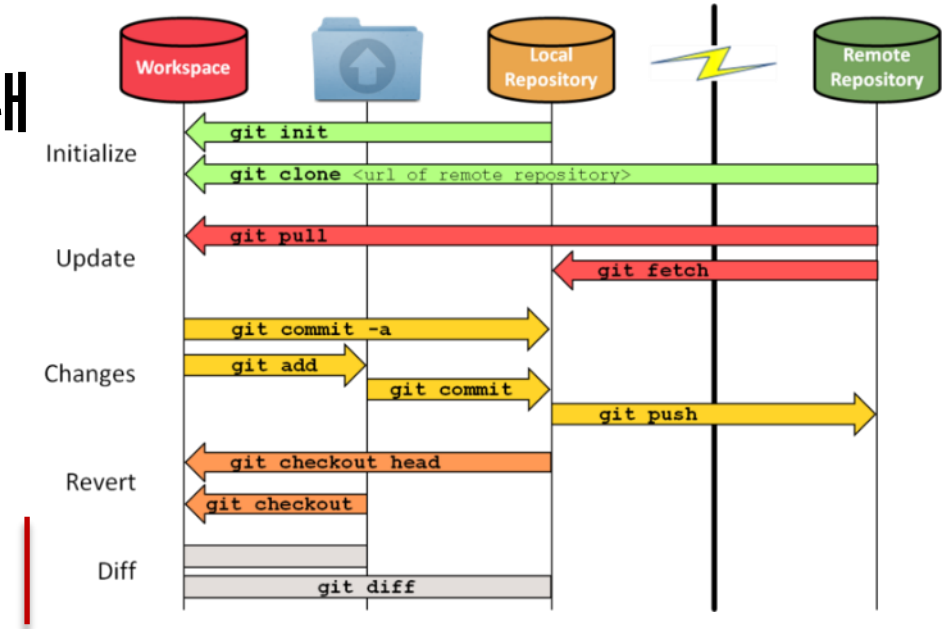
Git 핵심 명령어 – Diff

- **git diff**
  - add 하지 않은 변경 사항이 원본과 다른 부분을 보여줌

- **project_1.txt 내용을 수정하고 add하지 않은 상태**

```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git diff
diff --git a/project_1.txt b/project_1.txt
index b6fc4c6..fabfbb2 100644
--- a/project_1.txt
+++ b/project_1.txt
@@ -1 +1 @@
-hello
\ No newline at end of file
+hi hello
\ No newline at end of file
```

이전 내용과
수정한 내용

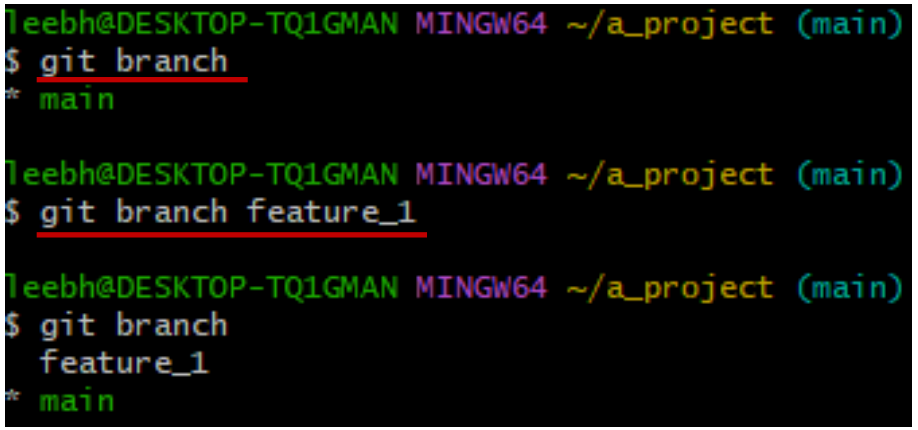


Git 핵심 명령어 – Branch

- git branch
  - 브랜치 목록을 보여주고 명령어 뒤에 [브랜치명]을 적으면 새 브런치 생성

- 브랜치 생성 및 브랜치 목록을 가져옴

- git branch –d [브랜치] : 브랜치 삭제



※ 주의 사항

1. 충돌 발생을 줄이기 위해 항상
   main 브랜치에서 시작
2. 브랜치는 작고 명확하게 유지
   브랜치 하나당 기능 하나만 수정하여 코드 리뷰가 쉽고 원인을 찾기 쉬움

Git 핵심 명령어 – Branch

- git switch [브랜치명]
  - 원하는 브랜치로 작업 공간을 전환
  - git switch –c [브랜치명]으로 적을 경우 생성과 동시에 이동

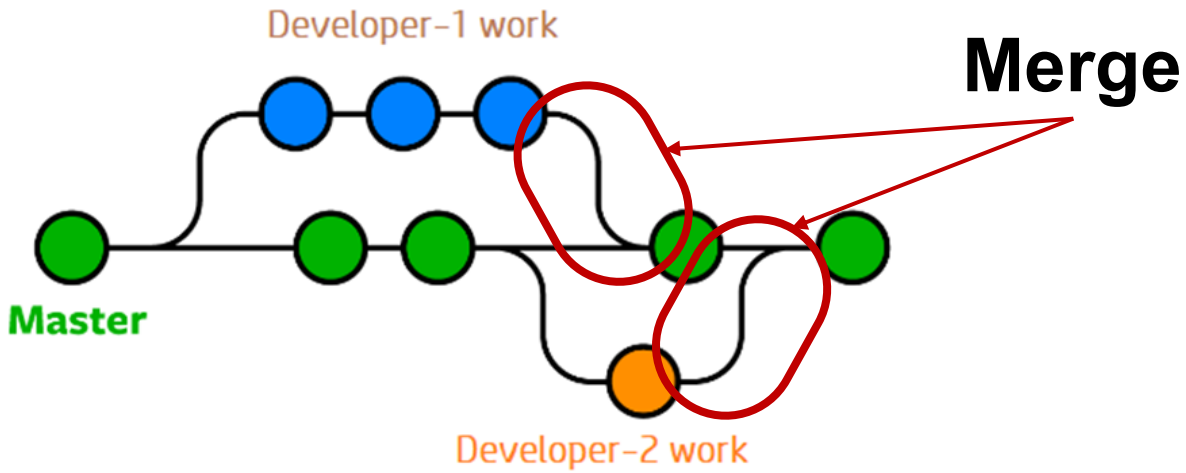
- 이전에 만든 feature_1 브랜치로 작업 공간 전환
  feature_2 브랜치를 생성과 동시에 이동

```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git switch feature_1
M       project_1.txt
Switched to branch 'feature_1'

leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (feature_1)
$ git branch
* feature_1
  main
```

```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (feature_1)
$ git switch -c feature_2
Switched to a new branch 'feature_2'

leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (feature_2)
$ git branch
  feature_1
* feature_2
  main
```

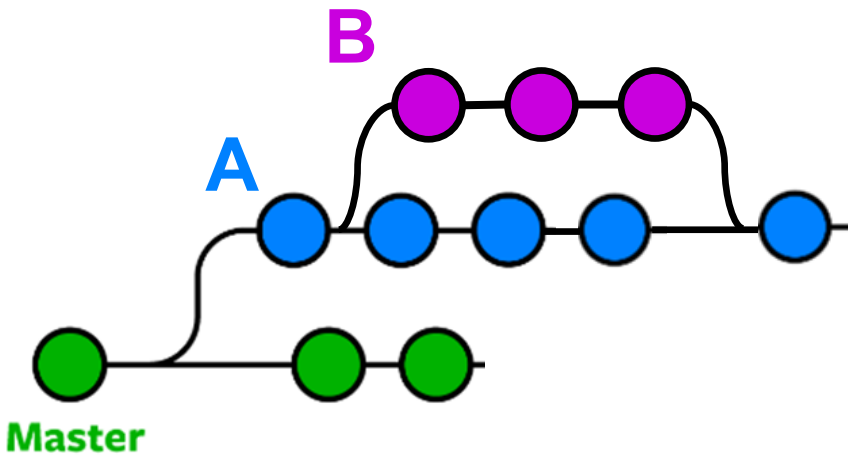
Git 핵심 명령어 – Branch

- git merge [브랜치명]
  - 현재 브랜치에 다른 브랜치의 작업 내용을 병합

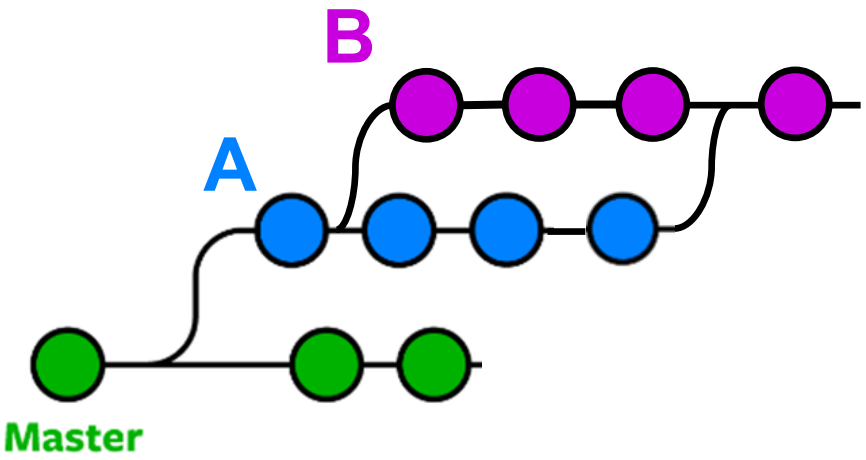


Git 핵심 명령어 – Branch

(A Branch에서) git merge B



(B Branch에서) git merge A



- 내가 현재 있는 Branch 로 다른 Branch 의 내용을 가져와 병합
- 현재 Branch 는 업데이트가 되고 다른 Branch 는 그대로

Git 핵심 명령어 – Branch

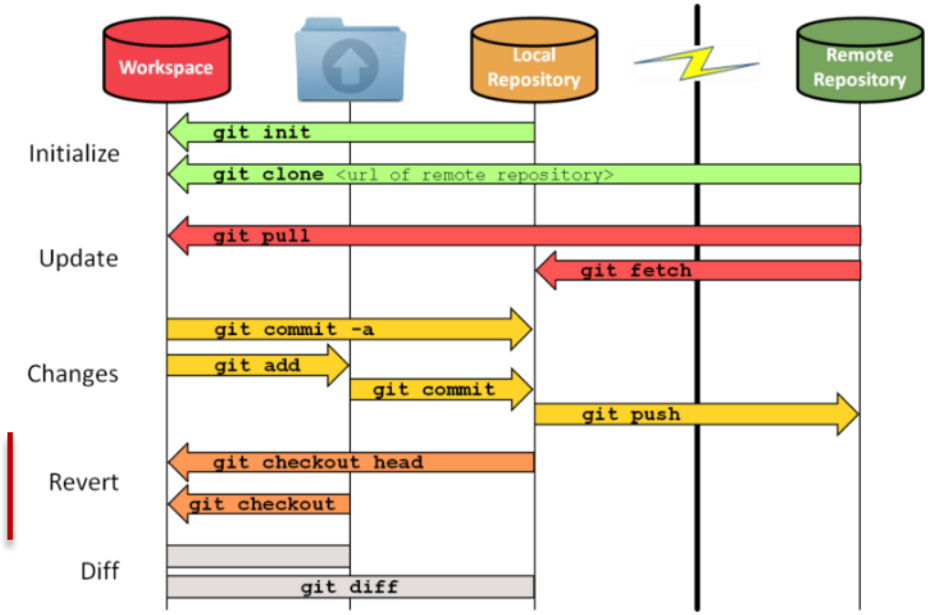
- git checkout [브랜치명]
  - switch와 같은 동작을 하는 명령어로, 브랜치를 전환

```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git checkout feature_1
Switched to branch 'feature_1'

leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (feature_1)
```

git switch VS git checkout

- git switch (브랜치 전환)
  - 브랜치 전환 : git switch [브랜치명]
  - 브랜치 생성 + 전환 : git switch –c [브랜치명]

- git checkout (파일 복원)
  - 브랜치 전환 : git checkout [브랜치명]
  - 브랜치 생성 + 전환 : git checkout –b [브랜치명]
  - 파일 복원 : git checkout -- [파일명]
  - 과거 시점 이동 : git checkout [커밋ID]
  - 브랜치에 있는 파일만 가져오기 :
    git checkout [브랜치명] – [파일명]



- switch는 브랜치 전환 명령어라면 checkout은 명령어는 여러가지를 수행

Git 핵심 명령어 – log

- git log
  - 그동안의 git 이력을 확인
  - --graph
    - graph 형태로 확인
  - --oneline
    - 한 줄 요약으로 간단하게 확인

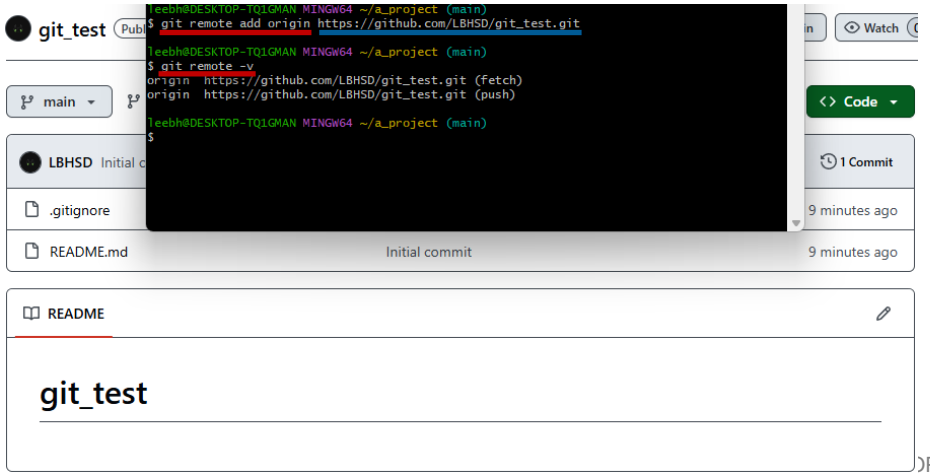
```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git log --graph
*   commit 4e3d34e591812faf901fbb132a6d0ac9aa2a0dbd (HEAD -> main, origin/main)
|\  Merge: edd1876 80f978e
| | Author: LBHSD <qw59slfre@naver.com>
| | Date:   Sun Sep 21 15:23:00 2025 +0900
| |
| |     Merge branch 'main' of https://github.com/LBHSD/git_test
| |
| * commit 80f978e27d6df7de2383e249c45ae1d6b0404436
|   Author: Byunghyun Lee <103364884+LBHSD@users.noreply.github.com>
|   Date:   Thu Sep 18 15:14:41 2025 +0900
|
|       Initial commit
|
* commit edd1876aaea14d18eebea320695f6c334b81965b (feature_1)
| Author: LBHSD <qw59slfre@naver.com>
| Date:   Sun Sep 21 14:45:17 2025 +0900
|
|     feature_1 branch commit
|
* commit ade16e1f6d79202b32f491bc1df8bd59cd4d8fdd (feature_2)
  Author: LBHSD <qw59slfre@naver.com>
  Date:   Sun Sep 21 14:42:58 2025 +0900

      my first commit
```

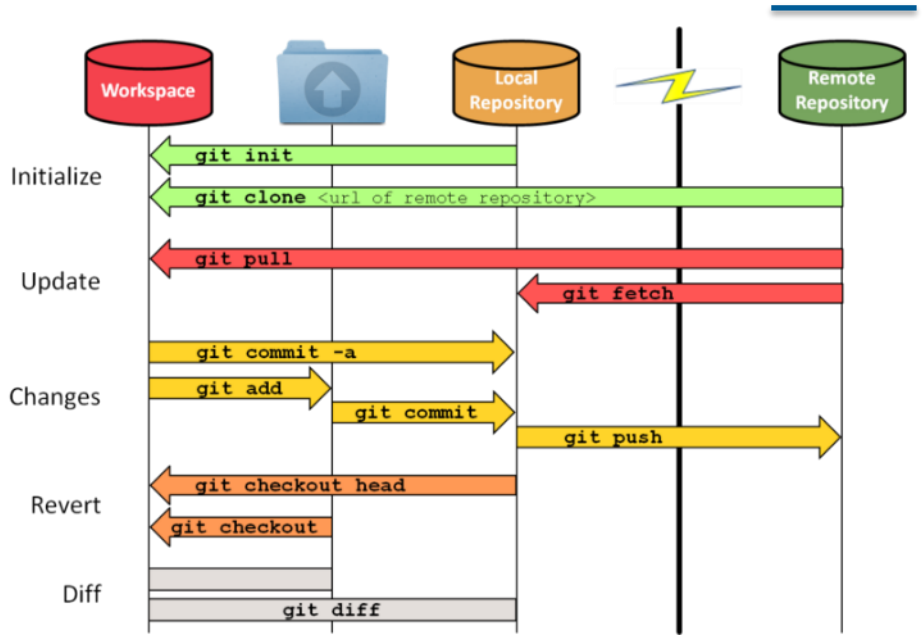
```
leebh@DESKTOP-TQ1GMAN MINGW64 ~/a_project (main)
$ git log --oneline
4e3d34e (HEAD -> main, origin/main) Merge branch 'main' of https://github.com/LB
HSD/git_test
edd1876 (feature_1) feature_1 branch commit
ade16e1 (feature_2) my first commit
80f978e Initial commit
```

Git 핵심 명령어 – Remote

- git remote add origin [원격 저장소 URL 주소]
  - git init으로 만든 저장소를 원격 저장소와 연결
  - origin은 원격 저장소 별명으로 보통 origin을 사용

- git remote –v
  - 현재 프로젝트에 연결된 원격 저장소의 주소 확인

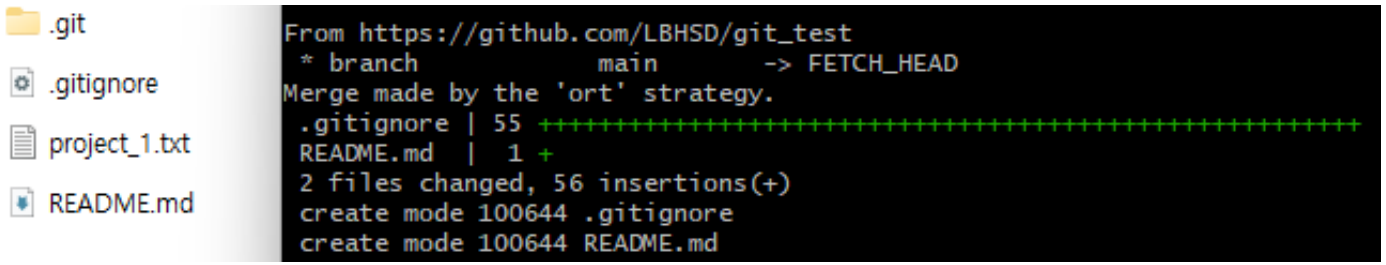


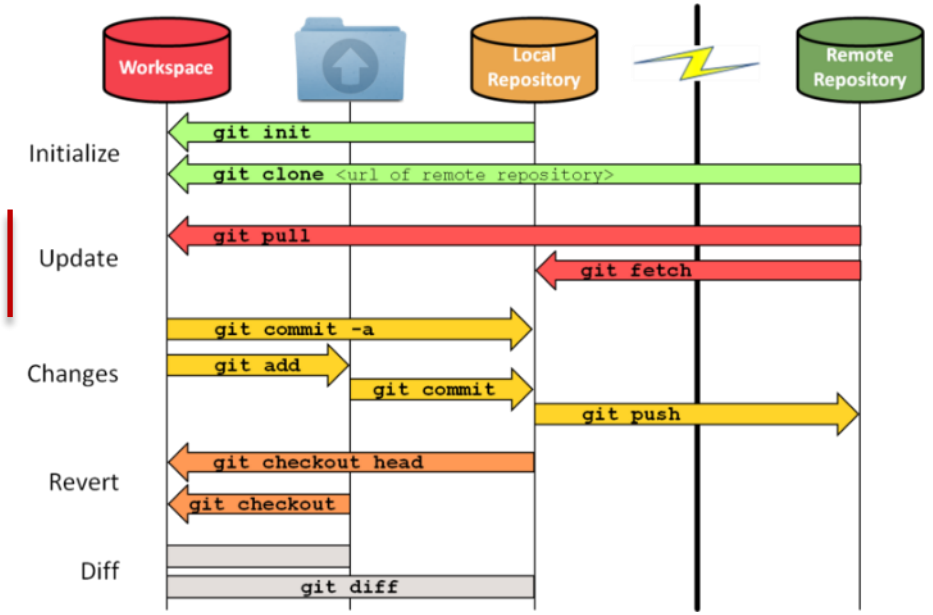


Git 핵심 명령어 – Update

- git pull [원격저장소명] [브랜치명]
  - 원격 저장소의 최신 내용을 컴퓨터로 가져와서 자동으로 합침

- git fetch
  - pull과 비슷하지만, 내용을 가져오기만 하고 자동으로 합치지는 않음.
  - 이후 git merge origin/main을 통해 최신 내용을 컴퓨터로 합침



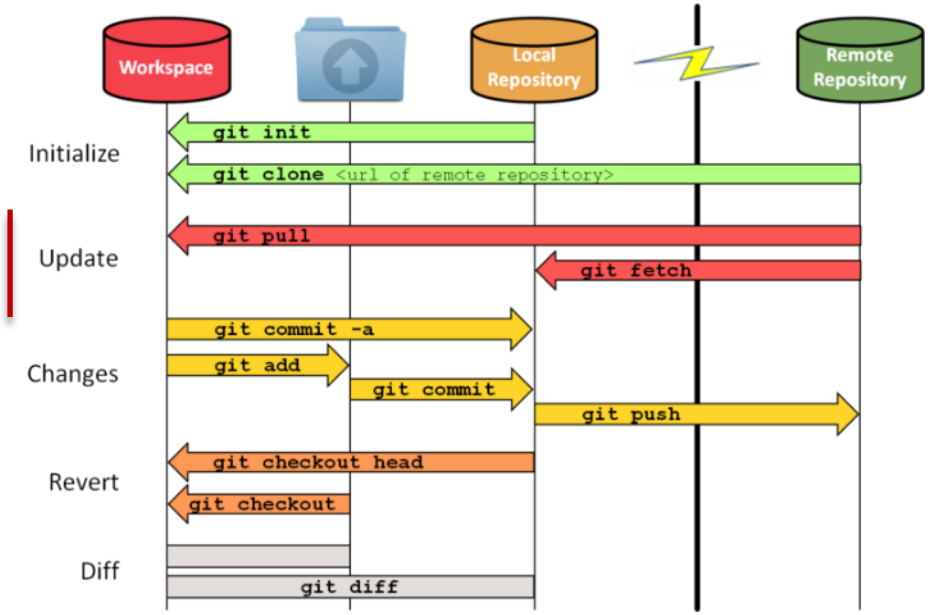


git pull VS git fetch

```
diff --git a/README.md b/README.md
new file mode 100644
index 0000000..f9894f4
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# git_test
\ No newline at end of file
diff --git a/project_1.txt b/project_1.txt
deleted file mode 100644
index e64a78a..0000000
--- a/project_1.txt
+++ /dev/null
@@ -1 +0,0 @@
-hello branch
```

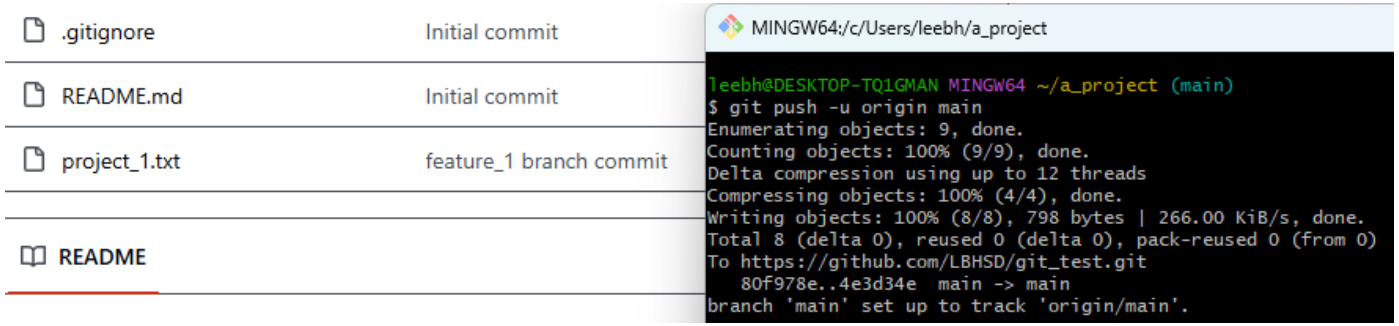
- git pull [원격저장소명] [브랜치명]
  - 원격저장소의 파일 내용을 입력한 브랜치에 즉시 병합

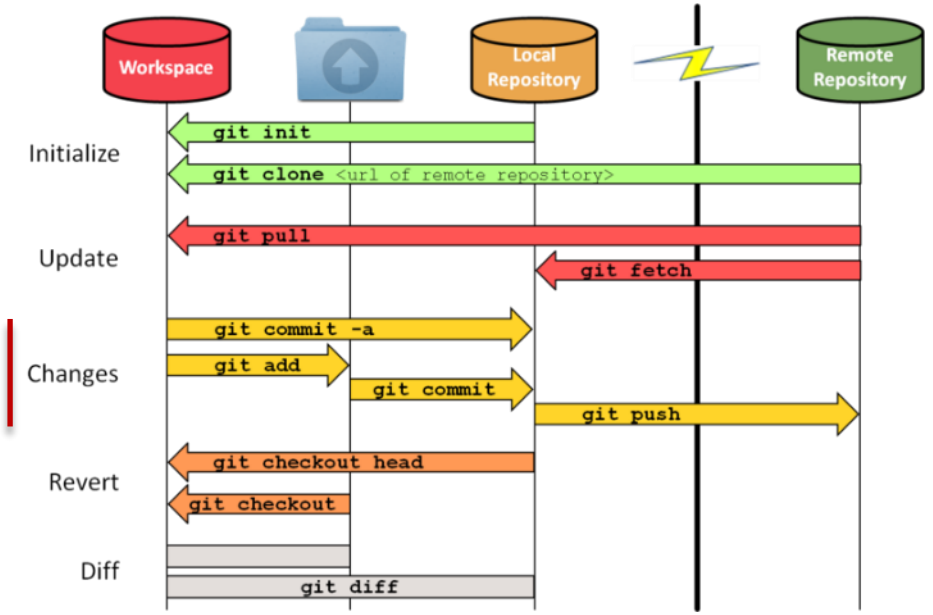
- git fetch
  - 내용을 합치지는 않으나 로컬 git 자체에 원격저장소 파일을 최신으로 업데이트
  - 대신에
    git show origin/main:[파일] 을 통해 파일의 내용 확인 가능
    git diff main origin.main 을 통해 차이점 비교
    git merge origin/main 을 통해 git pull과 같은 행동 수행



Git 핵심 명령어 – Changes

- git push -u [원격저장소명] [브랜치명]
  - 커밋한 내용을 원격 저장소에 업로드
  - -u 옵션을 추가하여 다음부터는 git push 만 해도 알아서 [원격저장소명] [브랜치명]으로 전달
  - 주의할 점은 여기서 브랜치명은 원격저장소의 브랜치



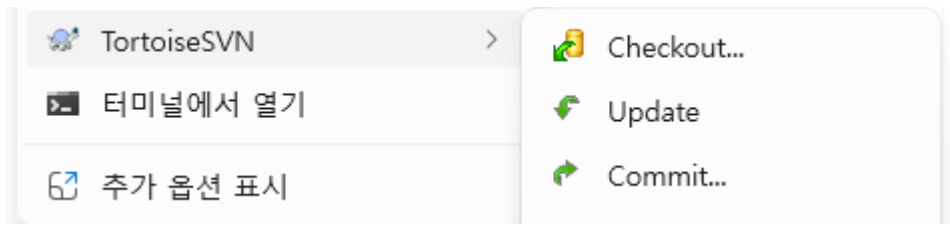


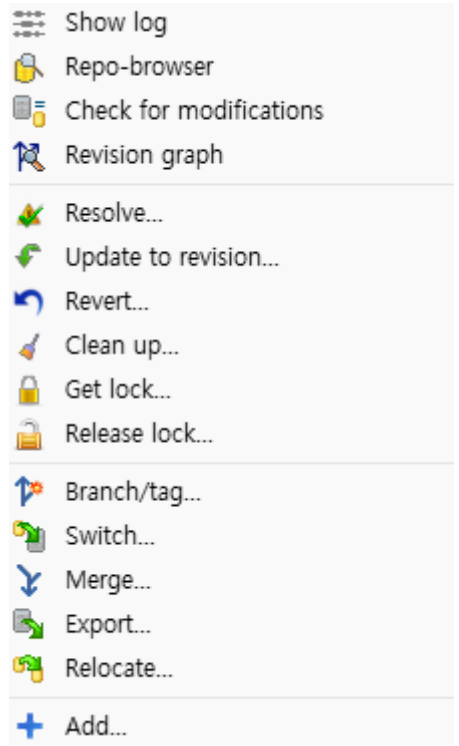
SVN vs Git

SVN vs Git – 유사기능

- Checkout = git clone
- Update = git pull

- Add = git add
- Commit = git commit
- Branch = git branch
- Switch = git switch
- Merge = git merge
- Show log = git log

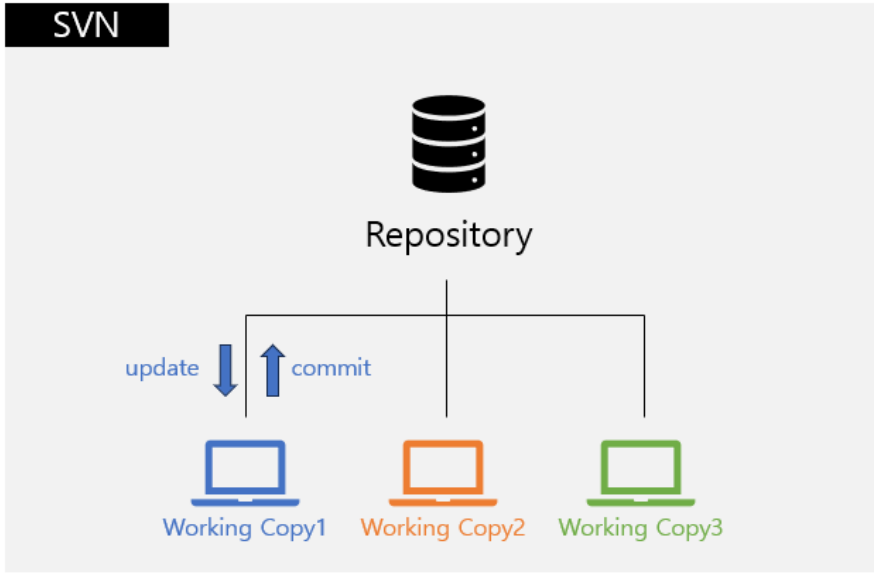


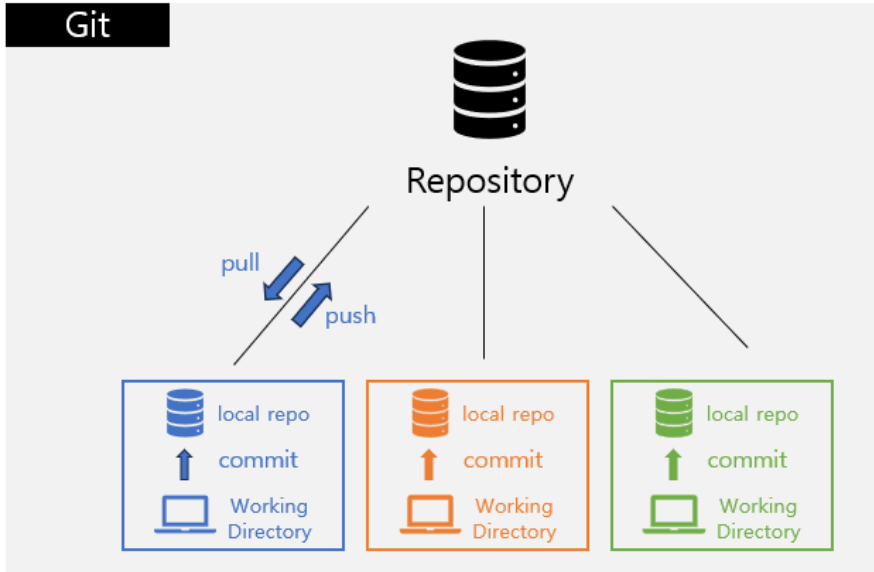


SVN vs Git – 차이점

- SVN은 중앙 서버에 존재하는 이력, 코드를 가져오는 개념
- Git은 로컬마다 이력, 코드를 복사본으로 개인마다 가지고 있는 개념

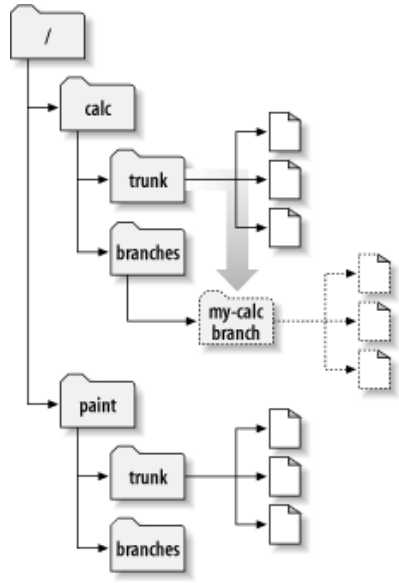
- SVN은 Commit을 하더라도 중앙 서버와 통신해야 하므로 네트워크가 필요
- Git은 push, pull 등 공유할 때만 필요



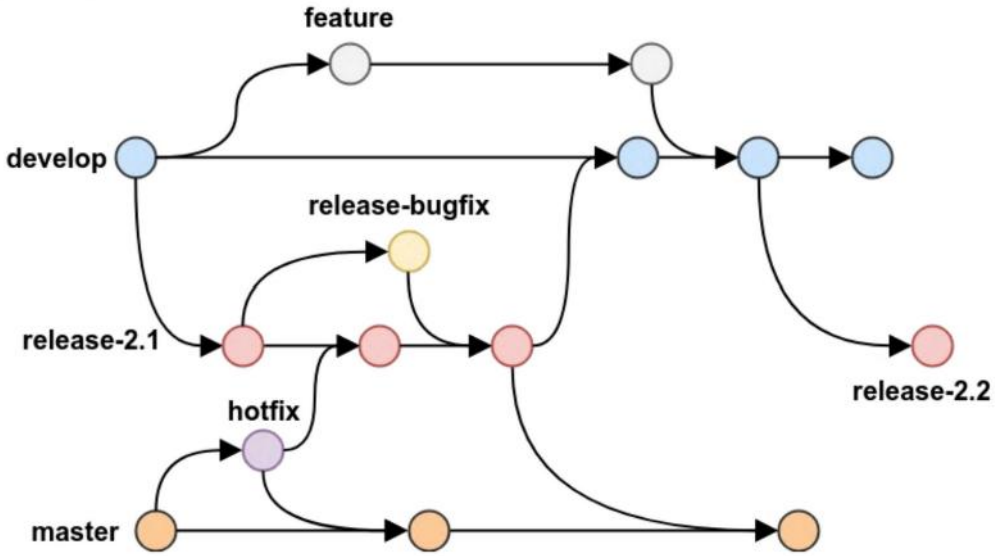


SVN vs Git – 차이점

- SVN의 브랜치는 서버의 디렉토리를 복사하는 방식이라 무겁지만
- Git의 브랜치는 특정 커밋을 가리키는 포인터 역할일 뿐이라 가벼움



- SVN은 리비전 번호를 사용하여 순차적으로 증가하는 방식이지만
- Git은 해시값을 사용하기 때문에 암호화하여 프로젝트를 저장



원격 저장소 (GitHub)

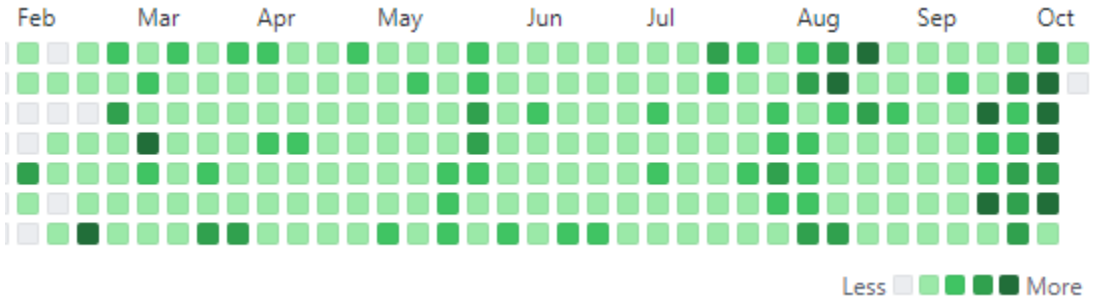
GitHub – 왜 사용해야 하는가?

- 팀의 생산성을 극대화하는 협업 플랫폼
  - 코드 리뷰
    - 메인 브랜치에 합치기 전에 동료들과 온라인으로 검토
  - 이슈 트래킹
    - 이슈가 생겼을 때 누가 수정하였는지 체계적으로 관리
  - 프로젝트 관리
    - 시각적으로 팀의 전체적인 작업 현황을 파악

GitHub – 제공하는 가치

- 오픈 소스
  - 전 세계 수많은 사용자가 오픈소스로 GitHub에 제공
  - 필요한 라이브러리를 쉽게 찾고 기여

- 개발자 포트폴리오
  - GitHub 활동(잔디 심기)과 코드는 이력서이자 포트폴리오가 될 수 있다.



Thank You