



CHAPTER 05

서버

5장 서버

5.1 서버 저장소

5.2 깃허브 서버 준비

5.3 깃허브 연동 및 원격 등록

5.4 서버 전송

5.5 자동으로 내려받기

5.6 수동으로 내려받기

5.7 순서

5.8 정리

5.1 서버 저장소



1. 서버 저장소

➤ 서버 저장소

- 서버 저장소는 다른 말로 원격(remote) 저장소라고도 함
- 서버 저장소는 로컬 저장소의 코드를 복제한 복사본이라고 할 수 있음
- 서버를 이용하면 코드를 안전하게 보관할 수 있음
- 서버에 있는 소스 코드는 다른 사람들과 공유하고 협업하여 개발을 진행할 수도 있음



1. 서버 저장소

➤ 협업 저장소

- 최근 프로젝트 개발 규모가 점점 커질 뿐 만 아니라 고객이 요구하는 소프트웨어 품질도 점차 높아지고 있음
- 규모가 큰 프로젝트는 혼자서 모두 개발하기 어렵고, 시간과 노력이 많이 필요함
- 여러 명이 같이 협업하여 개발한다면 적은 시간으로 좀 더 좋은 품질의 소프트웨어를 만들 수 있음
- 깃은 여러 개발자와 협업하려고 탄생한 도구
- 과거와 달리 요즘 컴퓨터는 항상 인터넷에 접속되어 있고
아직도 365일 24시간 인터넷에 연결하여 작업할 수 없는 개발 환경도 많이 있음
- 깃은 이 두 가지 환경을 고려하여 분산형 모델을 선택함

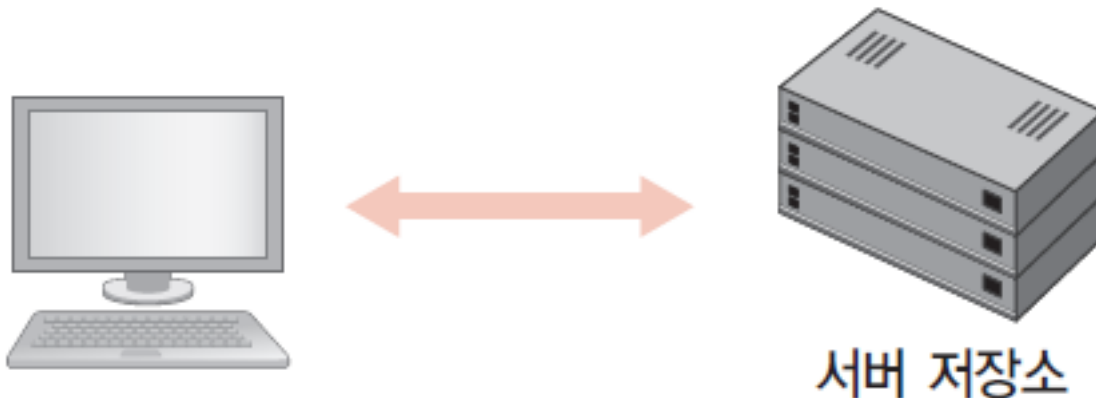


1. 서버 저장소

➤ 연속된 작업

- 원격 저장소가 있다면 언제 어디에서나 개발을 이어서 할 수 있음
- 사무실에서 개발 중인 코드를 서버에 저장함
- 집에 와서는 사무실에서 작업하고 서버에 올린 코드를 자신의 컴퓨터에 동기화할 수 있음
- 사무실, 집, 다른 여러 컴퓨터에 코드를 동기화하고 연속된 작업을 이어 갈 수 있음

▼ 그림 5-1 원격 저장소에 연결





1. 서버 저장소

➤ 연속된 작업

- 깃은 분산된 저장소 여러 개를 하나로 통합하고, 최신 코드를 배포할 수 있음
- 서버 저장소는 여러 컴퓨터에 동일한 깃 저장소를 복제하고, 작업한 결과물을 다시 서버로 통합



1. 서버 저장소

➤ 새 멤버

- 깃의 분산형 관리 체계는 다수의 사람과 협업하는 데 매력적임
- 기존 프로젝트에 새로운 멤버가 참여할 때, 지금까지 작업한 소스 코드의 마지막 버전을 공유해야 함
- 기존에는 코드를 공유하려고 이메일, 외부 저장 장치 등을 이용했지만, 이제는 깃의 원격 저장소 주소만 알려 주면 모두 해결됨
- 원격 저장소로 모든 구성원에게 코드의 최종 결과물을 동기화함

5.2 깃허브 서버 준비



2. 깃허브 서버 준비

➤ 깃허브 서버 준비

- 독립적인 깃 서버를 직접 운영하여 사용할 수 있음
- 365일 안정적인 서버를 운영하는 것은 쉽지 않음
- 직접 서버를 운영하지 않아도 전문적인 깃 호스팅으로 서버를 대체할 수 있음
- 호스팅을 받으면 직접 서버를 관리하지 않아도 쉽게 원격 저장소를 운영할 수 있음



2. 깃허브 서버 준비

➤ 깃허브

- 깃허브는 대표적인 깃호스팅 서비스임
- 깃허브의 모든 서비스는 무료로 사용할 수 있음
- 서비스를 사용하려면 먼저 회원 가입이 필요함
- 깃허브 공식 사이트는 <https://github.com>임



2. 깃허브 서버 준비

▼ 그림 5-2 깃허브 공식 사이트

Why GitHub? Enterprise Explore Marketplace Pricing

Search GitHub

Sign in Sign up

Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside 40 million developers.

Username

Email

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.



2. 깃허브 서버 준비

➤ 깃허브

- 회원 가입을 하려면 이메일 주소가 필요함
- 사용자 이름은 영문으로 작성하며, 저장소를 구별하는 주소 값으로 사용함
- 일반적으로 개별 깃허브 주소는 다음과 같이 표현함

`https://github.com/사용자이름`

- 회원 가입을 완료했다면 이메일로 본인 인증 과정을 거쳐야 함
- 회원 가입 후 입력했던 이메일 저장소에서 새 이메일을 확인함
- 이메일 목록에서 [GitHub] Please verify your email address.로 표시된 이메일을 클릭
- 이메일 안에 있는 Verify email address를 클릭
- 등록한 이메일 계정의 소유자인지 확인하는 절차임

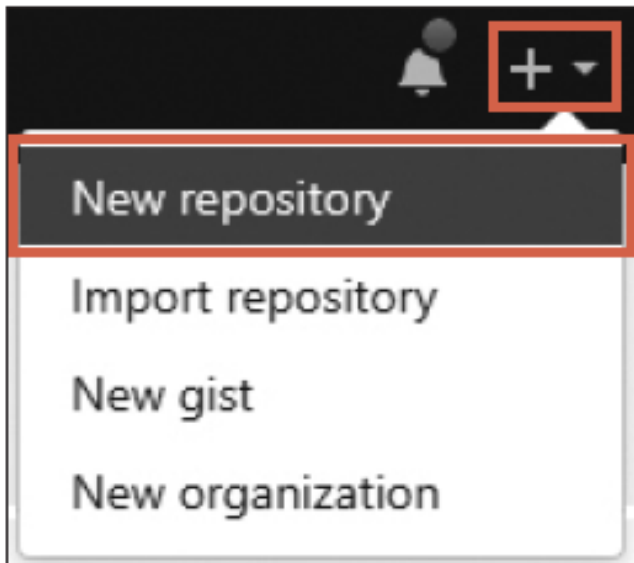


2. 깃허브 서버 준비

➤ 저장소 생성

- 공개된 저장소는 무제한으로 생성해서 사용할 수 있음
- 비공개용은 제약이 조금 있고, 일부는 유료 서비스임
- 로그인한 상태에서 새로운 저장소를 생성함
- 대시보드에서 New 버튼을 클릭하거나 + > New repository를 선택함

▼ 그림 5-3 저장소 생성 메뉴





2. 깃허브 서버 준비

➤ 저장소 생성


- 새 저장소 생성 화면으로 전환함
- 먼저 저장소를 생성할 소유자(owner)를 선택함
- Owner는 개인 계정 또는 생성한 조직을 선택하면 됨

▼ 그림 5-4 소유자 선택

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 jinygit ▾

Repository name *

Great repository names are short and memorable. Need inspiration? How about **didactic-eureka**.

Description (optional)



2. 깃허브 서버 준비

➤ 저장소 생성

- Repository name에는 저장소 이름을 입력함
- 이름은 대·소문자, 숫자, 하이픈, 밑줄을 조합하여 원하는 이름을 입력함
- 책에서는 gitstudy05로 입력함
- 입력한 이름으로 새 저장소가 생성됨
- 저장소 구성은 다음과 같이 URL을 이용하여 표기함

`https://github.com/소유자/저장소`

- 한 소유자 안에서 같은 저장소 이름은 중복하여 생성할 수 없음

5.3 깃허브 연동 및 원격 등록



3. 깃허브 연동 및 원격 등록

➤ 로컬 저장소

- 이 화면은 저장소를 생성할 때 README 체크 여부에 따라 달라짐
- README를 체크하지 않으면 다음과 같이 초기화 및 복제 방법을 안내함



3. 깃허브 연동 및 원격 등록

▼ 그림 5-5 깃허브에서 새 저장소 생성

The screenshot shows the GitHub interface for a new repository named 'gitstudy05' by user 'jinygit'. The page offers several options for setting up the repository:

- Quick setup — if you've done this kind of thing before**: Includes a button for 'Set up in Desktop' and a text input for 'HTTPS' with the URL 'https://github.com/jinygit/gitstudy05.git'. A note below states: 'Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.'
- ...or create a new repository on the command line**: Provides a code block with the following commands:

```
echo "# gitstudy05" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/click12click/gitstudy05.git
git push -u origin master
```
- ...or push an existing repository from the command line**: Provides a code block with the following commands:

```
git remote add origin https://github.com/click12click/gitstudy05.git
git push -u origin master
```
- ...or import code from another repository**: Includes a note: 'You can initialize this repository with code from a Subversion, Mercurial, or TFS project.' and an 'Import code' button.



3. 깃허브 연동 및 원격 등록

➤ 로컬 저장소

- 원격 저장소에 연결하려면 먼저 로컬 저장소가 있어야 함
- 로컬 저장소를 원격 저장소에 연결하는 방법은 크게 두 가지임
 - ① 새로운 로컬 저장소를 생성하고 원격 저장소를 연결하는 방법
 - ② 기존 저장소를 연결하는 방법



3. 깃허브 연동 및 원격 등록

➤ 로컬 저장소

- 책에서는 ① 방법으로 원격 저장소에 연결함
- 먼저 새 로컬 저장소를 생성하고 초기화함

```
$ mkdir gitstudy05 ----- 새 폴더 만들기
```

```
$ cd gitstudy05
```

```
$ git init ----- 저장소를 깃으로 초기화
```

```
Initialized empty Git repository in E:/gitstudy05/.git/
```



3. 깃허브 연동 및 원격 등록

➤ 로컬 저장소

- 저장소의 소개 페이지 파일을 하나 생성함
 - 소개 페이지 파일의 이름은 README.md이고, 텍스트 기반 마크업 언어인 Markdown 사용
- echo 명령어로 문자열을 파일로 리다이렉션하여 소개 페이지 파일을 만들
 - 직접 편집기를 이용하여 만들어도 됨

```
$ echo "# gitstudy05" >> README.md ----- 파일 생성
```

- 만든 README.md 파일을 추적 등록하고 커밋함

```
$ git add README.md ----- 스테이지에 등록
```

```
$ git commit -m "first commit" ----- 커밋
```



3. 깃허브 연동 및 원격 등록

➤ 프로토콜

- 깃은 서버와 통신할 수 있는 다양한 프로토콜을 지원함
 - 서버와 통신하려면 통신 프로토콜을 사용해야 함
- 깃은 기본적으로 Local, HTTP, SSH, Git 등 네 종류의 전송 방식을 지원함



3. 깃허브 연동 및 원격 등록

➤ 프로토콜

① Local(로컬)

- 로컬 컴퓨터에 원격 저장소를 생성하는 것을 의미함
- 이 방식은 자신의 컴퓨터를 NFS(Network File System) 등 서버로 이용할 때 편리함
- 로컬 저장소를 서버로 이용할 때는 폴더 경로만 입력하면 됨

```
$ git remote add 원격저장소별칭 폴더경로
```

- 로컬은 간단하게 원격 서버를 구축할 수 있을 뿐만 아니라 빠른 동작이 가능함
- 모든 자료가 자신의 컴퓨터에 집중되는 위험도 있음



3. 깃허브 연동 및 원격 등록

➤ 프로토콜

② HTTP

- 깃은 HTTP 방식의 프로토콜을 지원함
- HTTP는 SSH 프로토콜 보다 많이 사용하는 프로토콜 중 하나임
- 깃허브, 비트버킷 같은 호스팅 서비스도 기본적으로 HTTP 프로토콜을 지원함
 - 단, 서버에 접속하려면 로그인 절차를 거쳐야 함
- HTTP는 기존 아이디와 비밀번호만으로 접속자를 인증하여 처리할 수 있음
- HTTP는 익명으로도 처리할 수 있으며, 계정을 이용하여 처리할 수도 있음



3. 깃허브 연동 및 원격 등록

➤ 프로토콜

③ SSH

- SSH는 깃에서 권장하는 프로토콜로, 높은 수준의 보안 통신으로 처리하기 때문에 깃 서버를 좀 더 안전하게 운영할 수 있음
- SSH 프로토콜을 사용하려면 주소 앞에 'ssh://계정@주소' 처럼 프로토콜 타입을 지정해야 함
 - 계정을 생략하여 현재 로그인된 사용자로 대체할 수도 있음
- SSH 접속을 할 때는 인증서를 만들어 사용함
 - 인증서를 만들어 접속하면 별도의 회원 로그인 절차를 거치지 않아도 됨
 - 인증서는 공개키와 개인키로 구분하는데 공개키는 서버에 등록하며, 개인키는 로컬에 저장함
- SSH는 HTTP와 달리 익명으로 접속할 수 없음
 - 이러한 점이 기업에서 깃 서버를 운영할 때 적합한 프로토콜이라고 할 수 있음



3. 깃허브 연동 및 원격 등록

➤ 프로토콜

④ Git

- Git 프로토콜: 깃의 데몬 서비스를 위한 전용 프로토콜 방식을 의미함
- SSH와 유사하지만 인증 시스템이 없어 보안에 취약할 수 있음
- 실제로 이 프로토콜은 잘 사용하지 않는 편임



3. 깃허브 연동 및 원격 등록

➤ 원격 저장소의 리모트 목록 관리

- 깃은 원격 저장소(서버)를 관리하는 데 `remote` 명령어를 사용함
- `remote` 명령어를 사용하면 현재 연결된 원격 저장소 목록을 확인할 수 있음
- 동시에 등록과 취소 등 작업을 할 수 있음
- `remote` 명령어에는 다양한 옵션이 있으며, `-help` 옵션으로 확인할 수 있음
- 명령어 하나로 다수의 리모트 작업을 하기 때문에 몇 가지 옵션은 반드시 알고 넘어가야 함



3. 깃허브 연동 및 원격 등록

➤ 원격 저장소의 리모트 목록 관리

- remote 명령어를 독립적으로 사용하면 연결된 원격 저장소의 이름(별칭)을 출력함
- 간단하게 원격 저장소 목록만 확인할 때 편리함

```
$ git remote
```

- -v 옵션을 같이 사용하면 원격 저장소의 별칭 이름과 URL을 확인할 수 있음

```
$ git remote -v
```

- 깃은 복수의 원격 저장소를 연결하여 사용할 수 있음
- 리모트 저장소가 여러 개 있을 때는 목록을 모두 출력함
- 저장소의 권한 정보까지는 알 수 없음



3. 깃허브 연동 및 원격 등록

➤ 원격 자원 접근을 위한 URL과 별칭

- 로컬 저장소에 원격 저장소(서버)를 등록하려면 원격 저장소 URL이 필요함
 - 깃허브 같은 저장소를 이용해 보면 원격 자원 접근할 때 URL 형식을 사용하는 것을 확인할 수 있음
 - URL : Uniform Resource Locator
 - URL = 프로토콜 + 원격 서버의 도메인 이름 + 해당 서버에 존재하는 객체의 절대 경로
 - 원격 서버에 존재하는 객체의 절대 경로는 서버 내 해당 폴더 경로를 사용
- 원격 저장소의 긴 URL을 대신하는 별칭 사용
 - 매번 원격 서버에서 작업할 때마다 긴 문자열을 입력하는 것은 피곤함
 - 긴 원격 서버의 도메인 주 URL 문자열을 별칭으로 간략하게 만들어 사용할 수 있음
 - 자신의 목적에 따라 다른 이름을 사용해도 괜찮음
 - **origin**: origin은 대표적으로 사용하는 별칭임
 - 기본적으로 원격 서버와 연결할 때 별칭으로 origin을 사용하는 것을 많이 볼 수 있음



3. 깃허브 연동 및 원격 등록

➤ 원격 저장소에 연결

- 원격 저장소와 연결하려면 add 옵션을 사용함

```
$ git remote add 원격저장소별칭 원격저장소URL
```



3. 깃허브 연동 및 원격 등록

➤ 원격 저장소에 연결

- 원격 저장소를 추가할 때는 인자 값으로 원격 저장소 별칭과 원격 저장소 URL을 같이 입력
- 별칭은 원격 저장소 URL이 길기 때문에 쉽게 작업하려고 사용하는 별명임

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git remote add origin https://github.com/jinygit/gitstudy05.git ----- 자신의 서버 주소를 입력
```

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git remote -v ----- 원격 저장소 목록 확인
```

```
origin https://github.com/jinygit/gitstudy05.git (fetch)
```

```
origin https://github.com/jinygit/gitstudy05.git (push)
```




3. 깃허브 연동 및 원격 등록

➤ 원격 저장소에 연결

- 원격 저장소가 연결되면 fetch와 push 시 사용할 원격 저장소 URL이 출력됨
- push(푸시)는 서버로 전송하는 동작을 의미하고,
fetch(페치)는 반대로 서버에서 가지고 오는 동작을 의미함
- 별칭은 중복하여 선택할 수 없음



3. 깃허브 연동 및 원격 등록

➤ 소스트리에서 원격 브랜치

- 원격 저장소를 등록하면 기존 master 브랜치와 달리 또 하나의 브랜치가 표시됨
- 그림 5-6은 서버에 전송했던 실습 화면을 캡처한 것임

▼ 그림 5-6 원격 브랜치





3. 깃허브 연동 및 원격 등록

➤ 소스트리에서 원격 브랜치

- master는 현재 로컬 저장소를 의미함
- local/master 같은 별칭/브랜치는 원격 저장소의 브랜치를 의미함
- 로컬 저장소와 서버 저장소를 구분하여 표시함
- 서로 동기화한 시점을 판별할 수 있음



3. 깃허브 연동 및 원격 등록

➤ 별칭 이름 변경과 정보

- 별칭은 긴 문자열의 서버 주소를 대체함
- 등록된 서버의 별칭 이름은 다시 변경할 수 있음
- rename 옵션을 같이 사용함

```
$ git remote rename 변경전 변경후
```

- remote 명령어는 간략한 원격 저장소 정보만 출력함
- 원격 저장소의 좀 더 상세한 정보를 확인하고 싶다면 show 옵션을 같이 사용함

```
$ git remote show 원격저장소별칭
```



3. 깃허브 연동 및 원격 등록

➤ 별칭 이름 변경과 정보

- 예를 들어 다음과 같이 `remote show` 명령어를 실행하면 상세한 정보를 출력함

예

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/jinygit/gitstudy05.git
  Push URL: https://github.com/jinygit/gitstudy05.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local ref configured for 'git push':
    master pushes to master (up to date)
```



3. 깃허브 연동 및 원격 등록

➤ 원격 서버 삭제

- 로컬 저장소는 복수의 원격 저장소와 연결할 수 있음
- 깃을 사용하다 보면 풀 리퀘스트(pull request), 테스트 등 목적으로 임시 등록된 원격 저장소들도 있음
- 등록된 원격 저장소는 rm 옵션으로 삭제할 수 있음

```
$ git remote rm 원격저장소별칭
```



3. 깃허브 연동 및 원격 등록

➤ 원격 서버 삭제

- 등록한 origin 저장소를 삭제하고, 목록을 다시 확인해보자

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git remote rm origin ----- 원격 저장소 삭제
```

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git remote -v ----- 원격 저장소 목록 확인
```

- 등록, 변경, 삭제하여 다수의 원격 저장소를 관리할 수 있음



3. 깃허브 연동 및 원격 등록

➤ 원격 서버 삭제

- 다음 실습을 대비하여 삭제된 원격 저장소를 다시 등록해 놓음

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git remote add origin https://github.com/jinygit/gitstudy05.git ----- 재등록
```


5.4 서버 전송



4. 서버 전송

➤ push: 서버에 전송

- 푸시(push):
로컬 저장소의 커밋된 파일들을 원격 저장소로 업로드하는 동작
- 원격 저장소와 연결된 서버 목록을 확인해보자

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git remote -v
```

```
origin https://github.com/jinygit/gitstudy05.git (fetch)
```

```
origin https://github.com/jinygit/gitstudy05.git (push)
```



4. 서버 전송

➤ push: 서버에 전송

- origin `https://github.com/jinygit/gitstudy05.git` (push)처럼 업로드가 가능한 원격 저장소가 등록된 것을 확인할 수 있음
- 원격 저장소로 로컬 깃 저장소의 내용을 전송할 때는 `push` 명령어를 사용함

```
$ git push 원격저장소별칭 브랜치이름
```

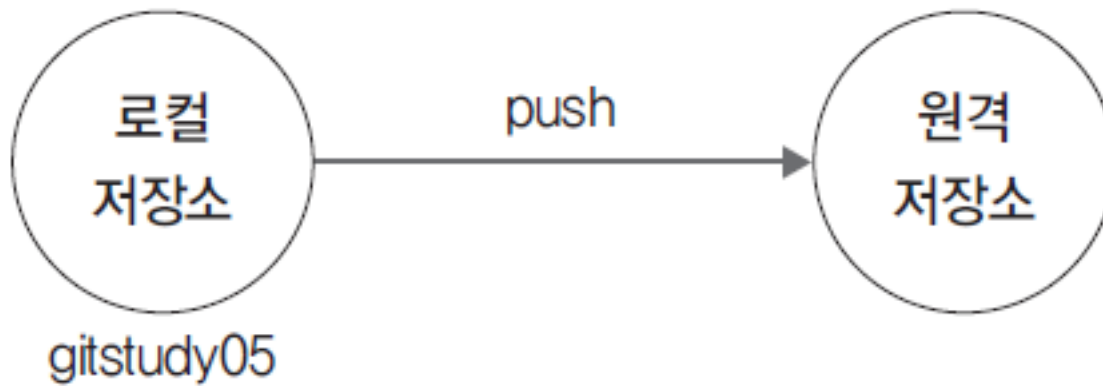


4. 서버 전송

➤ push: 서버에 전송

- 별칭 이름을 가지는 서버의 master 브랜치에 현재 브랜치를 업로드함

▼ 그림 5-7 푸시 동작





4. 서버 전송

➤ push: 서버에 전송

- 그럼 실제로 push 명령어를 사용하여 현재 master 브랜치를 origin 서버로 전송해보자

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
$ git push origin master ----- 원격 서버로 전송
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 222 bytes | 222.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/jinygit/gitstudy05.git
* [new branch]      master -> master
```



4. 서버 전송

➤ push: 서버에 전송

- 준비한 깃허브 저장소를 확인함
- 처음 푸시하면 서버에 새로운 master 브랜치를 생성함
- 로컬의 master 브랜치 안에 있는 소스 코드를 서버의 master 브랜치로 업로드함

4. 서버 전송



▼ 그림 5-8 원격 저장소에 푸시

The screenshot displays the GitHub interface for a repository named 'jinygit / gitstudy05'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the repository name, there are buttons for 'Unwatch', 'Star', and 'Fork'. The main navigation bar includes 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The repository details section shows '서버생성' (Server Creation) with an 'Edit' button. Below this, there are statistics: '1 commit', '1 branch', '0 packages', '0 releases', and '1 contributor'. A 'Branch: master' dropdown and a 'New pull request' button are also present. The 'Create new file', 'Upload files', 'Find file', and 'Clone or download' buttons are visible. The commit history shows 'infohojin first commit' with the latest commit at '4864581' made '2 minutes ago'. The README.md file is listed with the content 'gitstudy05'.



4. 서버 전송

➤ push: 서버에 전송

- 자신의 로컬 저장소를 백업하는 용도로 원격 저장소를 사용할 수 있음
- 꼭 다른 사람들과 협업하려고 깃의 원격 저장소를 사용하는 것은 아님
- 혼자 개발할 때도 백업을 위해 원격 저장소를 같이 사용하면 좋음

5.5 자동으로 내려받기



5. 자동으로 내려받기

➤ clone: 복제

- 복제는 기존 저장소를 이용하여 새로운 저장소를 생성하는 방법 중 하나임
- 일반적인 복제와 원격 저장소를 복제하는 방법은 조금 차이가 있음
- 복제할 때는 clone 명령어를 사용함
- clone 명령어는 초기화 init 명령어 외에 원격 서버 접속에 필요한 추가 설정을 자동으로 수행함
- 서버의 연결 설정을 마친 후 서버 안에 있는 모든 커밋된 코드 이력들을 한 번에 내려받음



5. 자동으로 내려받기

➤ clone: 복제

- 앞에서 업로드한 원격 저장소를 다른 이름의 로컬 저장소로 복제해보자

```
$ cd 메인폴더
```

```
$ mkdir gitstudy05_clone ----- 복제할 새 폴더 만들기
```

```
$ cd gitstudy05_clone/
```

```
infoh@hojin MINGW64 /e/gitstudy05_clone
```

```
$ git clone https://github.com/jinygit/gitstudy05.git . ----- 저장소 복제
```

```
Cloning into '.'...
```

```
remote: Enumerating objects: 3, done.
```

```
remote: Counting objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
```

```
Unpacking objects: 100% (3/3), done.
```



5. 자동으로 내려받기

➤ clone: 복제

- 복사한 후 `ls -all` 명령어로 목록을 확인해보자

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ ls -all ----- 목록 확인
```

```
total 13
```

```
drwxr-xr-x 1 infoh 197609  0 12월 12 17:23 .
```

```
drwxr-xr-x 1 infoh 197609  0 12월 12 17:23 ..
```

```
drwxr-xr-x 1 infoh 197609  0 12월 12 17:23 .git ----- 숨김 저장소
```

```
-rw-r--r-- 1 infoh 197609 14 12월 12 17:23 README.md
```



5. 자동으로 내려받기

➤ clone: 복제

- 원격 서버를 정상적으로 복제함
- 복제 후 remote 명령어를 사용하여 원격 저장소 정보를 확인함

```
infoh@gitstudy MINGW64 /e/gitstudy05_clone (master)
```

```
$ git remote -v ----- 원격 저장소 목록
```

```
origin https://github.com/jinygit/gitstudy05.git (fetch)
```

```
origin https://github.com/jinygit/gitstudy05.git (push)
```

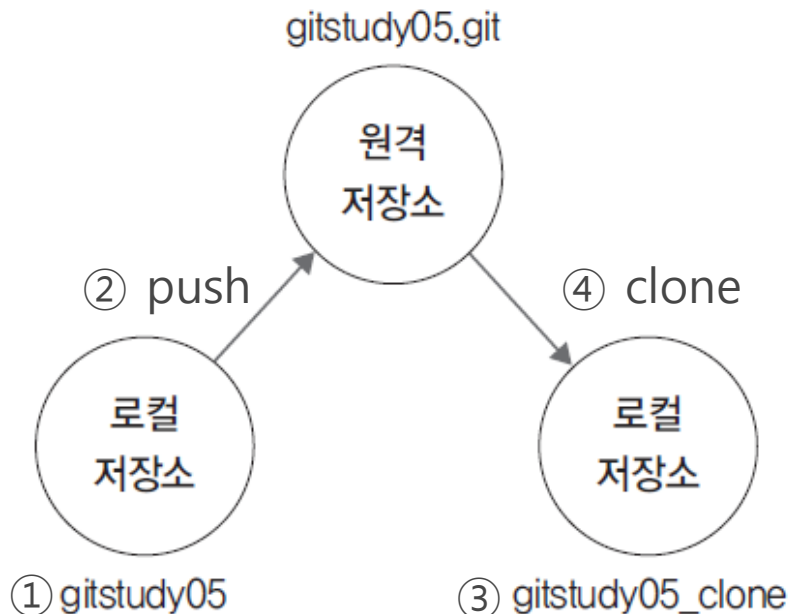


5. 자동으로 내려받기

➤ clone: 복제

- remote로 확인하면 연결된 서버 설정들이 표시됨
- 로컬 저장소를 생성한 후,
처음으로 서버에서 코드를 내려받을 때는 clone 명령어를 사용하면 좀 더 편리함

▼ 그림 5-9 원격 저장소와 로컬 저장소 연결





5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 복제는 원격 저장소에서 모든 내용을 한 번에 내려받음
- 복제 후 원격 저장소의 갱신된 내용을 추가로 내려받으려면 pull 명령어를 사용해야 함
- pull 명령어는 로컬 저장소보다 최신인 갱신된 원격 저장소의 커밋 정보를 현재 로컬 저장소로 내려받음
- pull 명령어를 주기적으로 사용하면 최신 커밋 정보로 로컬 저장소를 유지할 수 있음

```
$ git pull
```



5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 실습을 위해 원본 로컬 저장소로 이동함

```
infoh@gitstudy MINGW64 /e/gitstudy05_clone (master)
```

```
$ cd ../ gitstudy05 ----- 원본 폴더로 이동
```

```
infoh@gitstudy MINGW64 /e/gitstudy05 (master)
```

```
$ code server.htm ----- VS Code 실행
```

server.htm

```
<h1>서버 실습입니다.</h1>
```

```
<h2>오늘도 좋은 하루입니다.</h2>
```




5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 작성한 server.htm 파일을 커밋함

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git add server.htm
```

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git commit -m "good day"
```

```
[master 6a947b8] good day
```

```
1 file changed, 2 insertions(+)
```

```
create mode 100644 server.htm
```



5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 커밋한 내용을 원격 저장소(서버)로 업로드함

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
$ git push origin master ----- 푸시, 원격 서버로 전송
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 341 bytes | 341.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/jinygit/gitstudy05.git
4864581..6a947b8 master -> master
```



5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 방금 작성한 코드와 커밋으로 원격 저장소를 갱신함
- 갱신된 원격 저장소의 커밋을 복제한 저장소에도 동기화하기 위하여 복제된 저장소로 이동함
- 복제된 저장소에서 커밋 로그를 확인해보자.
 - 복제된 저장소에는 아직 하나의 커밋만 있음

```
$ cd ../gitstudy05_clone/ ----- 복제 저장소 폴더
```

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ git log
```

```
commit 486458111de5ec909e43460ec8ebf945ba9e932c (HEAD -> master, origin/master, origin/HEAD)
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:16:37 2019 +0900
```

```
first commit
```



5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 원격 저장소에서 갱신된 새 커밋 정보 가져오기
 - pull 명령어는 원격 저장소에서 갱신된 커밋을 로컬 저장소의 커밋 정보와 비교하여 갱신
 - 원격 저장소에서 복제된 저장소 동기화

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
$ git pull ----- 서버에서 정보를 가져오기
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/jinygit/gitstudy05
   4864581..6a947b8  master    -> origin/master
Updating 4864581..6a947b8
Fast-forward
 server.htm | 2 ++
1 file changed, 2 insertions(+)
create mode 100644 server.htm
```



5. 자동으로 내려받기

➤ pull: 서버에서 내려받기

- 이제 다시 복제된 저장소에서 log 명령어를 실행함

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ git log ----- 로그 확인
```

```
commit 6a947b89517661cde95c9bec4c766302a763437e (HEAD -> master, origin/master, origin/HEAD)
```

```
Author: hojinlee infohojin@gmail.com ----- 2번째 커밋 정보를 내려받았음
```

```
Date: Thu Dec 12 17:28:20 2019 +0900
```

```
good day
```

```
commit 486458111de5ec909e43460ec8ebf945ba9e932c
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:16:37 2019 +0900
```

```
first commit
```

5.6 수동으로 내려받기



6. 수동으로 내려받기

➤ 수동으로 내려받기

- 원격 저장소 내용을 내려받는 방법은 크게 pull(풀)과 fetch(페치) 두 가지임
- 이 두 방법의 차이는 병합을 자동 처리하는지 여부임



6. 수동으로 내려받기

➤ 자동 병합

- 풀은 원격 서버에서 현재 커밋보다 더 최신 커밋 정보가 있을 때 내려받음
- 내려받은 커밋 정보는 임시 영역에 저장함
- 스테이지 영역이 아닌 원격 저장소를 위한 전용 임시 브랜치가 따로 있음
- 내려받은 최신 커밋들을 현재 브랜치로 자동으로 병합 처리함
- 병합은 원격 서버 파일과 로컬 파일을 하나로 합치는 과정임
- 혼자서 개발하는 프로젝트는 pull 명령어만으로도 편리하게 사용할 수 있음
- 여러 개발자와 협업으로 코드를 작성할 때 pull 명령어를 사용한 자동 병합은 가끔씩 문제가 생김
- 여러 개발자와 협업하는 과정에서 pull 명령어가 자동으로 브랜치 병합을 하지 못하고 충돌이 발생하기도 함
- pull 명령어로 자동 병합을 하지 못할 때는 페치 방식을 사용해야 함



6. 수동으로 내려받기

➤ fetch: 가져오기

- fetch(페치)는 원격 저장소에서 코드를 수동으로 내려받는 작업을 함
- 페치는 원격 저장소에서 커밋된 코드를 임시 브랜치로 내려받음
- 내려받은 후 현재 브랜치와 자동 병합하지 않음

```
$ git fetch 원격저장소URL
```



6. 수동으로 내려받기

➤ fetch: 가져오기

- 그림 페치 동작을 실습해보자
- 먼저 실습 원본 저장소로 이동함

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ cd ../ gitstudy05 ----- 원본 폴더로 이동
```

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ code server.htm ----- VS Code 실행
```

server.htm

```
<h1>서버 실습입니다.</h1>
<h2>오늘도 좋은 하루입니다.</h2>
<h2>가끔은 하늘을 보세요.</h2>
```



6. 수동으로 내려받기

➤ fetch: 가져오기

- 수정된 파일을 스테이지 영역에 등록과 동시에 커밋함

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git commit -am "look sky"
```

```
[master 668b5ef] look sky
```

```
1 file changed, 1 insertion(+)
```



6. 수동으로 내려받기

➤ fetch: 가져오기

- 커밋된 수정 내용을 원격 저장소로 전송함

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ git push origin master ----- 커밋 서버 전송
```

```
Enumerating objects: 5, done.
```

```
Counting objects: 100% (5/5), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 369 bytes | 369.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0)
```

```
To https://github.com/jinygit/gitstudy05.git
```

```
6a947b8..668b5ef master -> master
```



6. 수동으로 내려받기

➤ fetch: 가져오기

- 원본 저장소 변경과 원격 저장소를 갱신함
- 이제는 다시 복제된 로컬 저장소로 이동하여 갱신된 원격 저장소 내용을 페치해 보자

```
infoh@hojin MINGW64 /e/gitstudy05 (master)
```

```
$ cd ../ gitstudy05_clone ----- 복제 폴더 이동
```

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ git fetch ----- 커밋 내려받기
```

```
remote: Enumerating objects: 5, done.
```

```
remote: Counting objects: 100% (5/5), done.
```

```
remote: Compressing objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
```

```
Unpacking objects: 100% (3/3), done.
```

```
From https://github.com/jinygit/gitstudy05
```

```
6a947b8..668b5ef master -> origin/master
```



6. 수동으로 내려받기

➤ fetch: 가져오기

- 원격 저장소의 커밋 내용을 페치한 후 커밋 로그를 확인함

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ git log ----- 로그 확인
```

```
commit 6a947b89517661cde95c9bec4c766302a763437e (HEAD -> master)
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:28:20 2019 +0900
```

```
good day
```

```
commit 486458111de5ec909e43460ec8ebf945ba9e932c
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:16:37 2019 +0900
```

```
first commit
```



6. 수동으로 내려받기

➤ fetch: 가져오기

- pull 명령어와 달리 fetch 명령어를 실행한 후에는 커밋이 추가된 것을 확인할 수 없음
- 페치는 원격 저장소의 커밋들만 가지고 왔을 뿐 로컬 저장소에서 어떤 작업도 하지 않음



6. 수동으로 내려받기

➤ merge 명령어로 수동 병합

- 페치는 데이터를 내려받기만 할 뿐 자동 병합하지 않음
- 내려받은 커밋을 로컬 저장소에 적용하려면 병합 명령을 실행해야 하는데, merge 명령어를 사용함
- merge 명령어는 다음과 같이 사용함

```
$ git merge 원격저장소별칭/브랜치이름
```




6. 수동으로 내려받기

➤ merge 명령어로 수동 병합

- fetch 명령어로 내려받은 커밋을 로컬 저장소에 병합
- 다음과 같이 merge 명령어를 입력함

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ git merge origin/master ----- 임시 원격 브랜치 병합
```

```
Updating 6a947b8..668b5ef
```

```
Fast-forward
```

```
server.htm | 1 +
```

```
1 file changed, 1 insertion(+)
```



6. 수동으로 내려받기

➤ merge 명령어로 수동 병합

- 이전과 다른 메시지들을 출력했지만, 어쨌든 잘 병합되었음
- 다시 로컬 저장소의 로그 기록을 확인함

```
infoh@hojin MINGW64 /e/gitstudy05_clone (master)
```

```
$ git log ----- 로그 확인
```

```
commit 668b5efb139f47d28e77d6cb99863e3961a5db1d (HEAD -> master, origin/master, origin/HEAD)
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:32:36 2019 +0900
```

```
look sky
```

```
commit 6a947b89517661cde95c9bec4c766302a763437e
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:28:20 2019 +0900
```

```
good day
```

```
commit 486458111de5ec909e43460ec8ebf945ba9e932c
```

```
Author: hojinlee <infohojin@gmail.com>
```

```
Date: Thu Dec 12 17:16:37 2019 +0900
```

```
first commit
```

5.7 순서



7. 순서

➤ 순서

- 원격 저장소에는 다수의 개발자가 동시에 커밋을 푸시할 수 없음
- 여러 명이 협력해서 개발할 때는 순차적으로 푸시해야 함



7. 순서

➤ 최신 상태

- 먼저 원격 저장소에 푸시하려면 자신의 로컬 저장소를 최신 상태로 유지해야 함
 - 자신의 저장소가 원격 저장소의 커밋 보다 최신이어야 한다는 의미임
 - 만약 누군가 내 저장소보다 먼저 커밋하여 새로운 커밋으로 서버를 갱신하면, 나는 간발의 차이로 갱신된 서버 정보를 가지고 있지 않게 됨
- 푸시는 서버의 마지막 커밋에 푸시할 때 로컬 저장소에 포함된 커밋을 병합
 - 이때 내 커밋은 서버의 최신 커밋보다 낮은 다음 커밋으로 등록됨
- 커밋이 순차적이지 않을 때 깃은 푸시 동작을 거부함
 - 푸시 동작이 거부되면 풀 또는 패치 작업으로 자신의 로컬 저장소를 갱신해 주어야 함
 - 갱신 후에 다시 푸시함



7. 순서

➤ 충돌 방지

- 깃이 최신 상태에서만 푸시를 허용하는 것은 충돌을 방지하기 위해서임
- 원격 저장소의 커밋을 내려받는 풀 작업은 내려받은 커밋들을 현재 브랜치로 자동 병합
 - 이때 커밋 내용이 순차적이지 않으면 병합 과정에서 충돌이 발생함
- 개발자 1과 개발자 2가 서로 다른 작업을 했다면 충돌이 없을 수도 있지만, 같은 영역을 동시에 수정했다면 개발자 2가 풀 작업을 할 때 무조건 충돌이 발생함
 - 개발자 2는 충돌을 해결한 후 커밋하고 푸시해야 하는데, 이를 해결하기가 어려움
- 깃은 이를 더 쉽게 해결할 수 있도록 푸시할 때 커밋의 순차 기록을 확인함



7. 순서

➤ 충돌 방지

- push 명령어를 사용할 때는 충돌을 최소화하기 위해 미리 원격 저장소 확인 필요함
- 새로운 커밋 갱신이 없는지 pull 명령어를 사용하여 지속적으로 확인하는 것이 좋음
- 최대한 충돌을 피할 수 있는 방법은
자신의 저장소와 원격 저장소의 상태를 자주 최신으로 유지하는 것임


▼ 그림 5-10 권장 순서

pull ➡ coding ➡ commit ➡ pull ➡ push



7. 순서

➤ 충돌 방지

- 풀과 푸시를 자주 실행하여 충돌을 최소한으로 줄여 나가면서 작업을 유지함
- 소스트리를 사용하면 푸시하기 전에 자신의 저장소와 어떤 차이점이 있는지 쉽게 확인할 수 있음
- 소스트리의 push  위에 조그마한 숫자가 표시됨
- 숫자는 현재 로컬 저장소와 원격 저장소 간 커밋 차이점을 출력함

5.8 정리



8. 정리

➤ 정리

- 깃은 코드 이력을 관리해 줄 뿐만 아니라 다른 개발자와 협업 도구로도 많이 사용함
- 다른 개발자와 협업하려면 공유 매개체의 역할을 수행할 서버가 필요함
- 깃은 다양한 종류의 서버를 지원함
- 깃 서버를 직접 만들 수도 있고, 인기 있는 깃 호스팅 서비스를 이용할 수도 있음
- 깃은 서버 역할을 수행하는 원격 저장소와 커밋 정보들을 주고받음
- 로컬 컴퓨터는 원격 저장소에 커밋 코드를 전송하거나 추가된 커밋들을 내려받을 수 있음
- 원격 저장소 기능은 좀 더 많은 사람이 깃을 사용하게 하는 촉매제가 됨
- 원격 저장소를 불특정 다수를 대상으로 공유할 수도 있음
- 오픈 소스는 깃과 공개된 원격 저장소를 사용하여 활발하게 수많은 사람과 협업할 수 있는 장점들을 제공함
- 깃은 오픈 소스를 활성화하는 데 가장 많은 기여를 하는 협업 툴이 됨