# GitHub 완전 정복 가이드(1)

## 1. Git과 GitHub, 무엇이 다른가요?

### Git: 버전 관리 시스템

* **Git**은 컴퓨터 파일의 변경사항을 추적하고 여러 사용자 간에 해당 파일 작업을 조율하기 위한 \*\*'버전 관리 시스템'\*\*입니다. 쉽게 말해, 프로젝트의 특정 시점들을 사진 찍듯이 저장하고, 원할 때마다 과거의 모습으로 돌아갈 수 있게 해주는 마법 같은 도구입니다. 내 컴퓨터에 설치해서 사용합니다.

### GitHub: Git 저장소 호스팅 서비스

* **GitHub**는 Git으로 관리하는 프로젝트를 다른 사람들과 공유하고 함께 작업할 수 있는 \*\*'웹 기반 호스팅 서비스'\*\*입니다. Git이 프로젝트의 역사를 기록하는 일기장이라면, GitHub는 그 일기장을 친구들과 함께 쓰고 돌려볼 수 있는 온라인 공간(클라우드 + 소셜 네트워크)인 셈이죠.

## 2. GitHub 기본 사용법: 이것만 알면 시작할 수 있어요!

### Repository (저장소)

* 프로젝트의 모든 파일과 버전 기록이 담겨 있는 폴더입니다. 줄여서 'Repo'라고도 부릅니다. GitHub에서 New repository 버튼을 눌러 새로운 프로젝트 저장소를 만들 수 있습니다.

### Commit (커밋)

* 프로젝트의 변경사항을 저장하는 작업 단위를 의미합니다. "이 부분을 이렇게 수정했어"라고 기록을 남기는 행위입니다. 각 커밋에는 변경사항을 설명하는 메시지를 함께 기록하여 어떤 내용이 바뀌었는지 쉽게 파악할 수 있습니다.

### Branch (브랜치)

* 독립적으로 어떤 작업을 진행하기 위한 '가지'입니다. 메인 가지(보통 main 또는 master 브랜치)를 그대로 둔 채, 새로운 기능 개발이나 버그 수정을 위한 자신만의 작업 공간을 만드는 것입니다. 여러 사람이 동시에 다른 작업을 해도 서로 영향을 주지 않아 안전합니다.

### Pull Request (풀 리퀘스트)

* 내가 만든 브랜치에서의 변경사항을 메인 브랜치에 합치고(Merge) 싶을 때 보내는 '요청'입니다. "제가 이런 기능을 만들었는데, 확인해 보시고 괜찮으면 반영해주세요!"라는 의미입니다. 다른 개발자들이 내 코드를 검토(리뷰)하고 피드백을 줄 수 있어 협업의 핵심 기능으로 꼽힙니다.

## 3. GitHub 고급 활용: 협업의 달인이 되어보세요!

### Fork (포크)

* 다른 사람의 공개 저장소를 내 계정으로 그대로 복사해오는 기능입니다. 원본에 영향을 주지 않고 자유롭게 코드를 수정하고 실험해볼 수 있습니다. 마음에 드는 수정이 이뤄졌다면, 원본 저장소에 Pull Request를 보내 기여할 수도 있습니다.

### Issues (이슈)

* 프로젝트의 할 일, 버그, 개선점 등을 기록하고 논의하는 공간입니다. 특정 이슈를 특정 담당자에게 할당하거나, 관련된 Pull Request와 연결하여 작업 진행 상황을 체계적으로 관리할 수 있습니다.

### Projects (프로젝트)

* 칸반 보드 스타일로 이슈와 Pull Request를 시각적으로 관리할 수 있는 도구입니다. 'To do', 'In progress', 'Done' 같은 열을 만들어 작업의 흐름을 한눈에 파악하기 좋습니다.

### GitHub Actions (액션)

* 커밋, Pull Request 등 특정 이벤트가 발생했을 때, 코드 테스트, 빌드, 배포 등의 작업을 자동으로 실행해주는 워크플로우 자동화 도구입니다. 반복적인 작업을 줄여 개발 생산성을 크게 높여줍니다.

## 4. 실습 예시: GitHub와 첫걸음 떼기

이제 이론을 배웠으니 직접 체험해볼 시간입니다! 아래 순서대로 따라 해보세요.

### 첫째, 내 저장소 만들고 자기소개하기

1. GitHub에 로그인한 후, 오른쪽 위 '+' 아이콘을 클릭하고 New repository를 선택합니다.
2. Repository name에 hello-github와 같이 원하는 이름을 입력합니다.
3. Public으로 설정하고, Add a README file 옵션을 체크한 후 Create repository 버튼을 클릭합니다.
4. 생성된 저장소에서 README.md 파일을 찾아 연필 모양(Edit) 아이콘을 클릭합니다.
5. 마크다운 문법을 이용해 간단한 자기소개를 작성하고, 페이지 하단의 Commit changes 버튼을 눌러 저장합니다.

### 둘째, 새로운 브랜치 만들고 코드 작성하기

1. 저장소 페이지에서 main이라고 표시된 브랜치 버튼을 클릭합니다.
2. Find or create a branch 입력창에 feature/add-hello와 같이 새로운 브랜치 이름을 입력하고 Create branch를 선택합니다.
3. 새로 만든 브랜치로 이동된 것을 확인한 후, Add file > Create new file을 클릭합니다.
4. 파일 이름에 hello.py를 입력하고, 내용에는 아래 코드를 작성합니다.print("Hello GitHub!")
5. Commit new file 버튼을 눌러 파일을 저장합니다.

### 셋째, Pull Request로 main 브랜치에 합치기

1. hello.py 파일을 커밋하고 나면, 페이지 상단에 노란색 배경의 알림창이 나타납니다. Compare & pull request 버튼을 클릭하세요.
2. Pull Request의 제목과 설명을 작성하고 Create pull request 버튼을 누릅니다.
3. 이제 내가 요청한 코드 변경사항을 검토할 수 있는 페이지가 열립니다. 실무에서는 동료가 코드를 리뷰해주겠지만, 지금은 직접 합쳐보겠습니다.
4. 초록색 Merge pull request 버튼을 클릭하고, Confirm merge를 클릭하여 변경사항을 main 브랜치에 최종 반영합니다.
5. 이제 main 브랜치로 돌아가 보면, hello.py 파일이 성공적으로 추가된 것을 확인할 수 있습니다!

**축하합니다!** 여러분은 방금 GitHub 협업의 가장 기본적인 과정을 모두 체험했습니다.

# 깃허브(GitHub) 핵심 기능, 쉽게 이해하기(2)

깃허브는 단순히 코드를 저장하는 공간을 넘어, 전 세계 개발자들과 협업하고 프로젝트를 체계적으로 관리할 수 있는 강력한 도구입니다. 처음에는 용어가 낯설 수 있지만, 하나씩 예시를 통해 알아보면 금방 익숙해질 수 있습니다.

### 1. Fork (포크) 🍴: 다른 사람의 맛집 레시피를 내 노트에 옮겨 적기

**개념:** 다른 사람의 공개 프로젝트(저장소)를 내 계정으로 그대로 복제하는 기능입니다. 마치 유명한 셰프의 레시피 북을 발견하고, 나만의 비법을 추가해보기 위해 내 요리 노트에 그대로 옮겨 적는 것과 같습니다.

* **원본에 영향 X:** 내 노트에 적은 레시피를 아무리 바꿔도 원본 셰프의 레시피 북은 그대로입니다. 안전하게 마음껏 실험해볼 수 있죠.
* **기여(Contribution):** 만약 제가 수정한 레시피가 원본보다 훨씬 맛있어졌다면, 셰프에게 "제 레시피도 한번 보실래요?"라고 제안(Pull Request)할 수 있습니다.

**쉬운 예시:**

친구 'A'가 만든 my-first-website라는 멋진 웹사이트 프로젝트를 깃허브에서 발견했습니다. 저는 이 웹사이트의 배경색을 파란색에서 초록색으로 바꿔보고 싶습니다.

1. 친구 A의 my-first-website 저장소 오른쪽 위에 있는 **Fork** 버튼을 누릅니다.
2. 이제 제 깃허브 계정에 내이름/my-first-website라는 똑같은 저장소가 생깁니다. (나만의 요리 노트가 생겼습니다!)
3. 저는 이제 내이름/my-first-website 저장소에서 배경색을 초록색으로 바꾸는 코드를 자유롭게 수정하고 저장(Commit)합니다.
4. 이 과정은 친구 A의 원본 프로젝트에 아무런 영향을 주지 않습니다.

### 2. Issues (이슈) 🎫: 프로젝트의 할 일 목록 및 버그 신고 센터

**개념:** 프로젝트의 버그, 개선 아이디어, 질문 등을 기록하고 논의하는 공간입니다. 마치 프로젝트의 공식 '게시판'이나 '고객센터'와 같습니다. 할 일을 티켓처럼 만들어서 관리할 수 있습니다.

* **체계적인 관리:** 누가 어떤 일을 할지 담당자를 지정하고, 진행 상황을 댓글로 남길 수 있습니다.
* **투명한 소통:** 프로젝트에 참여하는 모든 사람이 어떤 논의가 오가는지 투명하게 볼 수 있습니다.

**쉬운 예시:**

제가 만든 간단한-계산기 프로젝트를 친구들이 사용해보고 의견을 줍니다.

* **버그 신고:** 친구 B가 "10 나누기 0을 하면 앱이 꺼져버려!"라는 버그를 발견했습니다. 친구 B는 간단한-계산기 저장소의 **Issues** 탭에 가서 "0으로 나눌 때 발생하는 오류"라는 제목으로 이슈를 등록합니다.
* **기능 제안:** 친구 C는 "여기에 제곱 기능도 추가하면 좋겠어!"라는 아이디어를 줍니다. Issues 탭에 "제곱 기능 추가 제안"이라는 새 이슈를 만듭니다.

이제 저는 이 이슈들을 보고, "0으로 나누기 오류" 이슈를 저에게 할당(assign)한 뒤, 코드를 수정하여 버그를 해결하고 이슈를 닫습니다(close).

### 3. Projects (프로젝트) 📋: 할 일을 시각적으로 보여주는 화이트보드

**개념:** 위에서 만든 이슈(할 일 티켓)들을 칸반 보드 스타일로 정리하는 도구입니다. 마치 사무실 화이트보드에 '해야 할 일', '진행 중', '완료' 섹션을 나누고 포스트잇을 붙여 관리하는 것과 같습니다.

* **진행 상황 파악:** 프로젝트 전체의 흐름을 한눈에 파악할 수 있습니다.
* **유연한 관리:** 드래그 앤 드롭으로 카드를 옮기며 손쉽게 상태를 변경할 수 있습니다.

**쉬운 예시:**

간단한-계산기 프로젝트에 할 일이 많아졌습니다. 그래서 **Projects** 탭에서 '계산기 업그레이드'라는 새 프로젝트 보드를 만듭니다.

1. **'To do' 열:** "0으로 나누기 오류 수정", "제곱 기능 추가" 등 Issues에 등록된 할 일 카드들을 가져다 놓습니다.
2. **'In progress' 열:** 제가 "0으로 나누기 오류 수정" 작업을 시작하면, 해당 카드를 'In progress' 열로 드래그해서 옮깁니다.
3. **'Done' 열:** 버그 수정이 완료되어 코드에 반영되면, 카드를 'Done' 열로 옮깁니다.

이제 팀원 누구나 이 보드만 보면, 어떤 일이 남아있고 지금 무엇을 하고 있는지 바로 알 수 있습니다.

[a Kanban board 이미지](https://encrypted-tbn2.gstatic.com/licensed-image?q=tbn:ANd9GcSSOvWGOb5Rj4j61bxvBoYMEUTixZjfwAlUgIWDyGDFxX7zrPPijX7jxDsUn2-v6Tn66la3APDQ56508W-lO644VyG_QV-TIIK2e6Rpgx6rx6jSr6g)

### 4. GitHub Actions (액션) 🤖: 단순 반복 작업을 대신해주는 로봇

**개념:** 특정 조건이 만족되면, 정해진 작업들을 자동으로 실행해주는 자동화 도구입니다. 마치 제가 "새로운 코드가 올라오면, 자동으로 테스트하고, 문제가 없으면 나에게 알려줘"라고 설정해둔 똑똑한 로봇 비서와 같습니다.

* **생산성 향상:** 코드 테스트, 빌드, 배포 등 귀찮고 반복적인 작업을 자동화하여 개발자는 더 중요한 일에 집중할 수 있습니다.
* **실수 방지:** 사람이 직접 할 때 발생할 수 있는 실수를 줄여 프로젝트의 안정성을 높입니다.

**쉬운 예시:**

제 my-first-website 프로젝트에 새로운 코드가 추가될 때마다, 웹사이트가 제대로 표시되는지 수동으로 확인하는 것은 매우 번거롭습니다.

1. **자동화 규칙 설정 (워크플로우):** GitHub Actions에 ".github/workflows" 폴더를 만들고 규칙 파일을 작성합니다.
2. **규칙 내용:** "만약 main 브랜치에 새로운 코드가 Push(업로드) 되면, 자동으로 내 코드를 테스트하고, 테스트에 통과하면 gh-pages라는 주소에 자동으로 웹사이트를 배포(게시)해줘."

이제 제가 새 코드를 올리기만 하면, 깃허브 로봇이 알아서 테스트부터 배포까지의 모든 과정을 자동으로 처리해줍니다. 저는 커피 한잔하며 기다리기만 하면 됩니다!

# 깃허브 협업, "이것만 알면 문제없어요!" 캔버스 가이드 (3)

깃허브로 다른 사람들과 함께 프로젝트를 진행할 때 흔히 겪는 문제들을 알기 쉬운 비유와 함께, 그리고 **어떤 깃허브 기능으로 해결할 수 있는지** 정리했어요.

### 1. 코드 충돌: "하나의 레시피를 동시에 고치기" 🧑‍🍳💥👩‍🍳

**상황:** 셰프 두 명이 같은 요리책의 같은 줄을 동시에 수정하는 상황이에요.

* **A 셰프:** "소금 한 스푼"을 "설탕 한 스푼"으로 고침
* **B 셰프:** "소금 한 스푼"을 "간장 한 스푼"으로 고침

문제는?

요리책(GitHub) 입장에서는 누구의 말을 들어야 할지 알 수 없어요. 이게 바로 \*\*충돌(Conflict)\*\*이에요.

해결책은?

두 셰프가 직접 만나 상의하는 거예요.

개발자가 직접 충돌 난 코드를 보고 **어떤 버전을 남길지 결정하거나, 두 코드를 적절히 합쳐서** 새로운 코드를 만들어주면 해결돼요.

**💡 깃허브 기능 활용하기:**

* **Pull Request 충돌 해결 UI:** 깃허브 웹사이트의 Pull Request 페이지에서는 간단한 충돌의 경우, 어떤 코드를 선택할지 시각적으로 보여주고 버튼 클릭만으로 해결할 수 있게 도와줘요.
* **토론(Discussion) 기능:** 어떤 코드를 선택해야 할지 애매하다면, Pull Request 내의 댓글 기능을 이용해 동료와 충분히 상의한 후 결정할 수 있어요.

### 2. 버전 관리 혼란: "어지럽게 뻗은 나뭇가지" 🌳🤯

**상황:** 커다란 나무(프로젝트)에 여러 가지(브랜치)를 만들어서 동시에 작업을 진행해요.

* main 줄기: 가장 안정적인 버전
* develop 가지: 개발 중인 버전

문제는?

가지를 너무 많이 만들고 제때 정리하지 않으면, 어떤 가지가 최신인지, 어떤 가지에서 작업을 이어가야 할지 혼란스러워져요.

해결책은?

나무에 팻말을 달아주는 거예요.

명확한 **규칙(Git-flow 등)을 정하고** 꾸준히 브랜치를 정리하면 프로젝트의 흐름을 한눈에 파악할 수 있어요.

**💡 깃허브 기능 활용하기:**

* **브랜치 보호 규칙(Branch Protection Rule):** main이나 develop 같은 핵심 브랜치에 실수로 코드를 올리는 것을 막을 수 있어요. 'PR 승인 없이는 병합 불가', '특정 테스트 통과 필수' 같은 규칙을 설정해 안정성을 높여요.
* **네트워크 그래프(Network Graph):** Insights 탭에서 브랜치들이 어떻게 나뉘고 합쳐졌는지 시각적으로 보여줘서, 복잡한 프로젝트의 흐름을 쉽게 파악할 수 있어요.

### 3. 보안 문제: "공개된 장소에 집 열쇠 두기" 🔑👀

**상황:** 집 주소와 현관 비밀번호가 적힌 쪽지를 누구나 볼 수 있는 광장 게시판에 붙여놓는 것과 같아요.

문제는?

악의적인 해커가 이 정보를 보고 우리 집(서버, 데이터베이스)에 마음대로 들어와 중요한 정보를 훔치거나 망가뜨릴 수 있어요.

해결책은?

중요한 물건은 비밀금고에 넣어야죠.

* **.gitignore 파일**: "이 파일들은 절대 올리지 마!"라고 목록을 만들어 민감 정보가 실수로 올라가는 것을 막아요.

**💡 깃허브 기능 활용하기:**

* **GitHub Secrets:** 중요한 API 키나 비밀번호를 코드와 분리하여 깃허브의 '비밀 보관소'에 안전하게 저장해요. GitHub Actions 작업 시에만 안전하게 불러와 사용할 수 있습니다.
* **보안 스캐닝(Secret Scanning):** 깃허브는 저장소에 올라온 코드를 자동으로 검사해서 혹시라도 실수로 올린 API 키나 비밀번호가 있는지 찾아내 개발자에게 경고 알림을 보내줘요.

### 4. 협업 소통 부족: "서로 말없이 집 짓기" 🏠🧱❓

**상황:** 건축가 두 명이 서로 상의 없이 각자 벽을 쌓고 문을 달아요. 결국 문을 열 수 없는 이상한 집이 완성되죠.

문제는?

동료가 어떤 작업을 하는지 모르고 내 코드만 수정해서 합치면, 프로그램 전체가 제대로 동작하지 않는 버그가 생길 수 있어요.

해결책은?

집을 짓기 전에 설계도를 함께 보고 이야기하는 거예요.

**활발한 코드 리뷰 문화**는 버그를 사전에 예방하고 더 좋은 품질의 코드를 만들게 도와줘요.

**💡 깃허브 기능 활용하기:**

* **Pull Request (PR) 기능:** 코드 변경 사항을 합치기 전, 동료에게 검토를 요청하는 공식적인 절차예요.
  + **리뷰어 지정(Request Review):** 특정 동료를 지정해 코드 리뷰를 요청할 수 있어요.
  + **인라인 코멘트(Inline Comments):** 코드의 특정 줄에 직접 댓글을 달아 피드백을 주고받을 수 있어요.
  + **변경 제안(Suggest Changes):** 리뷰어가 더 좋은 코드를 직접 제안하고, 작성자는 클릭 한 번으로 바로 적용할 수 있어요.

### 5. 오픈소스 관리: "스팸으로 가득 찬 아이디어 함" 📥📬

**상황:** 마을 발전을 위해 누구나 아이디어를 낼 수 있는 '아이디어 함'을 설치했지만, 의미 없는 쪽지나 광고 전단지도 마구 들어와요.

문제는?

전 세계 개발자가 참여하는 오픈소스 프로젝트에는 품질이 낮은 코드가 너무 많이 제출되어 관리자가 좋은 기여를 선별하느라 지치게 돼요.

해결책은?

아이디어 함 앞에 '제출 양식'과 '가이드라인'을 붙여놓는 거예요.

* **CONTRIBUTING.md 파일**: "우리 프로젝트에 기여하려면 이렇게 해주세요!"라고 상세한 안내서를 제공해요.

**💡 깃허브 기능 활용하기:**

* **PR/Issue 템플릿:** 어떤 내용을 적어야 하는지 양식을 미리 만들어두면, 기여자들이 체계적으로 문제를 제기하거나 코드를 제출할 수 있어요.
* **GitHub Actions:** 제출된 PR이 코딩 스타일을 지켰는지, 테스트는 통과하는지 등을 자동으로 검사하게 만들 수 있어요. 이를 통해 관리자는 기본적인 검토 작업을 자동화하고 더 중요한 문제에 집중할 수 있습니다.
* **라벨(Labels):** '버그', '기능 추가', '도움 필요' 등 라벨을 붙여 수많은 이슈와 PR을 체계적으로 분류하고 관리할 수 있어요.

### 6. 대용량 파일 문제: "작은 우편함에 냉장고 넣기" 📮📦

**상황:** 일반 편지를 주고받는 작은 우편함(Git)에 거대한 냉장고(대용량 파일)를 억지로 넣으려고 하는 것과 같아요.

문제는?

Git은 텍스트 기반의 작은 변화를 관리하는 데 특화되어 있어요. 영상, 데이터셋 같은 큰 파일을 올리면 저장소가 무거워지고 속도가 매우 느려져요.

해결책은?

냉장고는 화물 배송 서비스를 이용해야죠.

**💡 깃허브 기능 활용하기:**

* **Git LFS (Large File Storage):** '대용량 파일 전용 배송 서비스'예요. 실제 파일은 다른 곳에 저장하고, Git에는 "이 파일은 저쪽에 있어"라는 작은 포인터(주소)만 남겨둬서 저장소를 가볍게 유지해요.
* **GitHub Releases:** 컴파일된 실행 파일, 압축된 데이터셋처럼 사용자에게 최종적으로 배포해야 할 큰 파일들은 'Releases' 기능을 통해 버전별로 묶어서 제공할 수 있어요.

# GitHub 첫걸음: 상세 실습 가이드(4)

이 가이드는 GitHub를 처음 사용하는 분들이 기본적인 작업 흐름을 익힐 수 있도록 돕기 위해 만들어졌습니다. 아래 순서를 차근차근 따라 해보세요.

### 첫째, 내 저장소 만들고 자기소개하기

내 프로젝트를 보관할 온라인 공간인 '저장소(Repository)'를 만들고, 프로젝트의 대문 역할을 하는 README 파일에 간단한 자기소개를 작성해봅니다.

**1. 새 저장소 만들기 시작**

* GitHub에 로그인한 후, 화면 오른쪽 위에 있는 + 아이콘을 클릭합니다.
* 드롭다운 메뉴에서 **New repository**를 선택하여 새 저장소 만들기 페이지로 이동합니다.

**2. 저장소 이름 및 설정**

* **Repository name** 필드에 hello-github와 같이 원하는 프로젝트 이름을 입력합니다. 이 이름은 URL의 일부가 됩니다.
* **Description** (선택 사항) 필드에는 "첫 번째 깃허브 저장소"와 같이 간단한 설명을 추가할 수 있습니다.
* 저장소 공개 범위를 **Public**으로 설정합니다. (누구나 코드를 볼 수 있습니다.)
* **"Add a README file"** 옵션에 반드시 체크합니다. 이 파일은 저장소의 소개 페이지 역할을 합니다.
* 모든 설정을 마쳤으면 초록색 **Create repository** 버튼을 클릭합니다.

**3. README 파일 수정하기**

* 저장소가 성공적으로 만들어지면 README.md 파일이 보입니다.
* 파일 목록에서 README.md를 클릭한 후, 파일 내용 오른쪽 위에 있는 **연필 모양(Edit this file) 아이콘**을 클릭합니다.

**4. 자기소개 작성 (마크다운)**

* 편집 화면이 나타나면 마크다운 문법을 사용하여 자유롭게 자기소개를 작성합니다. 예를 들어 아래와 같이 작성할 수 있습니다.# 안녕하세요! 제 이름은 OOO입니다.  
    
  ## 취미  
  - 코딩하기  
  - 음악 감상  
    
  ## 관심 분야  
  - 프론트엔드 개발  
  - 데이터 시각화  
    
  GitHub를 통해 성장하는 개발자가 되겠습니다!

**5. 변경사항 저장 (커밋)**

* 자기소개 작성이 끝나면 페이지 하단으로 스크롤합니다.
* **Commit changes** 섹션이 보일 겁니다. 첫 번째 입력란에는 "Add introduction"과 같이 변경사항에 대한 요약을, 두 번째 입력란에는 더 자세한 설명을 적을 수 있습니다. (지금은 기본값을 그대로 두어도 괜찮습니다.)
* **Commit changes** 버튼을 눌러 파일 수정을 최종 저장합니다. '커밋(Commit)'은 내 변경사항을 하나의 버전으로 기록하는 행위를 의미합니다.

### 둘째, 새로운 브랜치 만들고 코드 작성하기

'브랜치(Branch)'라는 독립된 작업 공간을 만들어 기존 코드에 영향을 주지 않고 새로운 기능을 추가하는 방법을 배웁니다.

**6. 브랜치 메뉴 열기**

* 저장소 페이지의 파일 목록 위, 초록색 <> Code 버튼 옆을 보면 \*\*main\*\*이라고 표시된 버튼이 있습니다. 이것이 현재 브랜치를 나타냅니다. 이 버튼을 클릭하세요.

**7. 새 브랜치 생성하기**

* "Find or create a branch"라는 텍스트 상자가 나타납니다.
* 여기에 feature/add-hello와 같이 새로운 브랜치 이름을 입력합니다. 일반적으로 기능/작업내용 형식으로 이름을 짓습니다.
* 이름 아래에 나타나는 \*\*Create branch: feature/add-hello from 'main'\*\*을 클릭합니다. 'main' 브랜치의 복사본으로 새 브랜치를 만들겠다는 의미입니다.

**8. 새 파일 추가하기**

* 이제 화면의 브랜치 표시가 feature/add-hello로 바뀐 것을 확인할 수 있습니다.
* 파일 목록 위 Add file 버튼을 클릭하고, 드롭다운 메뉴에서 **Create new file**을 선택합니다.

**9. 파이썬 코드 작성**

* 파일 이름 입력란에 hello.py라고 입력합니다.
* 아래 코드 편집기에 다음 코드를 그대로 입력합니다.print("Hello GitHub!")

**10. 새 파일 저장 (커밋)**

* 페이지 하단으로 내려가 **Commit new file** 버튼을 클릭하여 새 파일을 feature/add-hello 브랜치에 저장합니다.

### 셋째, Pull Request로 main 브랜치에 합치기

새 브랜치에서 만든 변경사항을 원래의 main 브랜치에 합쳐달라고 요청하는 '풀 리퀘스트(Pull Request)' 과정을 체험합니다.

**11. Pull Request 생성 시작**

* hello.py 파일을 커밋하고 나면, 저장소 페이지 상단에 \*\*"feature/add-hello had recent pushes"\*\*라는 노란색 알림창이 뜹니다.
* 이 알림창 오른쪽에 있는 초록색 **Compare & pull request** 버튼을 클릭합니다.

**12. Pull Request 내용 작성**

* Pull Request(PR) 생성 페이지가 열립니다.
* **Title**(제목)에는 "Add hello.py for greeting"과 같이 이 PR이 어떤 작업을 했는지 요약해서 적습니다.
* **Leave a comment** 영역에는 변경된 내용에 대한 상세한 설명을 적습니다. (지금은 간단히 "파이썬 인사 기능 추가"라고 적어보세요.)
* **Create pull request** 버튼을 클릭합니다.

**13. Pull Request 검토 및 병합**

* 이제 내가 요청한 코드 변경사항을 검토하는 페이지가 나타납니다. Files changed 탭을 누르면 hello.py가 추가된 것을 눈으로 확인할 수 있습니다.
* 실무에서는 동료 개발자가 이 페이지에서 코드 리뷰를 해줍니다. 오늘은 직접 합쳐보겠습니다.
* 다시 Conversation 탭으로 돌아와 아래로 스크롤하면 보이는 초록색 **Merge pull request** 버튼을 클릭합니다.

**14. 병합 최종 확인**

* 정말로 병합할 것인지 확인하는 창이 나타납니다. **Confirm merge** 버튼을 클릭합니다.
* 보라색으로 "Merged" 표시가 나타나면 성공적으로 feature/add-hello 브랜치의 내용이 main 브랜치로 합쳐진 것입니다.
* 이제 작업이 끝난 브랜치는 삭제해도 좋습니다. **Delete branch** 버튼을 클릭해 feature/add-hello 브랜치를 정리합니다.

**15. 최종 결과 확인**

* 페이지 상단의 저장소 이름을 클릭하거나, <> Code 탭을 눌러 저장소의 메인 페이지로 돌아갑니다.
* 브랜치가 main으로 설정되어 있는지 확인하고 파일 목록을 보세요.
* hello.py 파일이 성공적으로 추가된 것을 확인할 수 있습니다!

축하합니다! GitHub의 가장 기본적인 협업 흐름인 **Branch -> Commit -> Pull Request -> Merge** 과정을 모두 경험해보셨습니다.