Git 사용법

Introduction to Git

공학설계입문/
C프로그래밍1/
객체지향프로그래밍

순서

본 자료의 내용은 Pro Git (Git – Book (git-scm.com))의 내용을 기반으로 합니다.

- Git 시작하기
- Git 기초 명령어
- 브랜치의 개념
- Git 사용법 요약 및 실습

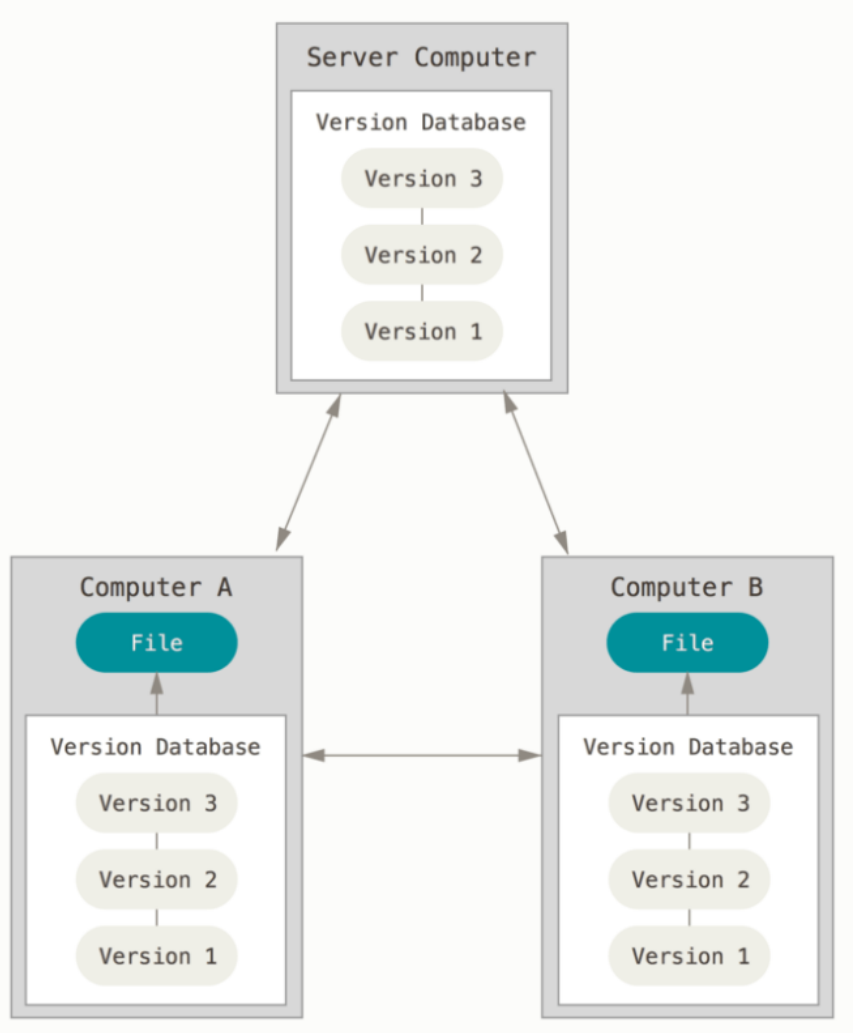
Git 시작하기

버전 관리 시스템 (VCS - Version Control System)

- 버전 관리 시스템이란?
  - 파일 변화를 시간에 따라 기록했다가, 나중에 특정 시점의 버전을 다시 꺼내올 수 있는 시스템

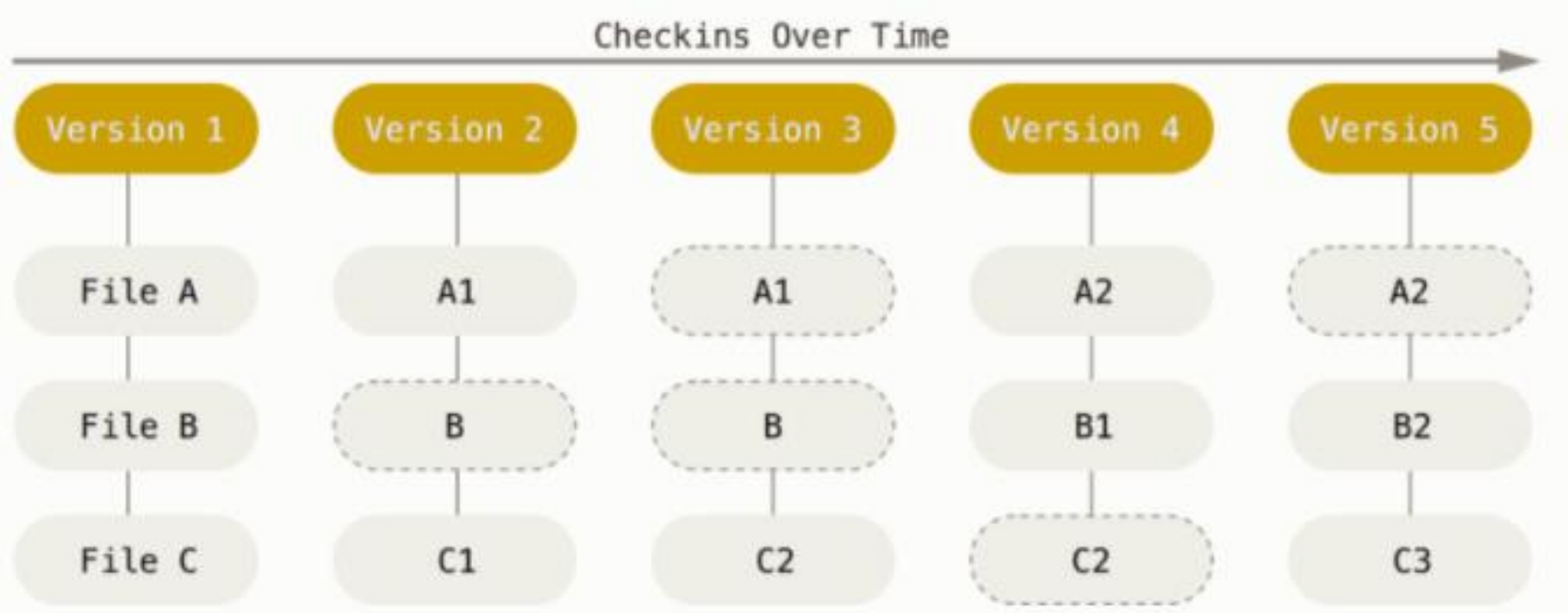
- 버전 관리 시스템이 필요한 이유
  - 잘못 동작하도록 수정한 코드를 작성한 후, 이전으로 되돌리기 위해
  - 동일 프로젝트에서 개발 협업을 하기 위해
  - 개발 이력을 남겨 나중에 확인하기 위해 (문제 추적 가능)

- Git은 대표적인 분산 버전 관리 시스템 (DVCS – Distributed VCS)
  - 서버의 저장소(repository)를 여러 클라이언트 컴퓨터에 모두 복제
  - 서버에 문제가 생겨도 클라이언트를 통해 복원 가능
  - Local repository (로컬 저장소)는 클라이언트에 존재하고, remote repository (리모트 저장소)는 보통 서버에 존재
  - 거의 모든 명령을 local에서 실행 가능



커밋(Commit)과 스냅샷(Snapshot)

- 스냅샷: 특정한 시점에서의 파일, 폴더, 또는 워크스페이스의 상태
- 새로운 버전을 기록하기 위한 커밋(commit)을 실행하면 스냅샷이 저장
  - git에서는 마지막 커밋의 스냅샷만 통째로 저장
  - 나머지 커밋에 대해서는 스냅샷과 스냅샷 간의 차이를 기록한 '델타'만 저장

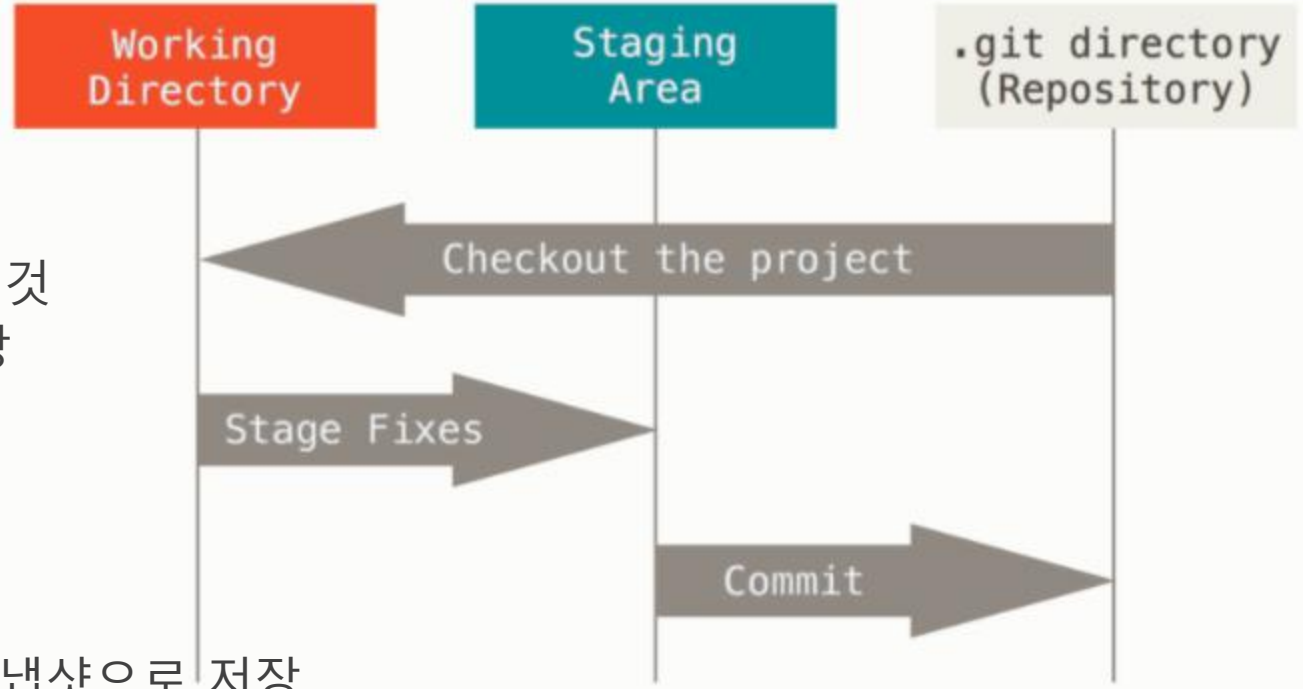


Git의 상태, 단계 및 작업 순서

- Git의 상태
  - Modified: 수정한 파일을 아직 로컬 데이터베이스에 커밋하지 않은 상태
  - Staged: 현재 수정한 파일을 곧 커밋할 것이라고 표시한 상태
  - Committed: 데이터가 로컬 데이터베이스에 안전하게 저장

- Git 프로젝트의 세 가지 단계
  - .git 디렉토리: Git이 프로젝트의 메타데이터와 객체 데이터베이스를 저장하는 곳
  - 작업 트리: 프로젝트의 특정 버전을 checkout*한 것
  - Staging area: 곧 커밋할 파일에 대한 정보를 저장

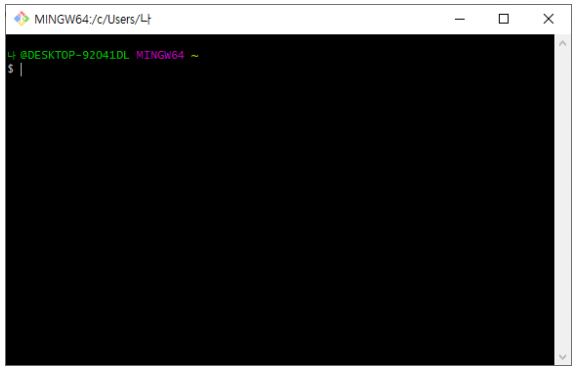
- Git의 작업 순서
  - 워킹 트리에서 파일(들) 수정
  - 수정된 파일(들)을 Stage해서 커밋할 스냅샷 생성
  - Staging area의 파일(들)을 커밋해서 영구적인 스냅샷으로 저장



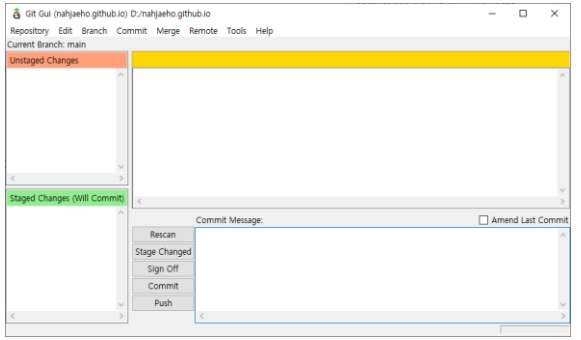
* checkout: 작업자의 작업트리를 저장소의 특정 시점과 일치 하도록 변경하는 작업

Git을 사용하는 방법

- CLI (command line interface)
  - 명령어를 직접 입력하는 방식
  - Git의 모든 기능 지원
  - 윈도우에서는 git bash 또는 git cmd를 실행하고, 리눅스에서는 직접 shell 상에 명령어 입력



- GUI (graphical user interface)
  - Git GUI 실행
  - 보기 쉽게 변경 사항 확인 가능
  - 버튼 클릭으로 명령 가능
  - 일부 기능만 구현



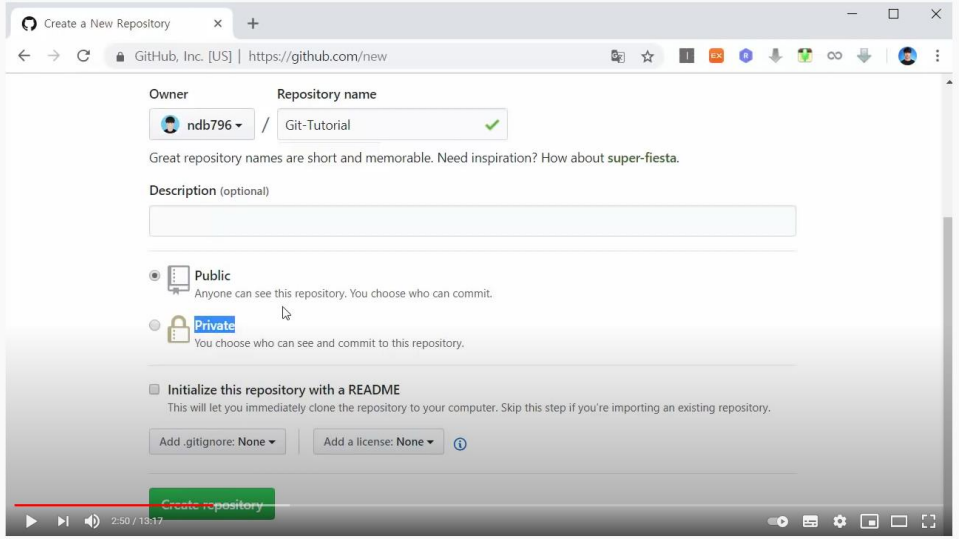
- 기본적으로 Git CLI를 통해 개념 및 사용법을 익히고, 나중에 여기에 익숙해졌을 때 Git GUI를 필요에 따라 사용하는 것을 추천합니다.

GitHub

- GitHub
  - 가장 큰 git repository host로, 수많은 오픈 프로젝트가 GitHub를 사용 중
  - 일반 사용자는 무료로 계정 생성 및 사용 가능 → 별도의 git server 관리가 필요 없음
  - 개인 홈페이지 용도로도 사용 가능: Nah, Jae-Ho (nahjaeho.github.io)

- GitHub 계정을 만들고 설정해 본 적이 없는 분은 아래 동영상을 클릭하기 바랍니다.
  - Git 설치 및 사용법 익히기 [Git으로 시작하는 협업 및 오픈소스 프로젝트 1강] – YouTube

- GitHub 계정 생성 후, 개인별 github 계정 주소 게시판에 본인의 계정 주소를 비밀 게시글로 등록해 주세요.
  - C프로그래밍1/객체지향프로그래밍 – 과제 제출 체크용
  - 공학설계입문 – 팀프로젝트 협업용



Git 기초

Git 저장소 만들기

- 아직 버전관리를 하지 않는 로컬 디렉토리 하나를 선택해서 Git 저장소를 적용하는 방법
  - 새로운 프로젝트를 시작할 때
  - 아래 명령어로 .git 하위 디렉토리가 만들어짐

    $ git init

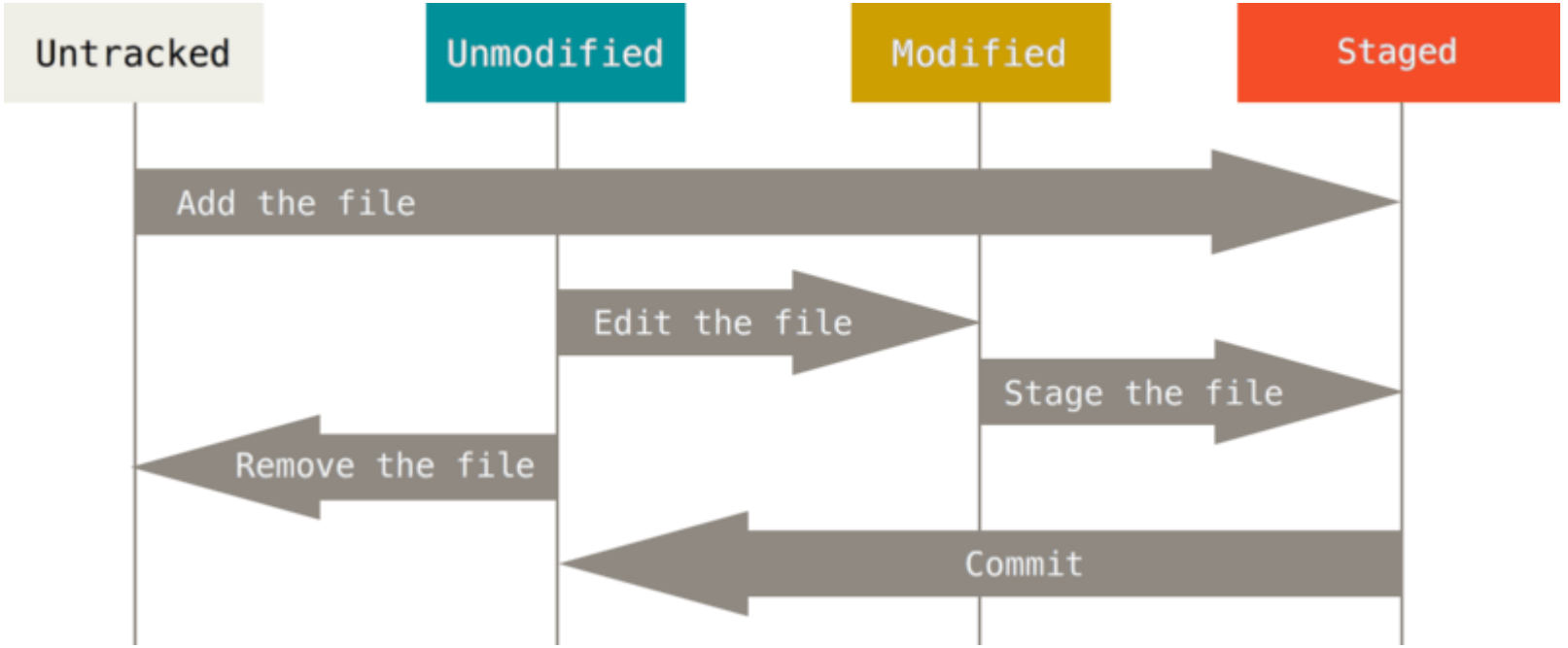
- 기존 저장소를 Clone하기
  - 다른 프로젝트에 참여하거나 남이 만든 Git 저장소를 복사하고 싶을 때
  - 다음 명령어로 clone되며, 자동으로 'origin' 저장소 생성

    $ git clone <주소(필수)> [새로운 디렉토리명 (옵션)]
  - 사용 예시 (libgit2 소스 코드를 mylibgit 디렉토리에 clone)

    $ git clone https://github.com/libgit2/libgit2 mylibgit

수정하고 저장소에 저장하기

- 파일의 라이프사이클
  - Untracked: git에서 관리되지 않는 파일들 (예: 임시 파일, 텍스트 파일, data 파일 등)
  - Unmodified: git에서 관리되나 수정되지 않은 파일들
  - Modifed: 수정된 파일들
  - Staged: 커밋하기 위해 준비된 파일들



추적 파일 추가 및 현재 상태 확인

- 추적 파일 추가: git add [옵션] <파일명 (와일드카드 사용 가능)>
  - 새롭게 파일을 추가할 경우 파일명 명시. 예) git add *.c
  - 업데이트된(수정 및 삭제된) 파일을 일괄적으로 추가하고 싶을 경우 –u 옵션 사용. 예) git add -u
- 현재 상태 확인 명령어: git status [옵션]
  - 사용 예시 (add 전)

```
$ echo 'My Project' > README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
```

(add 후)

```
$ git add README
```

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

  - Branch(추후 설명) 이름은 기본값으로 main (git v2.28 이후) 또는 master (v2.28 이전)으로 설정됨

변경 부분 확인

- 변경 부분 확인 명령어: git diff [옵션]
  - 옵션이 없으면 기본적으로 unstaged 파일에 대해 출력
  - --cached 또는 --staged 옵션: staged 파일의 변경 부분 출력
- 사용 예시

```
$ git add CONTRIBUTING.md
$ echo '# test line' >> CONTRIBUTING.md
```

```
$ git diff
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 643e24f..87f08c8 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -119,3 +119,4 @@ at the
 ## Starter Projects

 See our [projects list](https://github.com/libgit2/libgit2/blob/development/PROJECTS.md).
+# test line
```

변경 부분 확인

- 변경 부분 확인 명령어: git diff [옵션]
  - --stat 또는 --compact-summary 옵션: 변경 내역을 간단하게 출력 (추가/변경된 파일명 및 라인 수)
- 사용 예시

```
Techie Delight@ThinkPad MINGW64 ~/Desktop/api (master)
$ git diff HEAD~5..HEAD --stat
 CHANGELOG.md                                | 95 ++++++++++++++++++++++++++++++++++++
 RELEASE_NOTES_TEMPLATE.md                   | 57 +++++++++++++++++++++
 app/controllers/submissions_controller.rb | 54 ++++++++++++++++++-
 app/helpers/config.rb                       |  4 +-
 config/routes.rb                            |  3 +-
 judge0-api.conf.default                     |  6 +-
 6 files changed, 214 insertions(+), 5 deletions(-)
```

Find differences between two Git commits – Techie Delight

추적 파일의 삭제(또는 제외) 및 이름 변경

- 추적파일 삭제 명령어: git rm [옵션] <파일명>
  - 옵션에 --cached를 넣으면 실제 파일은 삭제되지 않고 추적에서만 제외 (untracked)
  - 위 옵션을 안 넣으면 기본적으로 실제 파일도 함께 삭제됨
  - 사용 예시)

    $ git rm PROJECTS.md

    $ git rm --cached PROJECTS.md

- 추적파일 이름 변경 명령어: git mv <원래 이름> <바뀔 이름>
  - 사용 예시)

    $ git mv README.md README

커밋 전 Undo

- git reset HEAD [옵션] <file>
  - [옵션]을 주지 않으면 <file> 상태를 unstaged로 변경
  - [옵션]에 --hard를 주면 <file>의 변경 내역을 취소하고 HEAD (마지막 스냅샷) 상의 내용으로 undo (변경 내역이 다 없어져 버리므로 주의)

- git checkout -- <file>
  - git reset --hard HEAD와 같은 결과

커밋 올리기 및 취소

- 커밋 올리기: git commit -m <메시지> [다른 옵션]
  - 옵션에 –m <메시지>를 안 쓰면 메시지 입력을 위한 에디터가 뜸
  - 옵션에 --amend를 적게 되면 커밋 변경 가능
  - 옵션에 –a를 쓰면 “git add –u” 명령어까지 한꺼번에 처리
  - 커밋을 올리면 지금 작업하는 로컬 브랜치의 마지막 스냅샷을 가리키는 HEAD 포인터가 방금 올린 커밋으로 이동됨
  - 사용 예시)
    $ git commit -m 'initial commit’
    $ git add forgotten_file
    $ git commit –amend
    $ git commit –amend –m ‘initial commit w/ forgotten_file’

- 커밋 취소: git reset [옵션] HEAD^
  - HEAD^는 HEAD 바로 직전의 커밋을 가리키는 포인터
  - [옵션]을 따로 안 주면 새로 올린 커밋만 삭제되고, 커밋을 올리기 직전의 상태로 돌아감 (로컬의 파일 변경 내역은 그대로 unstaged 상태로 존재)
  - [옵션]에 --hard를 넣으면 파일의 변경 내역까지 모두 삭제

리모트 저장소 (Remote Repository)

- 리모트 저장소
  - 인터넷이나 네트워크 어딘가에 있는 저장소
  - Fetch/Pull: 리모트 저장소→로컬 저장소
  - Push: 로컬 저장소→리모트 저장소

- 리모트 저장소 확인: git remote
  - 앞서 언급한 것처럼, clone하면 'origin'이라는 리모트 저장소가 자동으로 등록

```
$ git clone https://github.com/schacon/ticgit
Cloning into 'ticgit'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
$ cd ticgit
$ git remote
origin
```

리모트 저장소 추가하기

- 새 리모트 저장소 추가: git remote add <단축이름> <url>

```
$ git remote
origin
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb      https://github.com/paulboone/ticgit (fetch)
pb      https://github.com/paulboone/ticgit (push)
```

Pull/Fetch/Push하기

- 리모트 저장소의 데이터를 끌어오기: git fetch <remote>
  - 예) $ git fetch origin

- 새로 받은 브랜치의 내용을 merge할 때: git merge <remote> <branch>
  - 예) $ git merge origin main

- 위 두 명령어를 한 번에: git pull <remote> <branch>
  - Remote 저장소가 하나만 있고, 현재 branch에 합병할 경우 간단히 아래와 같이 입력
    $ git pull

- 리모트 저장소에 Push하기: git push <remote> <branch>
  - 예) $ git push origin main

더 많은 리모트 저장소 관련 명령어들

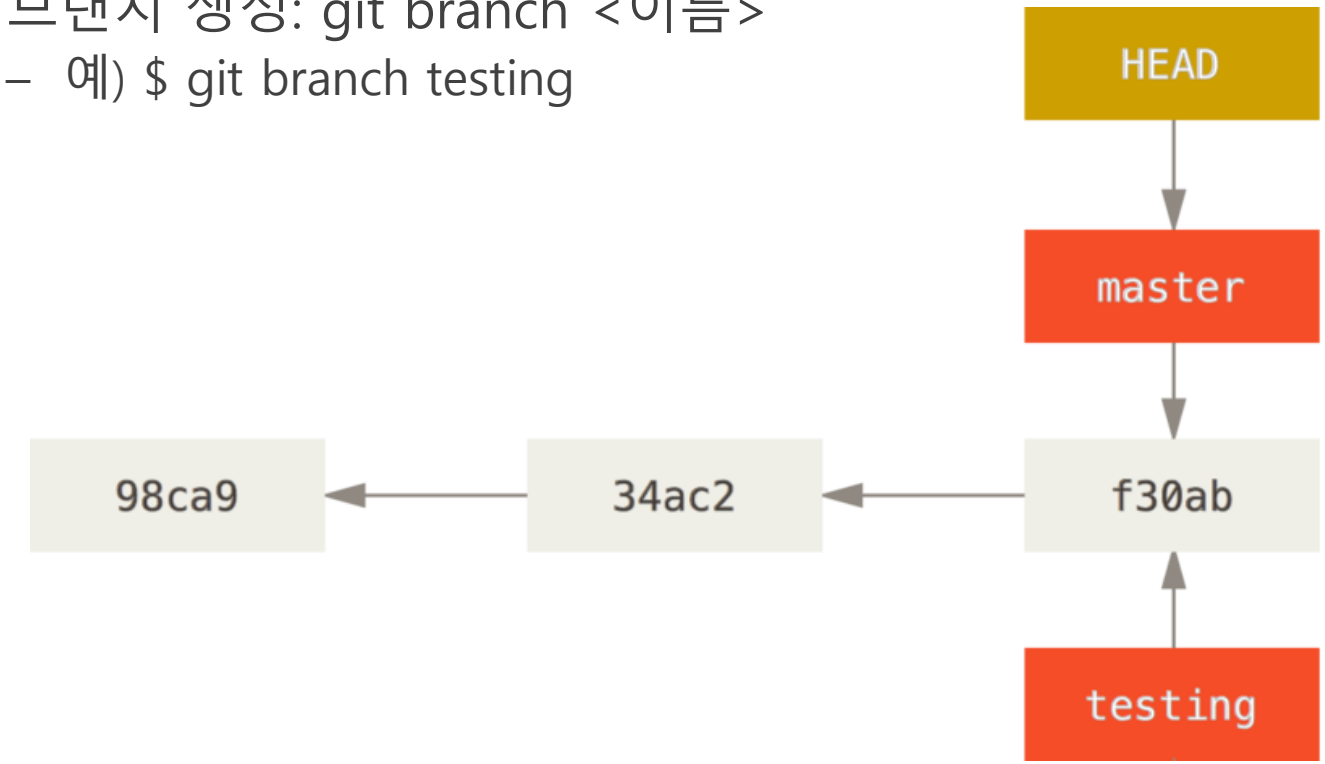
- 리모트 저장소 살펴보기: git remote show <remote>
- 리모트 저장소 이름 바꾸기: git remote rename <remote 원래 이름> <remote 바꿀 이름>
- 리모트 저장소 삭제: git remote remove <remote>

Branch의 개념

브랜치의 개념

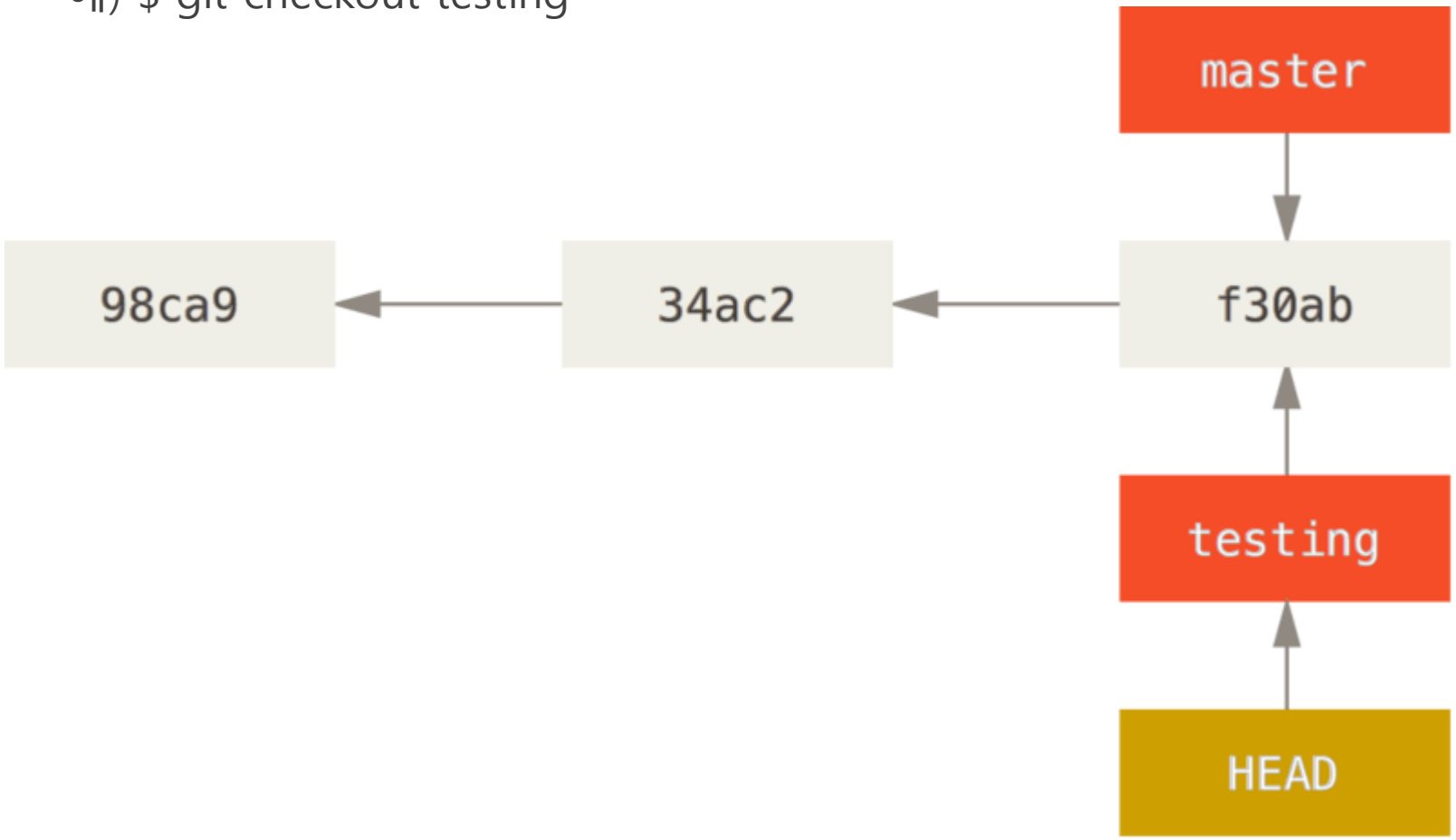
- 코드를 통째로 복사한 후, 원래 코드와 상관없이 독립적으로 개발하기 위한 버전
  - 예) 공식 릴리즈 버전, 기능 추가 1, 기능 추가 2, 기능 추가3, 베타테스트용, 알파테스트용 등등…

- 브랜치 생성: git branch <이름>
  - 예) $ git branch testing



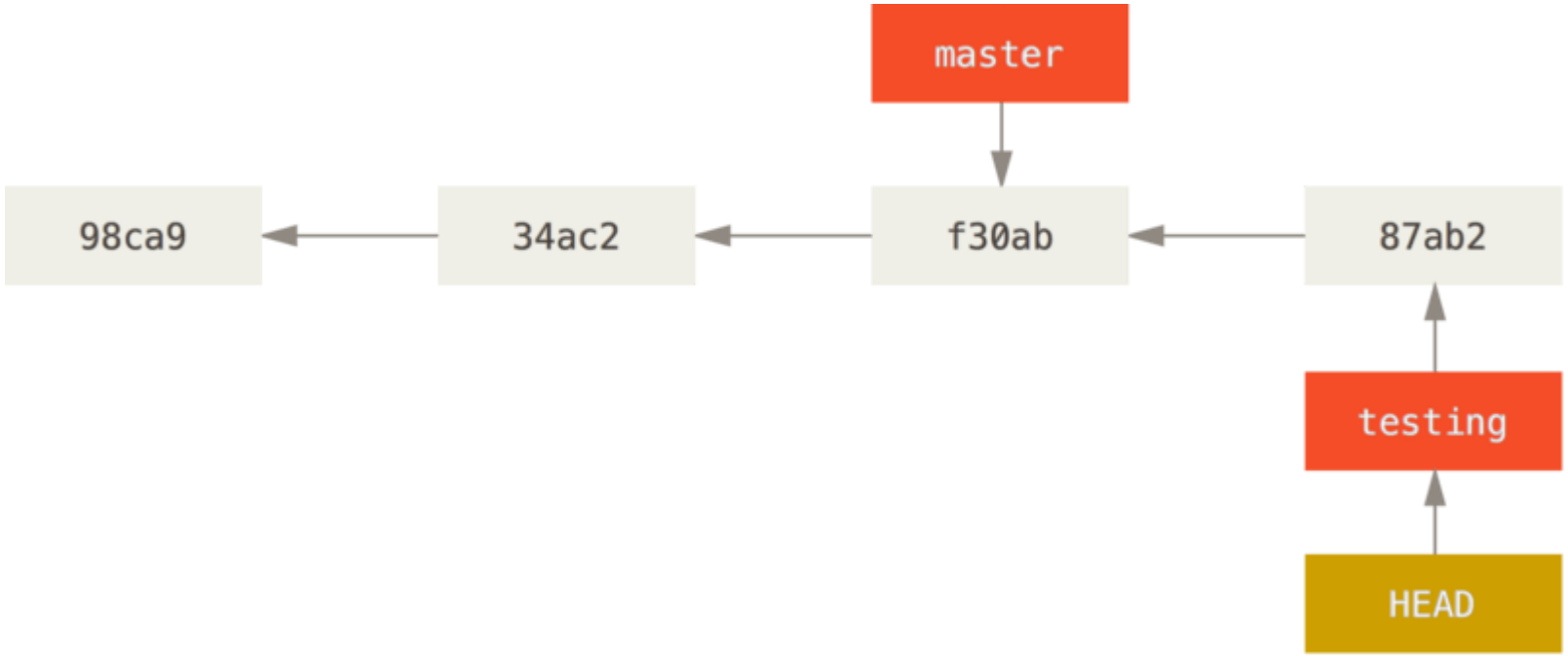
브랜치의 이동

- 브랜치 이동: git checkout <이름>
  - 예) $ git checkout testing



브랜치 이동 후 새 커밋을 올리면?

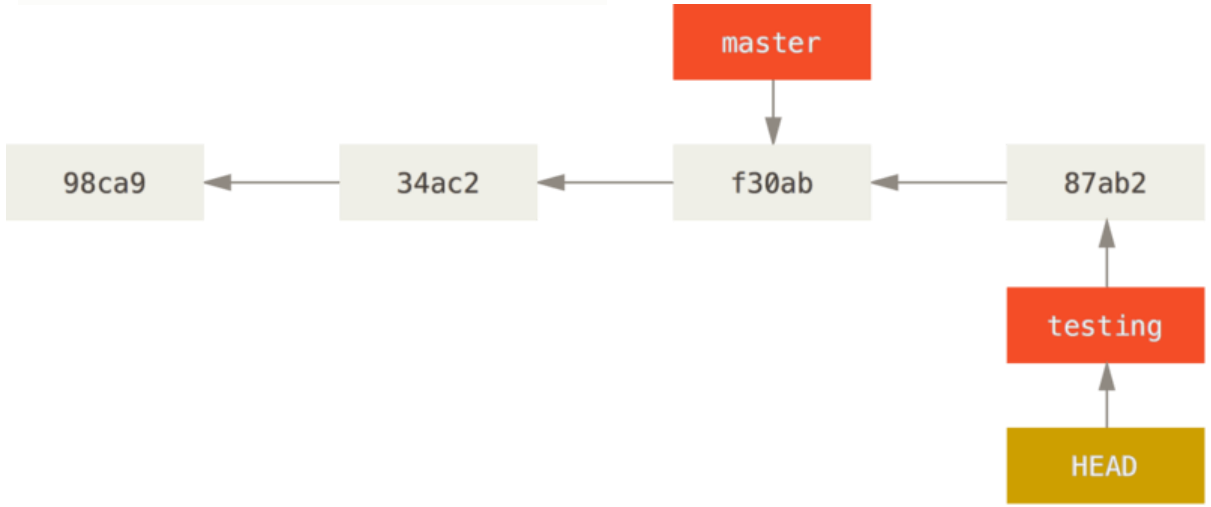
```
$ vim test.rb
$ git commit -a -m 'made a change'
```



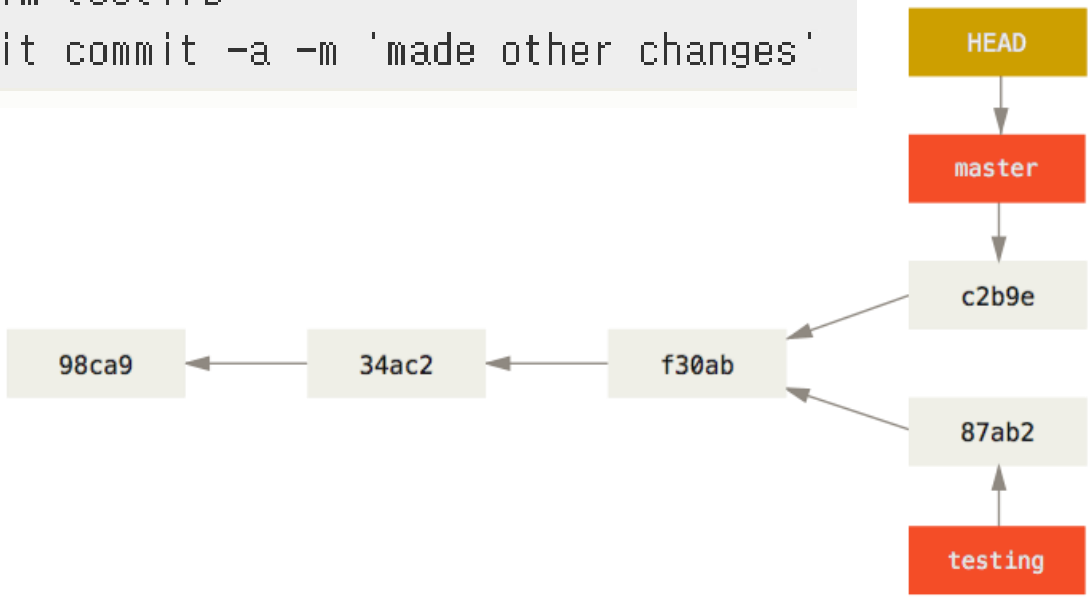
다른 브랜치로 다시 이동한 후 새 커밋을 올리면?

- 내용이 다른 두 test.rb가 서로 다른 브랜치에 서로 다른 커밋에서 따로 따로 존재하게 됨

```
$ git checkout master
```

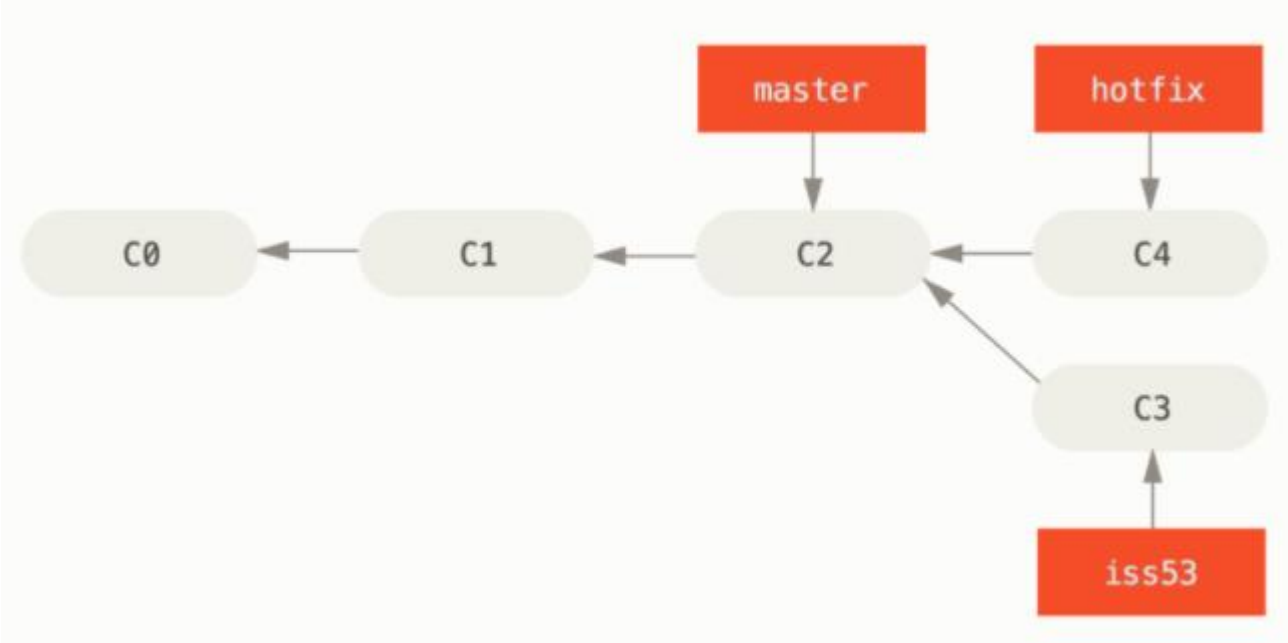



```
$ vim test.rb
$ git commit -a -m 'made other changes'
```



서로 다른 브랜치를 merge

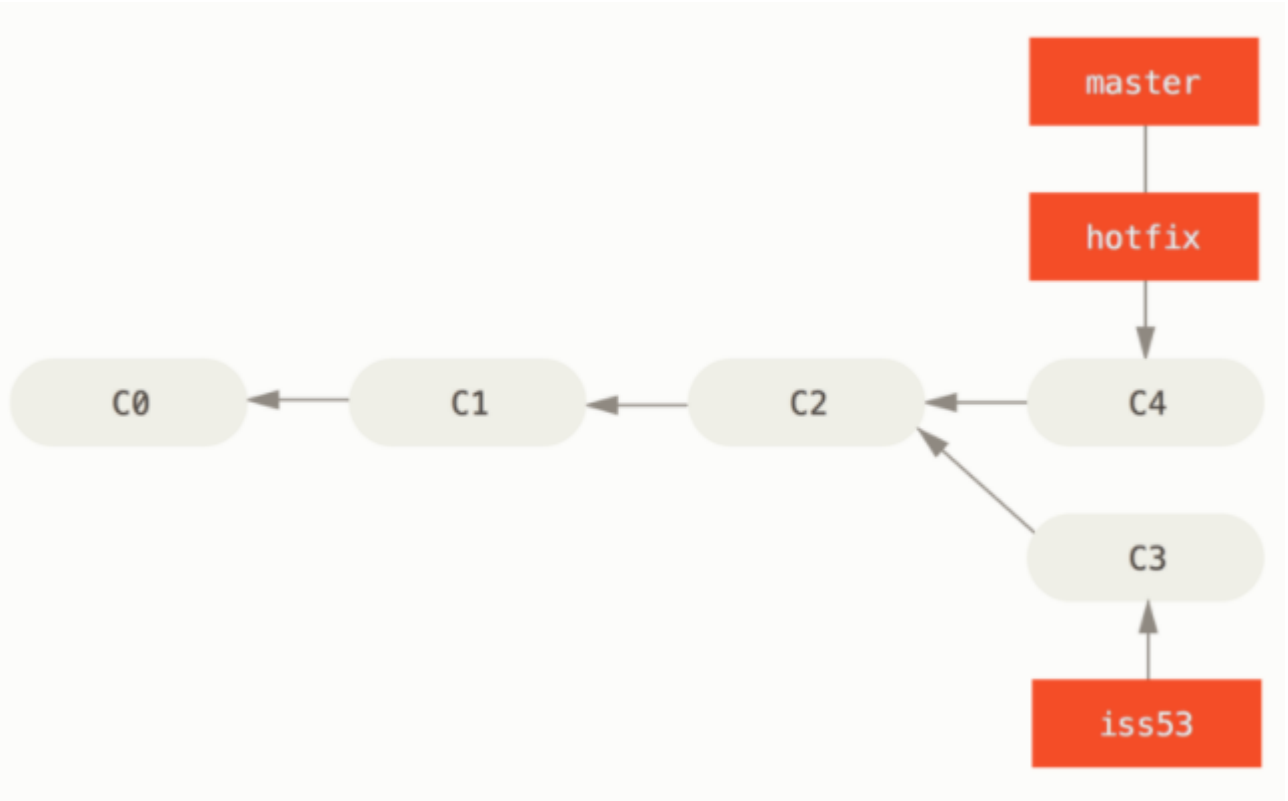
- 아래 세 브랜치가 있다고 가정
  - master
  - hotfix: master에 생긴 급한 문제의 해결 작업을 위해 만든 branch
  - iss53: master에 생긴 53번 이슈의 해결을 위해 만든 branch



서로 다른 브랜치를 merge

- master와 hotfix merge
  - Fast-forward: C4 커밋이 C2 커밋에 기반하므로 master의 포인터를 C4로 이동하는 것으로 완료

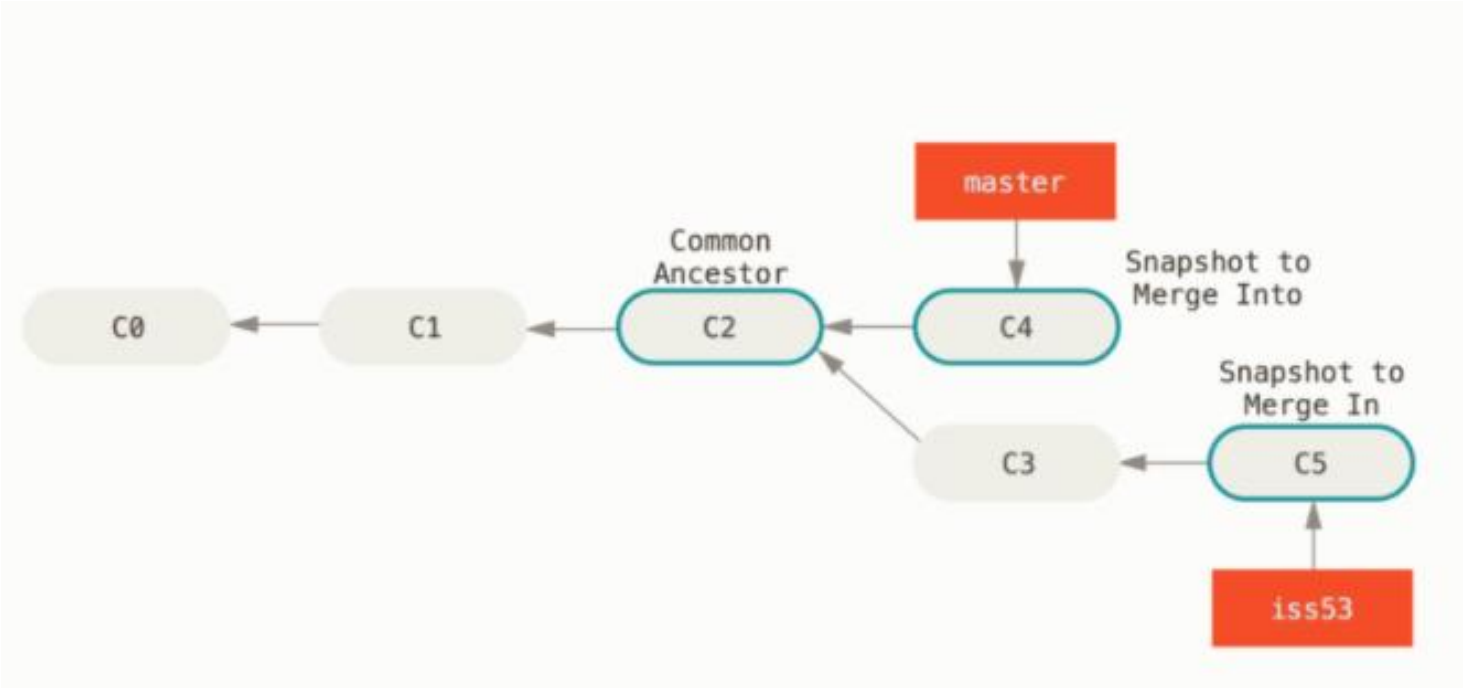
```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```



서로 다른 브랜치를 merge

- master와 iss53 merge
  - 3-way merge: 각 브랜치의 커밋 두 개와 공통 조상 하나를 사용하여 merge

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```



Merge중 충돌이 나면?

- 두 브랜치에서 같은 파일의 한 부분을 동시에 수정할 경우 발생
  - Merge하지 못하고 충돌(Conflict) 메시지 출력
  - git status 명령어를 통해 충돌난 파일 확인

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Merge중 충돌이 나면?

- 충돌난 파일의 내용은 아래와 같은 형식을 가짐
  - '=======' 위쪽의 내용은 master 브랜치의 HEAD의 내용, 아래쪽의 내용은 iss53 브랜치의 내용

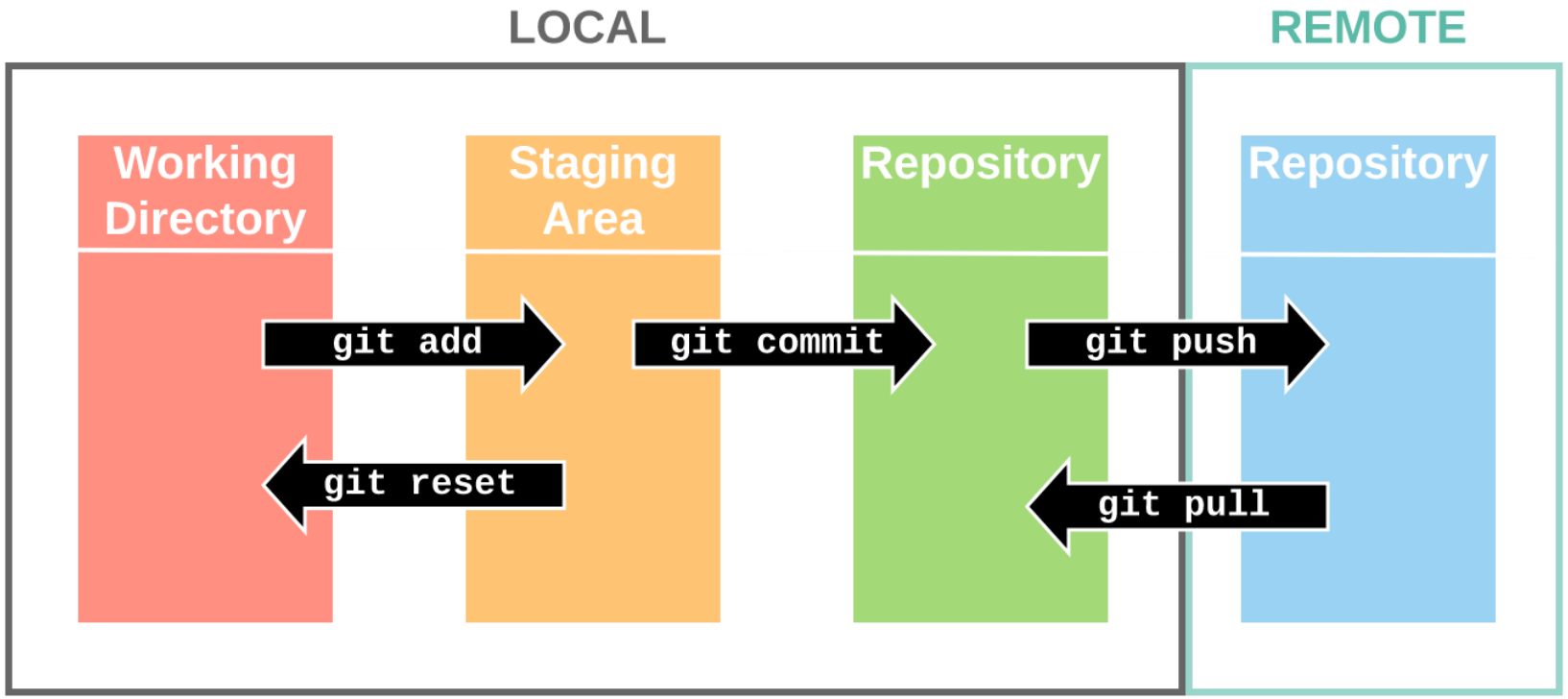
```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

- 수동으로 충돌 처리
  - 적절히 충돌 안 나게 수정하고, <<<<<<<, =======, >>>>>>>가 포함된 행은 삭제
  - 이후 git add 명령어로 다시 Git에 저장

Git 사용법 요약 및 실습

보통 코드 수정시 사용되는 git 명령어 루틴

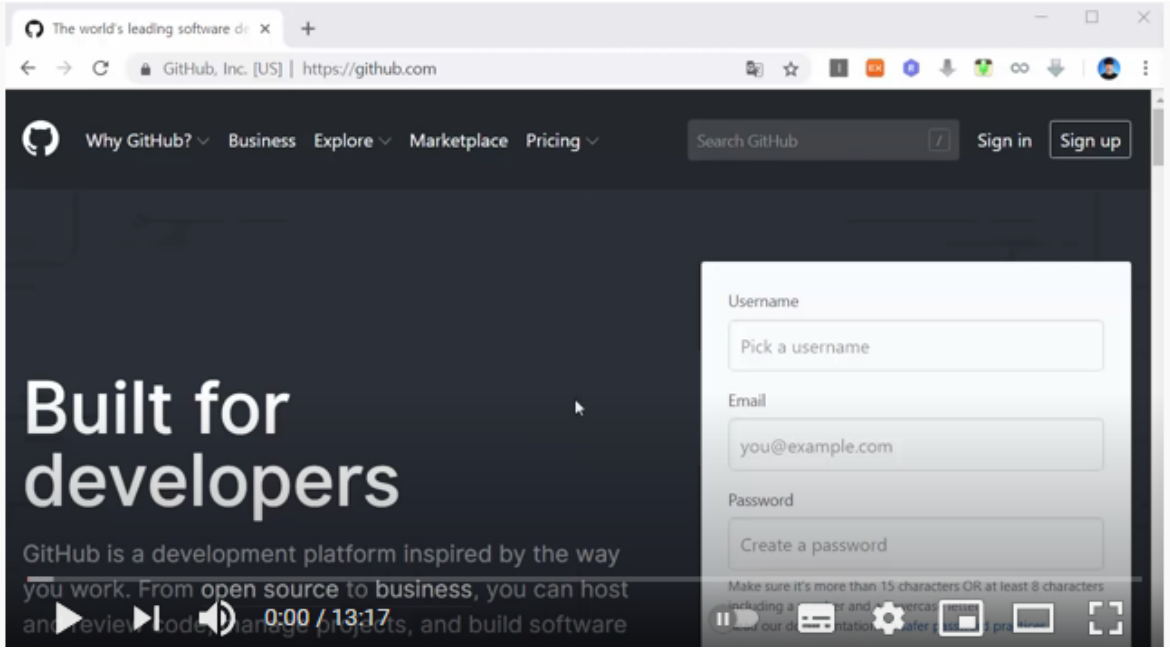
1. git pull
2. 코드 수정
3. git add <추가할 파일> 혹은
   git add –u
4. git commit –m '메시지'
5. git push

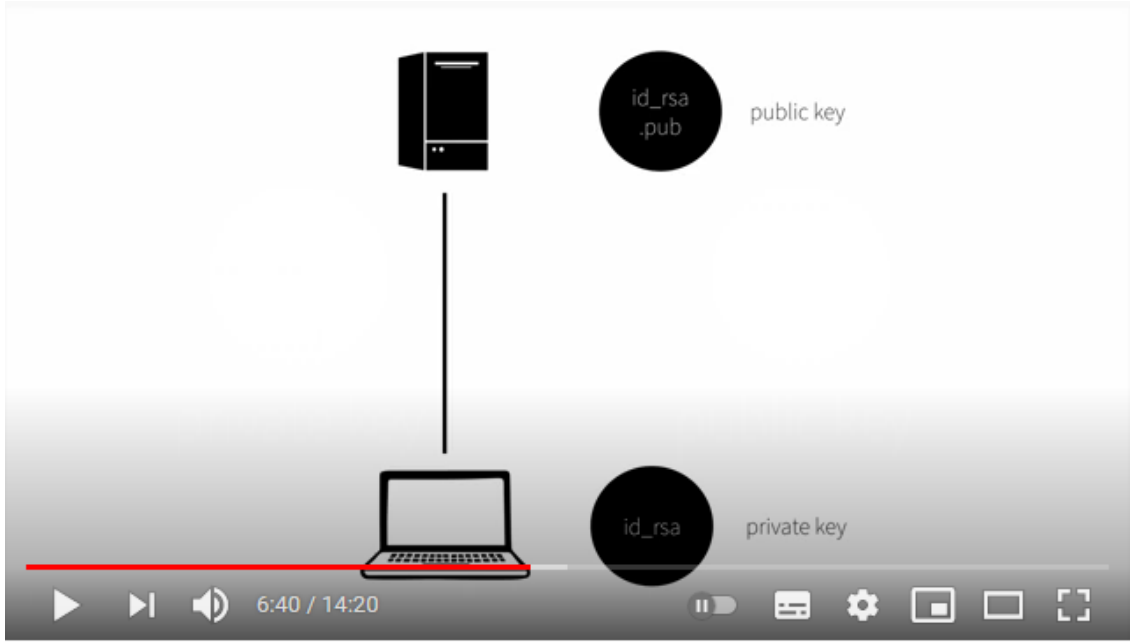


Git: Reference Sheet – NeSI Support

개인별 github 페이지 만들고 커밋 올리기

- Git 설치 및 사용법 익히기 [Git으로 시작하는 협업 및 오픈소스 프로젝트 1강] – YouTube
- 지옥에서 온 Git - ssh를 이용해서 로그인없이 원격저장소 사용하기 (Github) – YouTube





참여도 점수 1점 부여

- 앞에 소개한 동영상에 따라서 개인별로 github 페이지를 생성한 후, 커밋 업로드

- ‘개인별 github 페이지 주소 제출’ 게시판에 본인의 페이지 주소를 기재한 익명 글 제출
  - 이미 개인 github 페이지를 가지고 있을 경우, 한 줄짜리 글 하나만 올리면 됩니다.
  - **단, 커밋이 하나 이상 올라가 있어야 합니다.**
  - 올린 수강생에게 참여도 점수 1점 부여 (따로 댓글은 안 달아 드리며, 커밋만 올리면 무조건 1점)

다음주 예고 - git 추가 실습 (1시간 분량)

- 한종대 교수님의 '소프트웨어공학' 과목에서 다룬 실습 동영상을 별도로 업로드할 예정