

IDG Deep Dive

프로그래머 개발 도우미 **‘깃허브’ 완전정복**

2018년판

올해는 깃허브 탄생 10주년이다. 2007년 미국 샌프란시스코에서 처음 서비스를 시작한 이후 이제는 세계 최대 코드 저장소로 자리를 잡았다. 현재 깃허브에 만들어진 저장소는 8,500만 개, 가입자는 2,800만 명에 달한다. 오픈소스는 물론 상용 소프트웨어 개발업체 상당수도 깃허브에 둑지를 틀었다. 사실상 소프트웨어 개발의 산실이라고 해도 과언이 아니다. 깃허브 생태계에 대한 전반적인 이해부터 기본 활용법, 버전관리 개념을 알아본다. 지난 10년간의 성과와 한계를 정리하고, 앞으로의 10년을 전망한다. 특히 최근 마이크로소프트에 인수된 이후 제기되고 있는 우려에 대해서도 살펴본다.

▣ Tech Report

‘다시 처음부터’ 깃허브 생태계의 이해

▣ Tech Guide

‘가입부터 커밋까지’ 깃허브 기본 활용법

‘깃허브 초보 탈출’ 깃 버전 관리의 이해

▣ Tech Story

깃허브의 지난 10년과 앞으로의 10년

MS의 깃허브 인수, 개발자가 우려해야 할까?



무단 전재
재배포 금지

본 PDF 문서는 IDG Korea의 자산으로, 저작권법의 보호를 받습니다.

IDG Korea의 허락 없이 PDF 문서를 온라인 사이트 등에 무단 게재, 전재하거나 유포할 수 없습니다.

'다시 처음부터' 깃허브 생태계의 이해

Tech
Report

Martin Heller | InfoWorld

깃 허브(GitHub)는 깃(Git) 리포지토리 호스팅 서비스, 즉 클라우드 기반 소스 코드 관리 또는 버전 제어 시스템의 핵심이다. 그러나 이는 시작에 불과하다. 그 외에 깃허브는 풀(pull) 요청, 디프(diff) 및 리뷰 요청 등 코드 리뷰를 위한 기능을 지원하고, 이슈 추적 및 할당을 포함한 프로젝트 관리, 다른 개발자 툴, 팀 관리, 문서, '소셜 코딩' 과의 통합 기능도 제공한다.

또한 깃허브는 프로그래머를 위한 일종의 소셜 네트워킹 사이트라고 할 수 있다. 프로그래머가 자유롭게 오픈 소스 코드를 공유하고 협업할 수 있는 개방형 환경이다. 깃허브에서 손쉽게 유용한 코드를 찾아 리포지토리를 복사해서 사용하고 다른 사람의 프로젝트에 대한 변경 사항을 제출할 수 있다. 그 결과 깃허브는 거의 모든 오픈 소스 프로젝트의 본가와 같은 존재가 됐다. 필자는 오픈 소스 프로젝트를 살펴볼 때 가장 먼저 프로젝트 이름을 검색한다. 프로젝트 웹사이트에서 코드 리포지토리 링크를 확인하는데, 십중팔구는 깃허브로 연결된다.

깃 버전 관리

깃허브가 어떤 일을 하고 어떻게 작동하는지 이해하려면 먼저 깃을 이해해야 한다. 깃은 분산 버전 제어 시스템으로, 2005년 리누스 토발스가 리눅스 커널 개발을 위해 커뮤니티의 도움을 받아 만들었다. 깃이 얼마나 빠르고 작고 유연하고 인기가 있는지에 대한 장황한 설명은 생략하겠지만, 깃 리포지토리(짧게 '리포(Repo)'라고 함)를 복제해 보면 한 시점의 한 분기(fork) 스냅샷이 아니라 전체 버전 히스토리가 컴퓨터에 저장된다.

깃은 리눅스 커널 커뮤니티에 뿌리를 둔 만큼 처음에는 명령줄 도구였다. 지금도 원한다면 깃 명령줄을 사용할 수 있다. 그러나 그럴 필요는 없다. 명령줄 대신 또는 명령줄에 더해서 윈도우와 맥에서 무료 깃허브 클라이언트를 사용하거나 그 외의 다양한 깃용 GUI 또는 깃과 통합되는 코드 편집기가 있다. 모두 처음에 익힐 때 명령줄보다 사용하기 쉽다. 깃 명령줄은 맥과 리눅스 시스템 대부분에서 사전 설치된 상태로 제공되며 모든 작업을 지원한다. GUI는 일반적으로 많이 사용되는 깃 작업의 하위 집합을 지원한다.

깃은 중앙화가 아닌 분산된다는 점에서 서브버전(Subversion) 같은 버전 제어 시스템과 차별화된다. 또한 작업 대부분이 로컬 리포지토리에서 실행되므로 상당히 빠르다. 그러나 깃을 사용하면 복잡한 계층이 하나 추가된다. 로컬 리포지토리에 코드를 커밋하고 커밋을 원격 리포

지토리에 푸시하는 작업이 별개 단계로 구성되기 때문이다. 팀에서 이 부분을 잊을 경우(또는 이에 대해 교육을 받지 못한 경우) 여러 개발자가 분기된 코드 베이스로 작업을 하는 상황이 발생할 수 있다.

원격 깃 리포지토리의 위치는 서버일 수도, 다른 개발자의 시스템일 수도 있다. 덕분에 팀에서 여러 가지 용도로 사용할 수 있다. 예를 들어 리뷰와 철저한 테스트를 거친 코드만 개발자 리포지토리에서 발행된 풀 요청을 통해 커밋할 수 있는 ‘기준’ 리포지토리로 서버 리포지토리를 사용하는 방법이 있다.

깃허브 기능

깃허브는 코드 호스팅과 소셜 코딩을 위한 클라우드 기반 깃 서버이며 코드 리뷰, 프로젝트 관리, 다른 개발자 툴, 팀 관리 및 문서와의 통합을 구현한다.

깃허브 소셜 코딩의 최신 기능은 커밋 공동 작성자다. 커밋 메시지 끝에 하나 이상의 ‘공동 작성자’ 트레일러를 추가할 수 있다. 이 메커니즘은 리포 코어에 영향을 미치지 않고 일반 깃에서 리포의 형태도 바꾸지 않지만 깃허브의 커밋 목록에 여러 명의 커미터가 표시되며 기여 그래프를 통해 각 공동 작성자의 커밋 참여를 명시한다. 필요에 따라 깃허브 그래프QL(GraphQL) API (<https://developer.github.com/v4>)를 사용해 깃허브를 확장할 수도 있다. REST 호출을 기반으로 했던 깃허브의 이전 API에 비해 크게 개선된 점이다.

깃허브 엔터프라이즈

깃허브닷컴(GitHub.com)(<https://github.com>)은 다양한 계정 유형의 클라우드 호스팅 서비스(<https://github.com/pricing>)를 제공한다. 무료(공개 리포만 해당)부터 유료(월 7달러), 개발자 계정, 팀(사용자당 월 9달러), 비즈니스(사용자당 월 21달러) 등이 있다. 온프레미스 또는 AWS, 마이크로소프트 애저, 구글 클라우드 플랫폼, IBM 클라우드의 자체 클라우드 인스턴스에서 깃허브 엔터프라이즈(<https://enterprise.github.com/home>)를 사용하려면, 호스팅형 비즈니스 계정처럼 사용자당 월 21달러를 내면 된다. 깃허브 엔터프라이즈는 사용자에게 보내는 앱 내 메시징, LDAP 디렉토리와 통합된 액세스 프로비저닝 등 몇 가지 유용한 기능을 추가로 제공한다. 그러나 깃허브닷컴의 호스팅형 비즈니스 계정에 적용되는 99.95% 업타임 SLA

는 적용되지 않는다.

깃허브 대 빗버킷

호스팅 방식의 더 강화된 깃 서비스는 깃허브 외에도 다양하다. 기업용 온프레미스 제품 역시 깃허브 엔터프라이즈가 유일한 것은 아니다. 예를 들면, 아틀라시안 빗버킷(Atlassian Bitbucket)은 깃허브, 깃허브 엔터프라이즈 양쪽 모두의 경쟁자다. 가격이 약간 더 저렴하고 회원 5명의 무료 팀 레벨을 제공한다. 무료 레벨에서는 무제한 프라이빗 리포와 지속적 통합을 위한 빗버킷 파이프라인을 지원한다. 깃허브는 오픈 소스 프로젝트에서 더 인기 있고 오픈

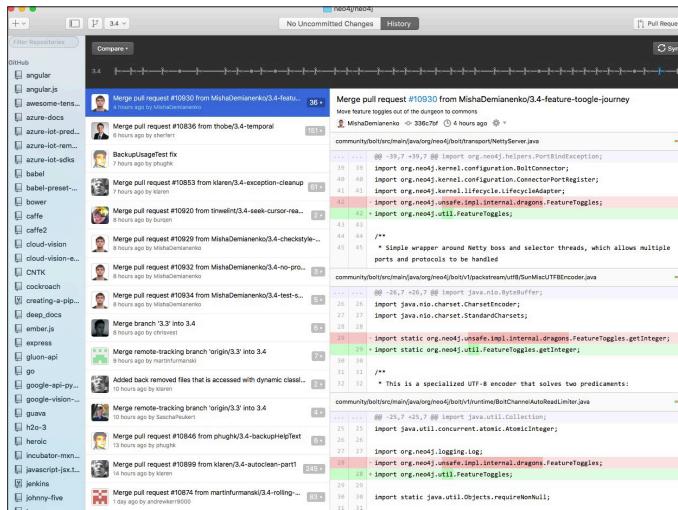


Credit : Daniel X. O'Neil/flickr

소스 개발자 풀도 훨씬 더 크다. 그러나 소규모 신생 업체에는 빗버킷의 가격이 더 매력적이다.

깃허브 대 깃랩

깃랩(GitLab)은 깃허브, 빙버킷과 호스팅, 온프레미스 등에서 모두 경쟁 관계에 있다. 표면적으로 깃랩은 다른 플랫폼보다 라이프사이클 관련 기능이 더 많지만, 빙버킷을 평가할 때 지라(Jira)를 포함하면 아틀라시안과 깃랩의 차이점은 대부분 사라진다. 깃랩은 오픈 소스 프로젝트에 무료로 골드(Gold) 플랜 클라우드 기능을 제공한다. 그러나 이 부가 기능의 장점이 깃허브의 더 방대한 오픈 소스 개발자 커뮤니티를 상쇄할 정도는 아니다.



화면 1 | 깃허브 데스크톱은 리포 추가 또는 복제, 분기 탐색, 변경 사항 푸시, 풀 요청 관리를 위한 유용한 GUI를 제공한다.

톱에 추가하고 원하는 변경 사항을 커밋하고 작업을 테스트하고 커밋을 분기된 원격 리포에 다시 커밋한 다음 마지막으로 상위 프로젝트에 풀 요청을 한다. 이를 위한 풀 요청(Pull Request) 버튼이 깃허브 데스크톱 인터페이스의 오른쪽 위에 있다. 또한 분기 또는 풀 요청의 병합인 Neo4j 프로젝트에서도 많은 커밋을 볼 수 있다. 커미터는 극소수고 기여자는 많은 오픈 소스 프로젝트의 전형적인 모습이다.

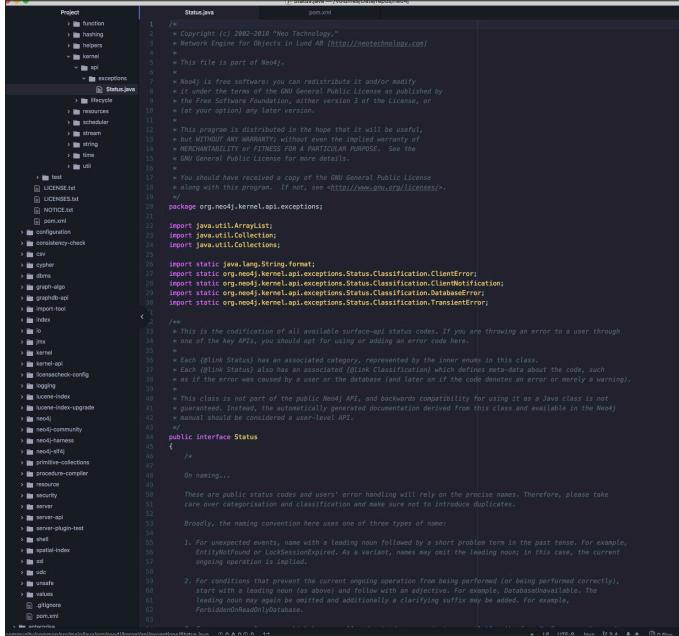
아톰 편집기

개발자는 자신이 원하는 프로그래밍 편집기에서 깃허브 코드를 편집할 수 있다. 대표적인 것이 무료 오픈 소스 편집기인 아톰(Atom)(<https://atom.io>)이다. 깃허브, 깃허브 데스크톱과도 잘 통합된다. 맥OS, 윈도우 또는 리눅스에서 사용할 수 있다. 깃허브 데스크톱에서 탐색 또는 편집할 리포지토리를 마우스 오른쪽 버튼으로 클릭해서 아톰으로 열 수 있다.

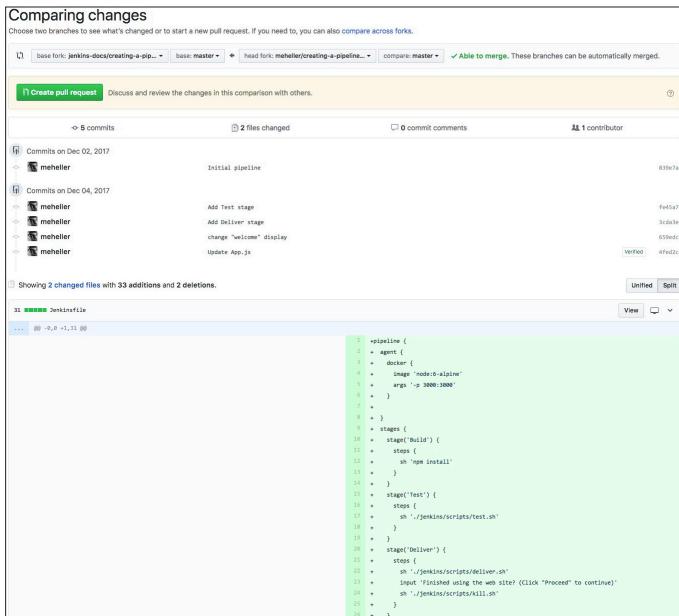
깃허브 데스크톱

깃허브 데스크톱(GitHub Desktop)(<https://desktop.github.com>)은 깃허브닷컴과 깃허브 엔터프라이즈 리포지토리를 쉽게 관리할 수 있는 툴이다. 깃 명령줄과 깃허브 웹 GUI의 모든 기능을 지원하지는 않지만 프로젝트에 기여하면서 데스크톱에서 일상적으로 하는 작업은 모두 구현한다. 예를 들면 깃허브에서 깃허브 데스크톱으로 리포를 복제하고 필요에 따라 동기화하고 작업을 위한 분기를 만들고 작업을 커밋하고 가끔 한두 개의 커밋을 되돌리는 정도로는 깃허브 데스크톱이 제 역할을 톡톡히 한다.

자신에게 커밋과 협업 권한이 없는 리포에서 작업하려면 일반적으로 깃허브에서 리포를 분기하고 이를 데스크톱으로 복제한다. 그런 다음 필요한 분기를 깃허브 데스크



화면 2 | 깃허브의 무료 오픈 소스인 아톰 편집기는 깃, 깃허브와 통합된다. 플러그인을 통해 주가 언어 또는 기능을 확장한다.



화면 3 | 깃허브는 커밋 히스토리(화면 위쪽)와 디프 뷰(화면 아래쪽)를 비롯해 코드에 대한 여러 가지 유용한 뷰를 제공한다.

립트, CSS와 Node.js를 통합해 만들어졌다. 웹 기술을 사용해 크로스 플랫폼 앱을 빌드하기 위한 프레임워크인 일렉트론(Electron)(<http://electron.atom.io>)에서도 실행된다(깃허브 데스크톱도 일렉트론에서 실행된다).

깃허브 프로젝트

오픈 소스 소프트웨어 프로젝트에는 핵심 커미터 팀이 외 사람의 기여를 받아들이면서도 품질 관리를 강화해야 할 경우가 많다. 기여자에 대한 수요는 크지만 프로젝트에 새 기여자를 조달하면서 코드 베이스의 무결성을 유지하는 것은 매우 어렵고 위험한 일이다. 물론 프로젝트 사용자의 피드백도 중요하다.

깃허브에는 이처럼 오픈 소스 프로젝트의 운영을 원활하게 하는 여러 가지 메커니즘이 있다. 예를 들어 사용자는 깃허브에서 프로젝트에 이슈를 추가해 버그를 보고하거나 기능을 요청할 수 있다. 이를 다른 시스템에서는 '티켓'이라고 부르기도 한다. 이슈를 다루는 프로젝트 관리자는 작업 목록을 생성하고 이슈를 특정 기여자에게 할당하고 관심을 가진 다른 기여자에게 이러한 내용을 전달해 이들이 변경 사항에 대한 알림을 받도록 하고 라벨과 이정표를 추가할 수 있다.

프로젝트에 기여하려면 먼저 원하는 커밋된 변경 사항이 포함된 토픽 헤드 분기에서 시작하고, 헤드 분기에서 풀 요청을 초기화한다. 그 후 커밋을 푸시하고 프로젝트 분기에 추가한다. 다른 기여자는 제안된 변경 사항을 리뷰하고 리뷰 의견을 추가하고 풀 요청 토론에 참여하고 자기 자신의 커밋을 풀 요청에 추가할 수 있다. 관여한 모든 사람이 제안된 변경 사항을 승인하면 컴퓨터가 풀 요청을 병합한다. 병합은 모든 커밋을 보존하고 모든 변경을 하나의 커밋에 밀어 넣거나, 헤드 분기에서 기본 분기로 커밋을 재지정한다. 병합으로 인해 충돌이 발생할 경우 깃허브

에서 또는 명령줄을 사용해 해결한다.

깃허브의 코드 리뷰를 사용하면 분산된 팀이 비동기적으로 협업할 수 있다. 리뷰어에게 유용한 깃허브 툴로는 디프, 히스토리, 블레임 뷰(blame view, 커밋별로 파일의 진행 과정을 볼 수 있음)(<https://github.com/blog/2304-navigate-file-history-faster-with-improved-blame-view>) 등이 있다. 깃허브의 코드 토론은 코드 변경과 함께 제공되는 의견을 통해 진행된다. 기본 툴로 부족하다면 깃허브 마켓플레이스에서 코드 리뷰와 지속적 통합 관리를 툴을 추가하면 된다.

마켓플레이스 애드온은 오픈 소스 프로젝트용으로 무료로 제공되는 경우가 많다.

깃허브 지스트

지스트(gist)(<https://gist.github.com>)는 작업(공개)을 공유하거나 나중에 재사용하기 위해 작업을 저장(비밀)하기 위한 특수한 깃허브 리포지토리다. 하나의 파일, 파일의 일부분 또는 전체 애플리케이션을 저장할 수 있으며, 지스트를 다운로드, 복제, 분기, 내장할 수도 있다.

공개된 지스트는 검색으로 찾으면(<https://gist.github.com/discover>) 된다. 키워드를 사용해 찾는 범위를 좁힐 수 있다. 예를 들어 접두어를 사용해 특정 사용자의 지스트, 별 수가 최소 N개인 지스트, 특정 파일 이름을 가진 지스트 등으로 검색 결과를 제한하면 된다. 비밀 지스트는 검색되지 않지만 URL을 알면 누구나 볼 수 있다. 따라서 코드를 정말 보호하고 싶다면 비공개 리포지토리를 사용해야 한다.

깃허브는 깃 리포지토리를 코드 리뷰, 프로젝트 관리, 다른 개발자 툴, 팀 관리, 소셜 코딩과의 통합을 위한 기능과 함께 서비스로 제공한다. 깃허브는 이런 기능을 하는 유일한 제품은 아니지만 오픈 소스 소프트웨어 개발 분야에서 독보적인 위상을 가진 리포지토리인 것은 분명하다. [ITWORLD](#)



테크놀로지 및 비즈니스 의사 결정을 위한 최적의 미디어 파트너



기업 IT 책임자를 위한 글로벌 IT 트렌드와 깊이 있는 정보

ITWorld의 주 독자층인 기업 IT 책임자들이 원하는 정보는 보다 효과적으로 IT 환경을 구축하고 IT 서비스를 제공하여 기업의 비즈니스 경쟁력을 높일 수 있는 실질적인 정보입니다.

ITWorld는 단편적인 뉴스를 전달하는 데 그치지 않고 업계 전문가들의 분석과 실제 사용자들의 평가를 기반으로 한 깊이 있는 정보를 전달하는 데 주력하고 있습니다. 이를 위해 다양한 설문조사와 사례 분석을 진행하고 있으며, 실무에 활용할 수 있고 자료로서의 가치가 있는 내용과 형식을 지향하고 있습니다.

특히 IDG의 글로벌 네트워크를 통해 확보된 방대한 정보와 전세계 IT 리더들의 경험 및 의견을 통해 글로벌 IT의 표준 패러다임을 제시하고자 합니다.

‘가입부터 커밋까지’ 깃허브 기본 활용법

Tech
Guide

Martin Heller | InfoWorld

오픈 소스에 기여하든 하지 않든, 그리고 다른 곳에 리포지토리(리포)가 있든 없든 모든 개발자는 깃허브를 ‘반드시’ 사용해야 한다. 농담이 아니다. 요즘 시대에는 거의 모든 개발자가 오픈 소스 프로젝트 사용한다. 또한 사용하는 모든 오픈 소스 프로젝트의 코드를 읽고 프로젝트의 문제와 변경점을 추적하고, 문제를 발견하면 해결할 수 있도록 알려야 한다. 커뮤니티를 통해 코드 수정, 개선된 문서화 또는 코드 향상에 기여할 수 있다면 더 좋다.

따라서 오픈 소스 리포지토리 대부분을 호스팅하는 깃허브닷컴(GitHub.com)(<https://github.com>)은 개발자라면 그 사용법을 알아 둘 필요가 있다. 지금부터 기본적인 깃허브 사용법을 살펴보자. 이는 개발자, 테크니컬 라이터 또는 테스터를 불문하고 기본적으로 같다.

깃허브 가입하기

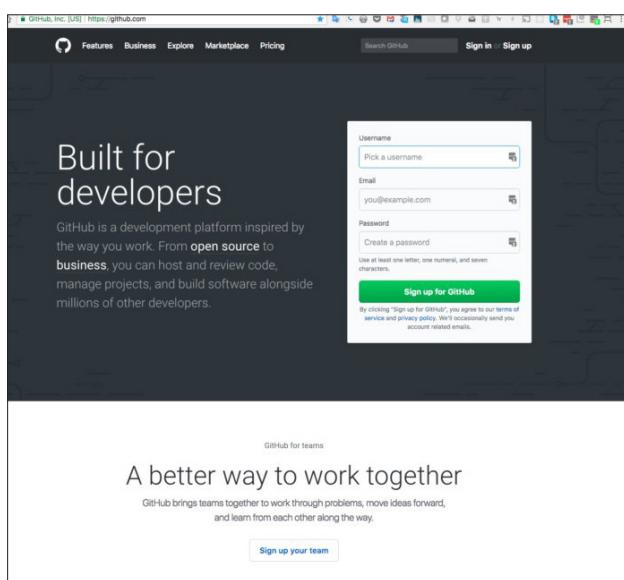
아직 깃허브 사용자가 아니라면 계정을 만들어야 한다. <화면 1>처럼 로그인하지 않은 경우 표시되는 깃허브닷컴의 양식을 통해 사용자 이름과 암호를 만들어 이메일 주소와 연결한다. 무료 계정에서는 다른 사람의 프로젝트에 참여해 작업하고 자신의 공개 리포를 만들 수 있

다. 확인된 학생, 교사 또는 학계 연구원을 제외하고 비공개 리포를 만들려면 유료 계정으로 업그레이드해야 한다. 이번 글에서 다루는 내용을 익히는 데는 무료 계정이면 충분하다. “Hello, World!” 코드를 다른 사람이 볼 수 없도록 숨길 필요는 없기 때문이다.

최종적으로 가입을 완료하려면 몇 가지 부가적인 단계를 더 거쳐야 한다. 먼저 이메일 주소 확인은 필수다. 그리고 급하지 않다면 프로필 사진을 추가하고 이중 인증으로 계정을 보호하는 것이 좋다. 약력도 적당히 채워 넣으면 된다.

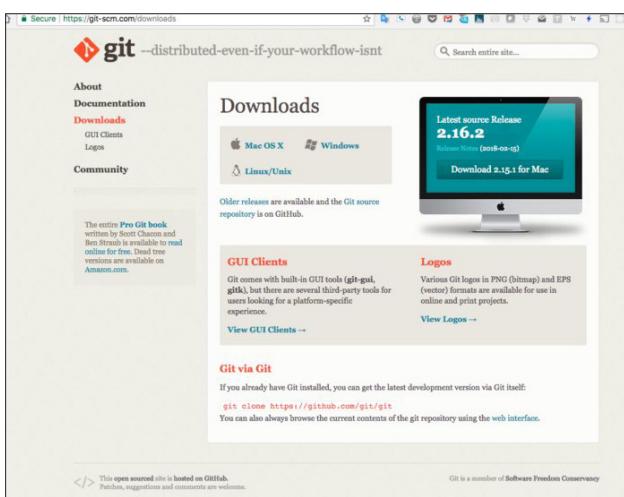
깃 설치하기

윈도우 또는 맥OS를 사용할 때 데스크톱 깃 리포를 깃허브 리포와 간편하게 동기화하는 방법은 깃허브 데스크톱(<https://>

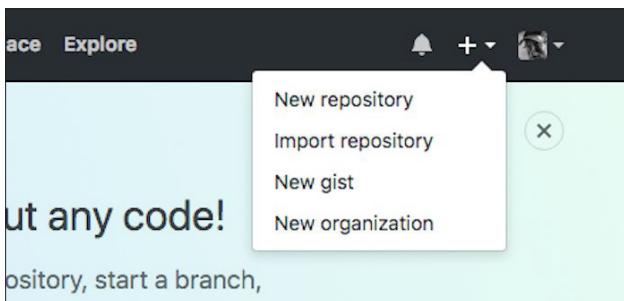


화면 1 | 깃허브닷컴의 로그인

desktop.github.com) GUI를 설치하는 것이다. 거의 모든 운영체제용으로 제공되는 깃 명령줄을 사용하거나 깃 GUI를 사용하는 방법도 있다. 이미 깃이 설치돼 있는지 확인하려면 명령 셸에서 다음을 입력하고 실행하면 된다.



화면 2 | 깃 설치 프로그램 다운로드



화면 3 | 새 리포지토리 만들기

화면 4 | 리포지토리 만들기 설정

```
$ git --version
git version 2.8.1
```

버전 번호 대신 오류 메시지가 뜨거나 표시되는 버전이 너무 오래된 경우 <화면 2>처럼 깃 웹사이트(<https://git-scm.com/downloads>)에서 해당 시스템용 최신 깃 프로그램을 다운로드해 설치하면 된다. 참고로 필자의 맥OS 시스템에는 깃 버전 2.15.1이 설치돼 있어서 역시 최신 버전을 다운로드했다.

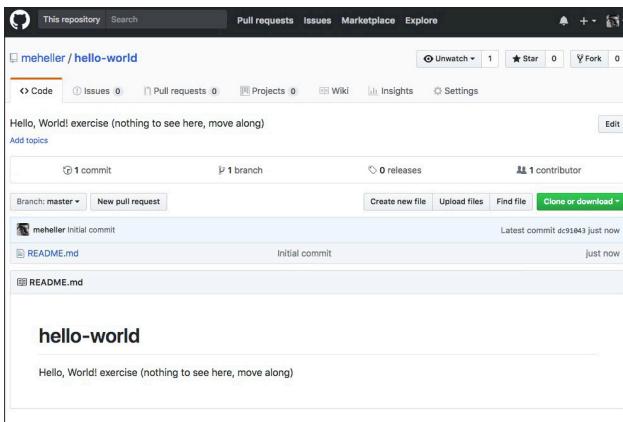
윈도우 10 시스템에서 리눅스용 윈도우 서브시스템(<https://docs.microsoft.com/en-us/windows/wsl/about>)을 설치해 사용 중이라면 리눅스 배시(Bash) 셸에 깃을 설치해야 할 수도 있다. 필자는 우분투 배시 셸이 설치된 윈도우 10 시스템도 사용하고 있는데, 여기에 깃을 설치하려면 APT를 업데이트한 다음(<sudo apt-get update>) <sudo apt-get install git>을 실행해서 깃을 설치해야 했다.

한편 깃허브 데스크톱 또는 다른 깃 GUI가 설치돼 있더라도 문제를 수정하거나 스크립트의 일부로 명령줄에서 깃을 사용해야 할 때가 종종 있다. 따라서 리포에서 어려운 상황에 직면할 때까지 기다리는 대신, 지금 바로 깃을 설치하고 정상 작동하는지 확인하는 것이 좋다.

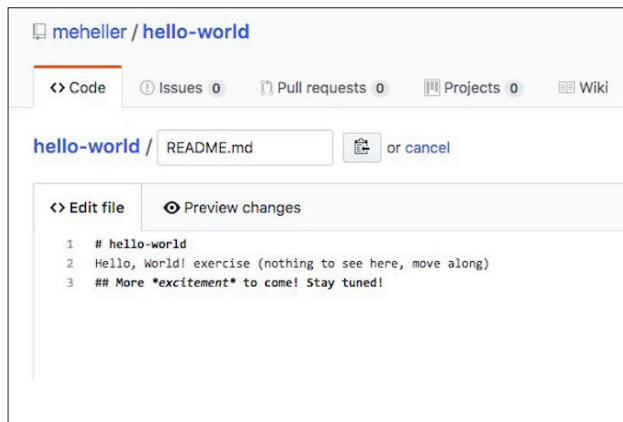
깃허브에서 새 리포지토리 만들기

자, 이제 깃허브 “Hello, World!” 자습서(<https://guides.github.com/activities/hello-world/#repository>) 내용을 따라가 보자. 이 자습서는 가장 먼저 깃허브 웹 인터페이스를 사용해서 리포지토리를 만드는 방법을 보여준다. 깃허브 페이지 오른쪽 위, 프로필 사진 바로 옆에 드롭다운이 있다. <화면 3>처럼 클릭한 후 “새 리포지토리(New repository)”를 선택한다.

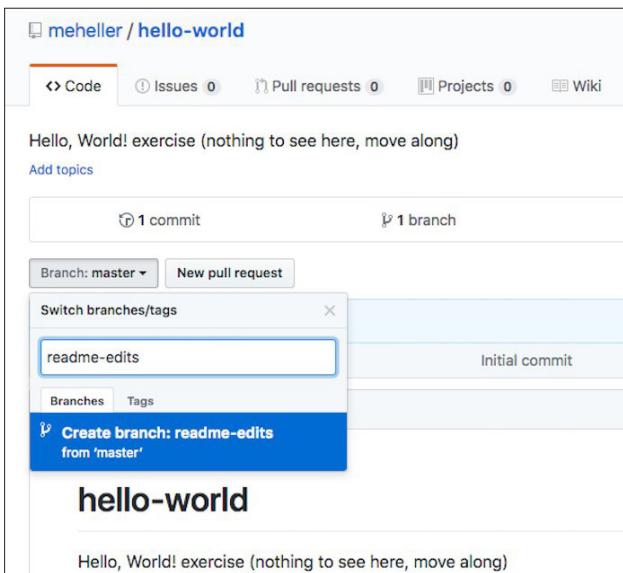
다음 양식이 열리면 <화면 4>처럼 내용을 선택하면 된다. 녹색 버튼인 “리포지토리 만들기(Create repository)”를 클릭하면 <화면 5>처럼 분기 하나, 커밋 하나, 관찰자(watcher) 하나, 기여자 하나인 리포가 나타난다.



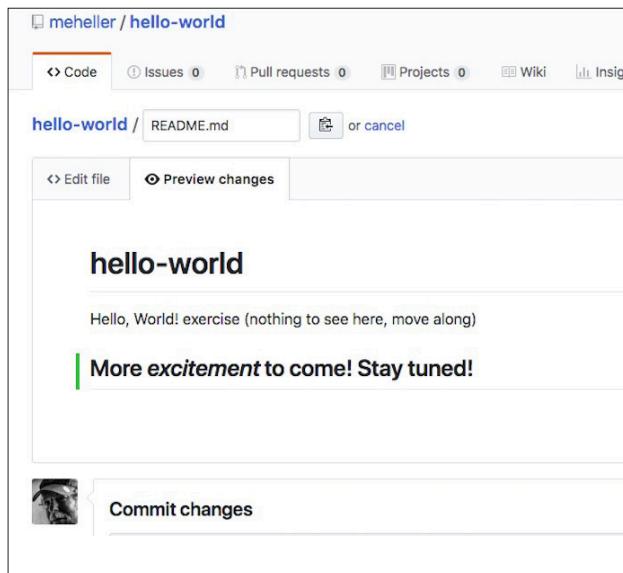
화면 5 | 리포지토리 만들기 최종 결과



화면 7 | README 텍스트 파일 변경하기



화면 6 | 새 분기 만들기



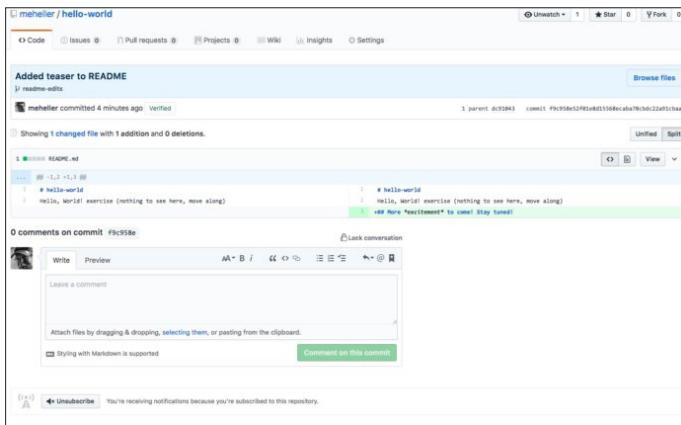
화면 8 | 변경 사항 내용 확인하기

깃허브에서 새 분기 만들기

리포에서 단순히 파일 편집을 시작하는 대신 새 분기(<https://guides.github.com/activities/hello-world/#branch>)를 만들어 보자. 새 분기는 기본 분기(이 글에서 다루는 사례에서는 master)의 그 순간 스냅샷이다. 두 분기를 병합할 때까지는 기본 분기에 영향을 미치지 않으면서 새 분기에서 변경 작업을 할 수 있다. <화면 6>을 보면 분기 드롭다운 메뉴에서 새 분기 이름으로 'readme-edits'를 입력했다. 여기서 파란색 버튼 "분기 만들기(Create branc)"를 누르면 master 분기가 새 **readme-edits** 분기로 복사되고 작업 대상이 새 분기로 전환된다.

깃허브 리포지토리에 변경 사항 커밋하기

이제 **master** 분기에 아무 문제도 일으키지 않으면서 새 분기의 변경 사항을 자유롭게 커밋(<https://guides.github.com/activities/hello-world/#commit>)할 수 있다. <화면 7>처럼 README.md의 왼쪽 파란색 링크를 클릭하고 파일 프레임 오른쪽 위의 연필 아이콘을 클릭해서



화면 9 | 변경 사항 커밋하기

README 텍스트를 편집한다. 깃허브의 **md** 파일은 특수 한 버전의 마크다운(Markdown) 언어인 깃허브 플레이어드 마크다운(GitHub Flavored Markdown)을 지원한다.

깃허브 자습서를 따라가면 프로필을 작성하도록 계속 권유한다. 여기서 <화면 8>처럼 깃허브의 마스터링 마크다운(Mastering Markdown)(<https://guides.github.com/features/mastering-markdown>) 페이지, 아담 프리차트의 마크다운 치트시트(Markdown Cheatsheet) (<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>), 그리고 두 페이지의 링크를 사

용해서 마크다운으로 이것저것 실험해 봐도 좋다. 변경 사항을 커밋하기 전에 “변경 사항 미리 보기(Preview changes)” 탭에서 마지막으로 확인할 수 있다.

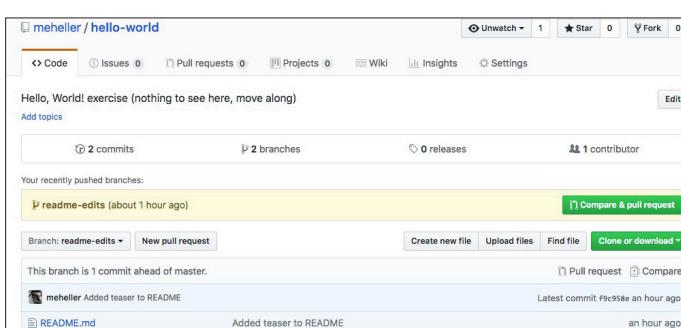
이제 변경 사항을 커밋하면 된다. 커밋 주석에 작업 내용을 잘 설명해야 한다. 필자는 커밋 주석에 <화면 9>처럼 “Added teaser to README”라고 입력했다. 이제 “직접 커밋(Commit directly)...” 선택 항목은 그대로 두고, 녹색 “변경 사항 커밋(Commit changes)” 버튼을 클릭한다. 최상위 리포지토리로 돌아가면(파란색 **hello-world** 링크 클릭) 커밋이 표시될 것이다. **master** 분기와 **readme-edits** 분기 사이를 오가면서 README 변경 사항을 보고 시각적으로 비교할 수 있다. **readme-edits** 분기에서 커밋 주석을 클릭하면 소스 코드 뷰가 나란히 열린다. 필요하면 여기서 페이지 아래의 커밋에 주석을 추가하면 된다. 코드 리뷰를 문서화하기 위한 효과적인 메커니즘이다.

깃허브 리포의 풀 요청 열기

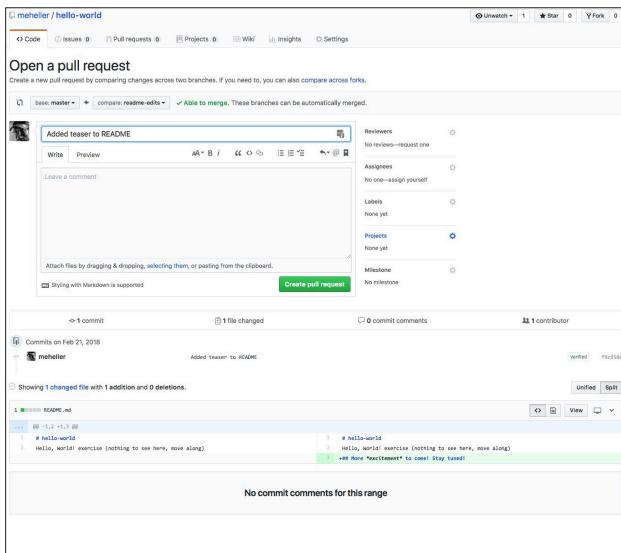
이제 master에서 떨어진 분기에 변경 사항이 있으니 풀 요청을 열어야 한다(<https://guides.github.com/activities/hello-world/#pr>). 풀 요청을 열면 변경 사항을 제안하고 다른 누군가에게 자신의 기여를 검토하고 가져가서 분기로 병합할 것을 요청하게 된다. 물론 이 글에서 작성한 내용에 비하면 풀 요청이 과해 보일 수 있다. 그러나 여기서는 다른 사람의 리포로 보내는 방법을 익히는 것이 주목적이므로 풀 요청을 진행해 보자. 이처럼 풀 요청(PR)을 제출하는 것은 커미터가 소수인 오픈 소스 프로젝트에 기여할 수 있는 대표적인 방법이다.

먼저 주 리포 페이지에서 최근 푸시된 분기를 보여주는 유용한 메모를 볼 수 있다. 여기서

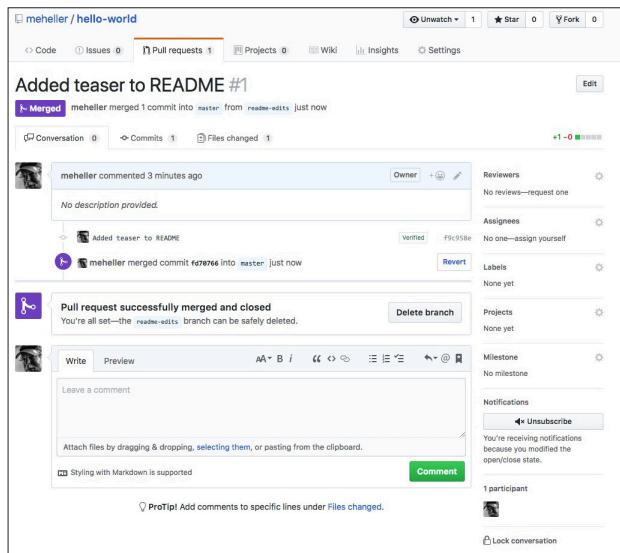
푸시는 분기가 깃허브 리포에 추가됐음을 의미한다. 로컬에서 먼저 변경 작업을 수행할 경우 로컬 리포에 커밋하고 깃허브 같은 업스트림 리포에 별도로 푸시해야 한다. 이제 <화면 10>처럼 회색 ‘새 풀 요청(New pull request)’ 버튼을 눌러 소스와 대상을 선택한다. 과정을 모두 따라갈 필요는 없고 눌러서 UI를 미리 보는 정도로 지나가면 된다.



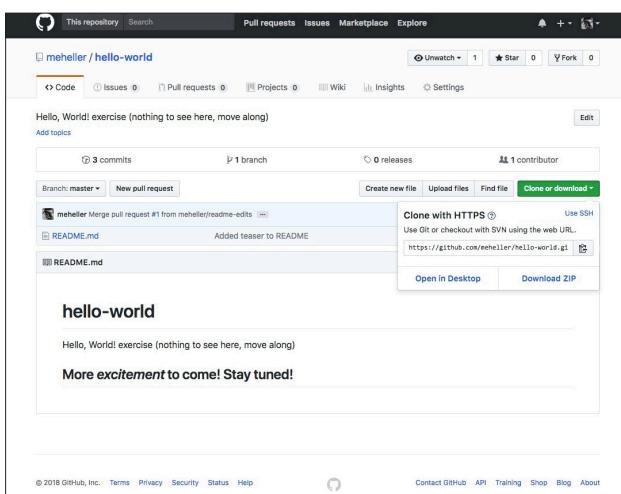
화면 10 | 새 풀 요청하기



화면 11 | 병합을 위해 풀 요청 만들기



화면 12 | 병합 후 분기 삭제하기



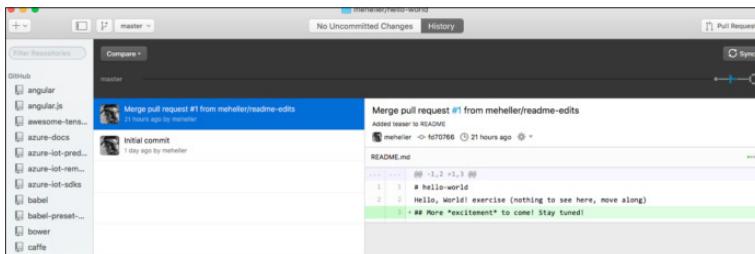
화면 13 | 깃허브 리포 복제하기

는 항상 만들고 삭제할 수 있다. 실제 프로젝트에서 작업할 경우 기능에 대한 작업을 시작할 때마다 분기를 추가하고 프로덕션 분기로 성공적으로 병합한 후 작업 분기를 삭제하는 것이 좋다.

깃허브 리포를 로컬 시스템에 복제하기

실제 환경에서 대부분은 코드와 문서를 온라인에 직접 쓰지 않고 자신의 컴퓨터에서 작성한다. 깃허브 워크플로우에서 이를 어떻게 하는지 알아보기 위해 **hello-world** 리포를 복제해 보자. 깃허브 페이지에서 **hello-world** 리포의 코드 페이지로 이동한 다음 <화면 13>처럼 녹색 “복제 또는 다운로드(Clone or download)” 버튼을 클릭한다. 여기서는 Zip 파일을 다운로드하지 않고 깃 또는 깃허브 데스크톱으로 리포를 복제한다.

깃허브 데스크톱을 설치했다면 “데스크톱에서 열기(Open in Desktop)”를 클릭하고, 설치하지 않았다면 리포 URL 오른쪽의 복사 아이콘을 클릭하고, 콘솔에서 command **git clone** 다음



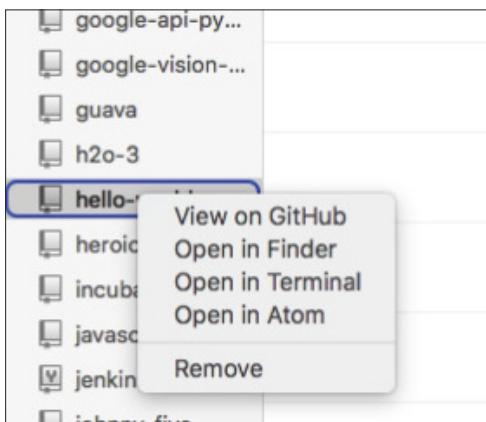
화면 14 | 복제된 깃허브 리포

```
martinheller — bash — 97x24
Last login: Tue Feb 20 16:43:56 on ttys000
Martins-Retina-MacBook:~ martinhellers$ git clone https://github.com/meheller/hello-world.git
Cloning into 'hello-world'...
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
Martins-Retina-MacBook:~ martinhellers$
```

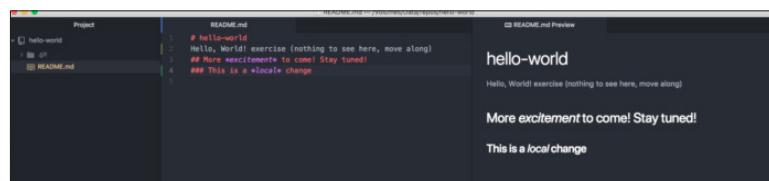
화면 15 | 복제 작업의 명령줄 버전

우스 오른쪽 버튼으로 클릭하고 원하는 옵션을 선택한다. 필자 시스템의 컨텍스트 메뉴는 <화면 16>과 같다.

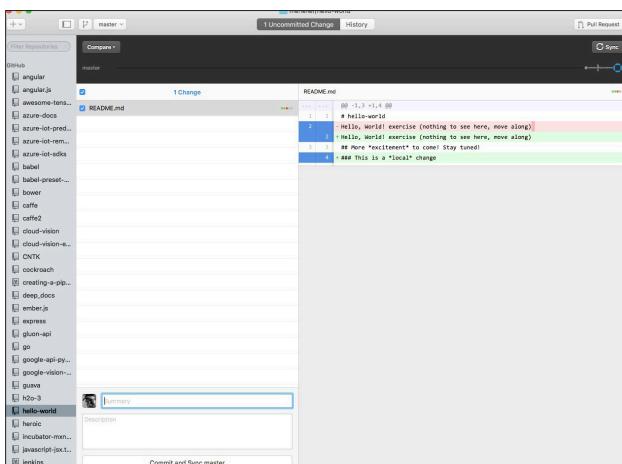
<화면 17>은 깃허브의 무료 편집기인 아톰(Atom)(<https://atom.io>)에서 리포를 연 다음 README.md에 라인을 추가하고 미리보기를 한 것이다. 파일을 저장하면 <화면 18>처럼 리포에 커밋되지 않은 변경 사항이 하나 있다는 메시지가 깃허브 데스크톱에 표시된다. 이제 왼쪽 아래의 커밋 요약을 입력하고 “커밋 및 마스터 동기화(Commit and Sync master)”를 누르면 변경 사항이 로컬로 커밋되고 깃허브로 푸시된 것을 확인할 수 있다. 브라우저를 새로 고친 후의 깃허브 데스크톱 사용자라면 [hello-world](#) 리포를 마



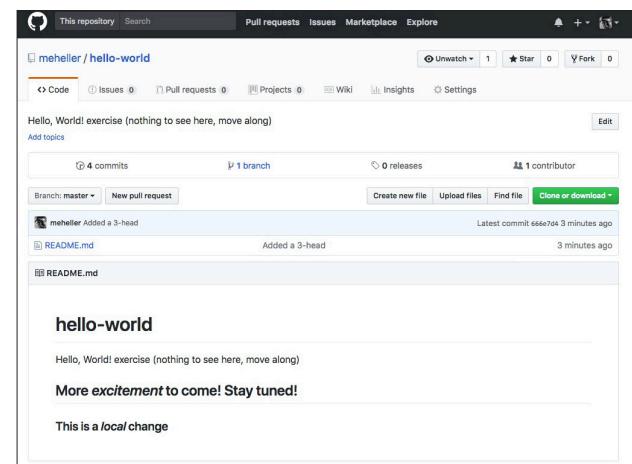
화면 16 | 리포의 컨텍스트 메뉴



화면 17 | 아톰에서 미리보기하기



화면 18 | 커밋되지 않은 변경사항 알림



화면 19 | 로컬 커밋과 깃허브 푸시 완료

에 URL을 붙여넣는다. 둘 중 어떤 방법으로 하든 컴퓨터에 깃 리포가 만들어진다. 복제가 완료되면 깃허브 데스크톱은 <화면 14>와 같은 상태가 된다. 복제 작업의 명령줄 버전은 <화면 15>와 같다.

깃허브 리포지토리에 커밋 푸시

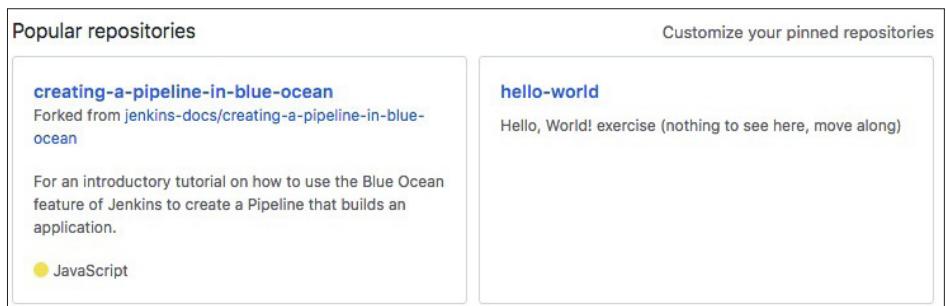
이제 로컬 리포가 생겼으니 로컬에서 편집하고 커밋한 다음 깃허브에 커밋을 푸시하자. 명령줄을 이용하면 **cd**를 사용해 [hello-world](#)로 들어가서 선호하는 텍스트 편집기로(마크다운을 지원하는 편집기가 좋다) README.md를 편집한다. 깃허브 데스크톱 사용자라면 [hello-world](#) 리포를 마

브 리포를 보면 <화면 19>와 같다.

그렇다면 이런 변경 작업을 분기에서 할 수 있을까? 물론이다. 깃허브 데스크톱의 왼쪽 위 분기 드롭다운 바로 왼쪽 아이콘, 즉 "분기 추가(Add a branch)" 버튼을 이용하면 된다.

깃허브 리포 분기

분기된 리포에서도 변경 작업이 가능하다. 깃허브 리포 페이지에서 상단의 분기(Fork) 버튼을 클릭해서 손쉽게 리포 스냅샷을 찍을 수 있다. 이미 리포에 대한 커밋 권한을 갖고 있다면 별 의미가 없지만 다른 사람의 오픈 소스 프로젝트에 기여하고자 한다면 상당히 쓸모가 있다. 예를 들어 필자는 젠킨스 블루 오션(Jenkins Blue Ocean)을 공부하고 있었는데, 그중 한 단계가 [Jenkins-docs](#)에서 [creating-a-pipeline-in-blue-ocean](#) 리포를 복제하는 것이었다. 이 단계를 통해 <화면 20>처럼 편집할 수 있고, 변경 사항(블루 옵션에 의해 생성된)을 커밋할 수 있는 리포 복사본을 생성했다.



화면 20 | 커밋할 수 있는 리포 복사본

분기된 리포를 확보하면 이 리포를 자신의 것처럼 다룰 수 있을 뿐만 아니라 분기를 생성한 업스트림 리포에서 나중에 커밋을 적용하고 풀 요청을 업스트림 리포에 제출할 수도 있다. 오픈 소스 프로젝트에 기여할 때 일반적으로 따르게 되는 워크플로우다. 이로써 우리는 분산 개발의 세계에 들어왔다. 분산 오픈 소스 개발에 입문한 것을 환영한다! [@WORLD](#)

깃허브 초보 탈출 깃 버전 관리의 이해

Tech
Guide

William Rothwell | InfoWorld

깃과 버전 제어 개념을 이해하는 것은 깃허브의 구조를 파악하고 더 효과적으로 이용하는 데도 매우 중요하다. 이를 위해 가장 좋은 방법은 역사적인 관점에서 버전 제어를 살펴보는 것이다.

버전 제어 소프트웨어의 변천사

버전 제어 소프트웨어는 지금까지 3세대를 거쳤다. 1세대는 극히 단순했다. 여러 개발자가 같은 실제 시스템에서 작업한 다음 한 번에 파일 하나를 "체크아웃"하는 방식이었다. 이 세대의 버전 제어 소프트웨어는 파일 잠금이라는 기술을 사용했다. <그림 1>처럼 개발자가 파일을 체크아웃하면 파일이 잠겨서 다른 개발자는 편집할 수 없다.

1세대 버전 제어 소프트웨어로는 리비전 컨트롤 시스템(Revision Control System, RCS)과 소스 코드 컨트롤 시스템(Source Code Control Systems, SCCS)이 있다. 그러나 1세대에는 다음과 같은 문제점이 있었다.

- 한 번에 개발자 1명만 파일 작업을 할 수 있고 이로 인해 개발 프로세스에서 병목 현상이 발생한다.

그림 1 | 1세대 버전 제어 소프트웨어의 구조

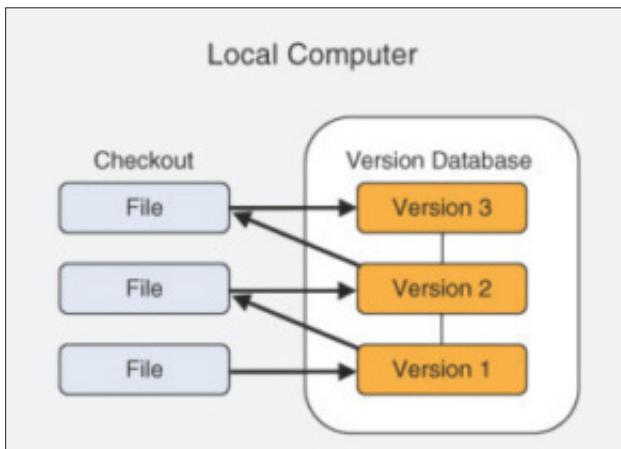
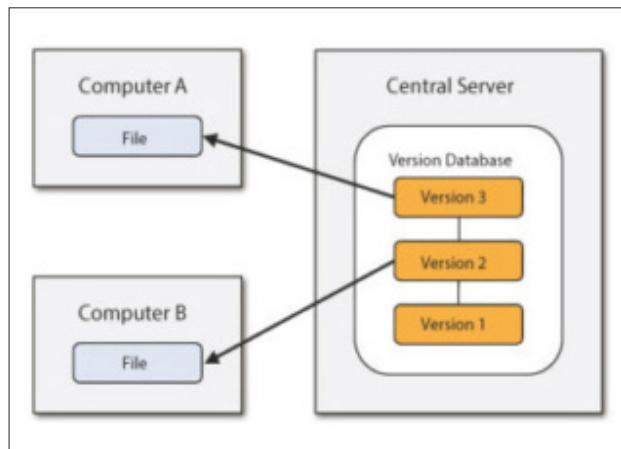


그림 2 | 2세대 버전 제어 소프트웨어



- 버전 제어 소프트웨어가 포함된 시스템에 개발자가 직접 로그인해야 한다.

이러한 문제는 2세대 버전 제어 소프트웨어에서 해결됐다.<그림 2>와 같은 2세대에서는 중앙의 리포지토리 서버에 파일이 저장된다. 개발자는 파일의 개별 복사본을 체크아웃할 수 있다. 개발자가 파일 작업을 마치면 파일이 리포지토리에 체크인된다.

이 경우 두 명의 개발자가 파일의 같은 버전을 체크아웃할 때 문제가 발생할 가능성이 있는데, 2세대에서는 병합이라는 프로세스를 통해 이를 해결한다. 그렇다면 병합이란 무엇일까? 예를 들어 A와 B, 개발자 2명이 **abc.txt**라는 파일의 버전 5를 체크아웃한다고 가정하자. A는 작업을 마친 후 파일을 다시 체크인한다. 일반적으로 이 경우 파일의 새로운 버전 6이 만들어진다. 나중에 B가 자신의 파일을 체크인한다. B가 체크인한 새 파일은 B의 변경 사항과 A의 변경 사항을 포함해야 한다. 이 과정은 병합 프로세스를 통해 실행된다.

병합을 처리하는 방법은 사용하는 버전 제어 소프트웨어에 따라 다르다. 예를 들어 A와 B가 전혀 다른 파일 부분을 작업한 경우 병합 프로세스는 매우 간단하다. 그러나 두 사람이 같은 코드 라인을 편집했다면 병합 프로세스가 복잡해진다. 이 경우 B가 열쇠를 주고 있다. A의 코드와 자신의 코드 중 파일의 새 버전이 될 코드를 결정한다. 병합 프로세스가 완료되면 파일을 리포지토리에 커밋하는 프로세스가 진행된다. 파일 커밋은 기본적으로 리포지토리에 새 버전을 만드는 것을 의미한다. 이 예에서는 파일의 버전 7이 만들어진다. 2세대 버전 제어 소프트웨어로는 컨커런트 버전 시스템(Concurrent Version System, CVS)과 서브버전(Subversion)이 있다.

3세대는 분산 버전 제어 시스템(Distributed Version Control Systems, DVCS)으로 불린다. 2세대와 마찬가지로 중앙 리포지토리 서버에 프로젝트의 모든 파일이 들어간다. 다만 개발자가 리포지토리에서 개별 파일을 체크아웃하는 방식이 아니라 전체 프로젝트를 체크아웃한다. 이로써 개발자는 개별 파일이 아닌 전체 파일 집합으로 작업을 하게 된다. 3세대 버전 제어의 구조는 <그림 3>에서 확인할 수 있다.

그림 3 | 3세대 버전 제어 소프트웨어

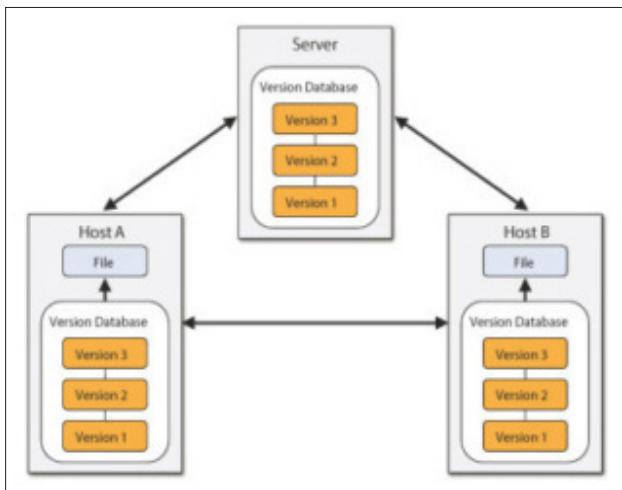
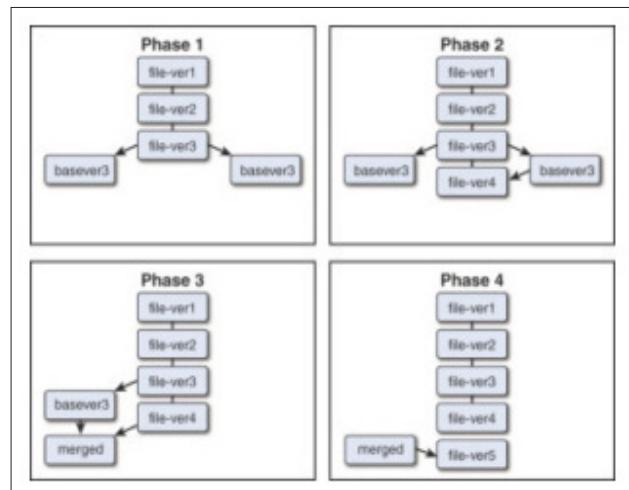


그림 4 | 2세대 병합과 커밋



2세대와 3세대 버전 제어 소프트웨어의 큰 차이점은 병합과 커밋 프로세스의 작동 방식이다. 2세대는 병합을 수행한 다음 리포지토리에 새 버전을 커밋하는 순서로 진행된다. 반면 3세대 버전 제어 소프트웨어에서는 파일이 체크인된 다음 병합된다. 두 기술의 차이점을 이해하기 위해서는 <그림 4>를 보자.

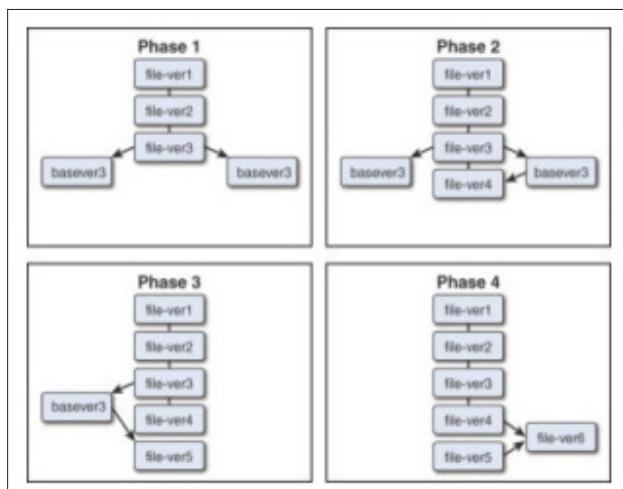
1단계에서 두 명의 개발자가 3번째 버전을 바탕으로 하는 파일을 체크아웃한다. 2단계에서 한 개발자가 파일을 체크인하면서 파일 버전은 4가 된다. 3단계에서 두 번째 개발자는 먼저 체크아웃한 사본을 버전 4 및 다른 버전의 변경점과 먼저 병합해야 한다. 병합이 완료된 후 새 버전은 버전 5로 리포지토리에 커밋한다. 여기서 주목해야 할 것은 리포지토리에 무엇이 있는지다. 버전 1부터 5까지 개발이 매우 직선적으로 진행됨을 알 수 있다. 이 단순한 소프트웨어 개발 접근 방법은 몇 가지 문제점을 갖고 있다.

무엇보다 커밋하기 전에 병합해야 하므로 개발자가 정기적인 변경 사항 커밋을 꺼리는 경우가 많다. 병합 프로세스는 골치가 아프기 때문에 정기적으로 병합을 하지 않고 나중까지 기다렸다가 한 번만 병합하는 것이다. 결과적으로 갑자기 큰 코드 덩어리가 파일에 추가되므로 소프트웨어 개발에 부정적인 영향을 미치게 된다. 문서를 작성할 때 정기적으로 그 문서를 저장하는 것이 좋듯, 개발자도 리포지토리에 변경 사항을 자주 커밋하는 것이 바람직하다.

더 중요한 것은 이 사례에서 버전 5가 개발자가 원래 완료한 작업이 아닐 가능성성이 있다는 점이다. 즉, 개발자가 병합 프로세스를 완료하기 위해 자신의 작업 일부를 포기할 수 있다. 자칫 더 좋은 코드가 손실될 수 있다는 점에서 이상적인 프로세스라고 할 수 없다.

반면 3세대인 <그림 5>를 보자. 이른바 방향성 비순환 그래프(Directed Acyclic Graph, DAG)다. 2세대보다 더 복잡하다는 지적이 있지만 더 좋은 방법임은 분명하다. 1단계와 2단계는 <그림 4>와 같다. 그러나 3단계에서 두 번째 체크인 프로세스의 결과가 버전 4가 아니라, 버전 4와 별도인 버전 5임을 주목해야 한다. 결국 프로세스의 4단계에서 파일 버전 4와 5가 병합돼 버전 6이 생성됐다. 이 프로세스는 더 복잡하지만(개발자 수가 늘어나면 훨씬 더 복잡해진다) 일렬로 진행되는 개발보다 다음과 같은 몇 가지 이점을 제공한다.

그림 5 | 3세대 커밋과 병합



- 개발자가 정기적으로 변경점을 커밋할 수 있으며 나중까지 병합에 대해 걱정할 필요가 없다.
- 전체 프로젝트 또는 코드를 다른 개발자에 비해 잘 파악하고 있는 특정 개발자에게 병합 프로세스를 위임할 수 있다.
- 프로젝트 관리자는 언제든 뒤로 돌아가서 개별 개발자가 한 작업을 정확히 볼 수 있다.

물론 2세대와 3세대 두 가지 방법의 장단점을 놓고 여전히 논쟁이 있는 것이 사실이다. 다만 이 글에서는 깃에 초점을 두고 진행한다. 깃은 3세대 버전 제어 시스템의 방향성 비순환 그래프 방법을 사용한다.

깃 설치하기

깃은 여러 시스템에 기본적으로 설치된 경우가 많으므로 일단 다음과 같이 깃 설치 여부를 확인하자.

```
ocs@ubuntu:~$ which git
/usr/bin/git
```

깃이 설치돼 있다면 **git** 명령 경로가 나타난다. 설치돼 있지 않으면 출력이 없거나 다음과 같은 오류가 표시된다.

```
[ocs@centos ~]# which git
/usr/bin/which: no git in (/usr/lib64/qt-3.3/bin:/usr/local/bin:/usr/
local/sbin:/usr/
bin:/usr/sbin:/bin:/sbin:/root/bin)
```

데비안(Debian) 기반 시스템 관리자는 **dpkg** 명령을 사용해서 깃 패키지가 설치되어 있는지를 확인한다.

```
root@ubuntu:~# dpkg -l git
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-aWait/
Trig-pend
||/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version       Architecture Description
+++=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
=====
ii   git          1:1.9.1-1ubun  amd64        fast, scalable, distributed
revision con
```

레드햇 기반 시스템 관리자는 **rpm** 명령으로 깃 패키지가 설치돼 있는지 확인한다.

```
[root@centos ~]# rpm -q git
git-1.8.3.1-6.el7_2.1.x86_64
```

깃이 시스템에 설치되어 있지 않다면 루트 사용자로 로그인하거나 **sudo** 또는 **su**를 사용해서 소프트웨어를 설치해야 한다. 데비안 기반 시스템에 루트 사용자로 로그인하는 경우 다음 명령을 사용해서 깃을 설치한다.

```
apt-get install git
```

레드햇 기반 시스템에 루트 사용자로 로그인했다면 다음 명령으로 깃을 설치할 수 있다.

```
yum install git
```

깃 개념과 기능

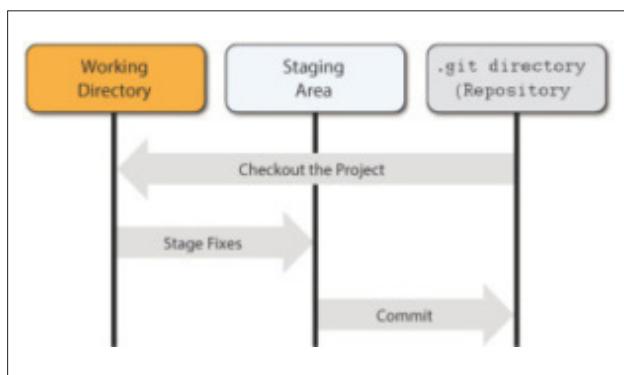
깃을 사용하는 데 있어 한 가지 과제는 기반 개념을 이해하는 것이다. 개념을 이해하지 못하면 모든 명령이 그냥 문자로만 보일 뿐이다. 여기서는 중요한 깃 개념에 초점을 두고, 기본적인 명령 몇 가지를 살펴보자.

깃 스테이지

전체 프로젝트를 체크아웃한다는 점과 작업 대부분이 시스템 로컬에서 진행된다는 점을 기억하는 것이 중요하다. 체크아웃하는 파일은 홈 디렉토리 아래에 위치한다. 깃 리포지토리에서 프로젝트 사본을 받으려면 복제(clone) 프로세스를 사용한다. 복제는 단순히 리포지토리에 있는 모든 파일의 복사본을 만드는 것이 아니다. 다음과 같은 3가지 작업으로 구성된다.

- 홈 디렉토리의 `project_name/.git` 디렉토리 아래에 프로젝트의 로컬 리포지토리를 생성한다. 이 위치의 프로젝트 파일은 중앙 리포지토리에서 체크아웃된 것으로 간주한다.
- 파일을 직접 볼 수 있는 위치에 디렉토리를 만든다. 이 디렉토리를 작업 영역이라고 한다. 작업 영역에서 실행한 변경이라고 해도 즉각 버전 제어가 되지는 않는다.
- 스테이징 영역을 만든다. 스테이징 영역은 파일 변경점을 로컬 리포지토리에 커밋하기 전에 저장하는 역할을 한다.

그림 6 | 깃 스테이지



예를 들어 `Jacumba`라는 프로젝트를 복제한다면 전체 프로젝트가 홈 디렉토리 아래의 `Jacumba/.git` 디렉토리에 저장된다. 이 디렉토리를 수정하면 안 된다. 대신 `~/jacumba` 디렉토리에서 프로젝트 파일을 보면 된다. 변경해야 할 파일은 여기 있는 파일이다. 파일을 변경하려고 하지만 로컬 리포지토리에 변경점을 커밋하기 전에 몇몇 다른 파일에도 작업해야 하는 경우가 있다. 이때는 작업을 마친 파일을 스테이징할 수 있다. 그러면 로컬 리포지토리에 커밋할 준비가 된다.

필요한 부분을 모두 변경하고 모든 파일을 스테이징한 다음에는 로컬 리포지토리에 커밋한다. <그림 6>은 이 전체 과정을 정리한 것이다. 스테이징된 파일을 커밋하면 로컬 리포지토리에 전송된다. 이는 지금까지 수행된 변경점에만 액세스할 수 있음을 의미한다. 중앙 리포지토리에 새 버전을 체크인하는 프로세스를 푸시(push)라고 한다.

깃 리포지토리 호스트 선택

먼저 좋은 소식은 많은 기업이 깃 호스팅을 제공한다는 점이다. 2018년 상반기를 기준으로

20개 이상의 깃 호스팅 서비스가 있다. 따라서 선택의 범위가 넓다. 그러나 이는 나쁜 소식이기도 하다.

나쁜 소식인 이유는 각 호스팅 업체의 장단점을 확인하는 데 시간을 써야 한다는 데 있다. 예를 들어 업체 대부분은 기본 호스팅을 무료로 제공하지만 대규모 프로젝트는 비용을 받는다. 공개 리포지토리만 제공하는 업체도 있고(누구나 리포지토리를 볼 수 있다), 비공개 리포지토리를 만들 수 있는 업체도 있다. 그 외에도 고려해야 할 기능이 많다.

특히 중요하게 생각해야 할 기능이 웹 인터페이스다. 로컬 시스템에서 거의 모든 리포지토리 작업을 할 수 있지만 웹 인터페이스를 통해 일부 작업을 지원하면 실무에서 상당히 유용하다. 서비스를 최종 선택하기에 앞서 지원하는 인터페이스를 잘 살펴봐야 한다. 필자가 추천할 수 있는 호스팅 업체는 깃버킷(<https://bitbucket.org>), 클라우드포지(<http://www.cloud-forge.com>), 코드베이스HG(<http://www.codebasehq.com>), 깃허브(<https://github.com>), 깃랩(<http://gitlab.com>) 등이다. 참고로 이 글에서는 깃허브를 중심으로 설명한다. 특별한 이유는 없고 개인적으로 사용한 호스트였기 때문이다. 앞서 추천한 호스트 중 어느 것을 사용해도 상관없다.

깃 구성하기

이제 이론적인 부분을 모두 살폈으니, 실제 깃으로 작업을 할 차례다. 여기서는 시스템에 **git** 또는 **git-all** 소프트웨어를 설치했고, 깃 호스팅 서비스에 계정을 만들었다고 가정하고 진행한다. 가장 먼저 할 일은 기본적인 설정이다. 커밋 작업을 수행할 때마다 메타데이터에 이름과 이메일 주소가 포함된다. 이 정보를 설정하려면 다음 명령을 실행한다.

```
ocs@ubuntu:~$ git config --global user.name "Bo Rothwell"
ocs@ubuntu:~$ git config --global user.email "bo@onecoursesource.com"
```

물론 여기서 “**Bo Rothwell**”은 독자의 이름으로, “**bo@OneCourseSource.com**”은 독자의 이메일 주소로 바꿔야 한다. 다음 단계는 깃 호스팅 서비스에서 프로젝트를 복제하는 것이다. 복제 전 사용자의 홈 디렉토리에는 다음 파일 하나만 있을 것이다.

```
ocs@ubuntu:~$ ls
first.sh
```

이제 ocs라는 프로젝트를 복제한다.

```
ocs@ubuntu:~$ git clone https://gitlab.com/borothwell/ocs.git
Cloning into 'ocs'...
Username for 'https://gitlab.com': borothwell
Password for 'https://borothwell@gitlab.com':
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```

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

실행이 성공하면 사용자의 홈 디렉토리에 새 디렉토리가 생성된다.

```
ocs@ubuntu:~$ ls
first.sh  ocs
```

새 디렉토리로 전환하면 다음과 같이 리포지토리에서 무엇이 복제되었는지 보인다(현재까지는 하나의 파일만 리포지토리에 있다).

```
ocs@ubuntu:~$ cd ocs
ocs@ubuntu:~/ocs$ ls
README.md
```

이제 리포지토리 디렉토리에 새 파일을 만든다. 새로 만들거나 다른 위치의 파일을 복사한다.

```
ocs@ubuntu:~/ocs$ cp ../first.sh
```

단, 유의해야 할 것은 이 디렉토리는 작업 디렉토리이므로 여기 배치되는 파일은 버전 제어되지 않는다는 점이다. 파일을 로컬 리포지토리에 넣으려면 먼저 스테이징 영역에 추가한 다음 리포지토리에 커밋해야 한다.

```
ocs@ubuntu:~/ocs$ git add first.sh
ocs@ubuntu:~/ocs$ git commit -m "added first.sh"
[master 3b36054] added first.sh
1 file changed, 5 insertions(+)
create mode 100644 first.sh
```

git add 명령을 이용하면 파일을 스테이징 영역에 배치할 수 있다. **git commit** 명령은 스테이징 영역의 모든 파일을 로컬 리포지토리에 커밋한다. **-m** 옵션을 사용해 메시지를 추가할 수 있는데, 이 사례에서는 커밋 이유("added first.sh")를 추가했다.

또 하나 중요한 것은 서버의 리포지토리에는 변경된 점이 없다는 점이다. **git commit** 명령은 로컬 리포지토리만 업데이트한다. 따라서 현재 프로젝트의 웹 기반 인터페이스를 보면, 서버 리포지토리는 수정되지 않았다. 원본 파일인 **README.md**가 며칠 전에 서버로 푸시됐지만 새 파일인 **first.sh**는 항목이 없다. 따라서 다음과 같이 서버 리포지토리에 체크인(푸시)하자.

```
ocs@ubuntu:~/ocs$ git push -u origin master
Username for 'https://gitlab.com': borothwell
Password for 'https://borothwell@gitlab.com':
```

```

Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 370 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://gitlab.com/borothwell/ocs.git
    12424f5..3b36054 master -> master
Branch master set up to track remote branch master from origin.

```

이제 다시 웹 기반 인터페이스를 확인하면 푸시 작업이 성공적으로 실행되었음을 확인할 수 있다. 이 시점에서는 스테이징 영역에 있는 파일의 모든 변경점이 로컬 리포지토리와 중앙 서버 리포지토리에 업데이트된다. 



IT 트렌드 종합 정보센터 IDG Tech Library

IDG Tech Library는 IDG 글로벌 네트워크를 통해 축적된 전문 정보를 재구성하여 최신 기술의 기본 개념부터 현황, 전략 및 도입 가이드까지 다양한 프리미엄 IT 정보를 제공합니다. Computer World, Info World, CIO, Network World 등의 세계적 IT 유명 매체의 심도 깊은 정보를 무료로 만나보세요.

IDG Deep Dive, Tech Focus, Summary, World Update 등의 다양한 콘텐츠를 제공 받을 수 있습니다.



한국IDG(주) 서울시 종로구 종로 1가 108번지 창화빌딩 4층 100-161 Tel : 02-558-6950 Fax : 02-558-6955
www.itworld.co.kr www.twitter.com/ITWorldKR www.facebook.com/ITworld.Korea

세계 최대 코드 저장소 깃허브의 지난 10년과 앞으로의 10년

Tech
Story

Dan Swinhoe | IDG Connect

지난 4월은 깃허브(GitHub)가 비공개 베타로 출시된 후 10년째 되는 시점이었다. 미국 샌프란시스코에서 탄생한 코드 저장소 기업인 깃허브의 공식 창립일은 4월이다. 그러나 비공식적으로는 2017년 10월이 10주년이었다. 2007년 당시 자체 깃허브를 구축하기 위해 레일즈(Rails) 앱을 가져와 최초의 깃 커밋을 실행했다.

이후 깃허브는 코드 저장소의 표준으로 자리를 잡았다. 구글 코드, 마이크로소프트 코드플렉스(CodePlex), 빗버킷(BitBucket), 소스포지(SourceForge) 등과의 경쟁에서도 우위를 점했고 깃랩(GitLab), 클라우드포지(CloudForge)와 같은 수많은 신생 경쟁업체가 시장에 진출하는 중에도 선두 위치를 공고히 했다. 깃허브는 2014년 '세계 최대 코드 호스트'에 오른 이후 현재까지 계속 몸집을 키우고 있다.

정상에 오른 깃허브의 다음 행보는 무엇이 돼야 할까? 수익성에 관한 불편한 진실 문제는 어떻게 풀어야 할까?

깃허브의 최고 전략 책임자인 헐리오 아발로스는 샌프란시스코에서 열린 깃허브 유니버스(GitHub Universe) 컨퍼런스에 앞서 열린 기자 회견에서 "그동안 '기업으로서' 깃허브뿐만 아니라 '소프트웨어 개발의 미래를 위한 대리자로서' 깃허브의 다음 10년을 고심해 왔다. 분명한 것은 우리가 아직 새로운 소프트웨어 개발의 매우 초기 단계에 있다는 사실이다. 5년,

10년, 15년 후를 생각하면 지금과는 완전히 다른 개발 프로세스가 자리 잡을 것이다"라고 말했다.

다음 단계: 단순 코드 라이브러리를 넘어라

아발로스에 따르면, 깃허브는 '코드의 알렉산드리아 도서관'이라는 개념을 넘어 더 완전한 형태를 목표로 한다. 그는 "깃허브는 완성된 책이 선반에 꽂혀 있는 도서관이 아니다. 모든 패치, 모든 버그, 부지불식간에 코드에 들어간 모든 보안 취약점, 그 취약점을 수정하는 패치까지 모든 것의 완전한 역사가 들어 있다. 이 방대한 데이터는 소프트웨어 개발 프로세스를 더 쉽게하고 접근성을 높이기 위해 활용할 수 있는 소중한 자산이



Credit : othree/flickr

다. 이는 향후 5~10년 동안 우리에게 중대한 테마가 되고 깃허브의 미래를 좌우할 것이다"라고 말했다.

깃허브 내부적으로는 도서관이라는 개념을 지나치게 수동적인 비유라고 평가한다. 아발로스는 "우리는 스스로를 소프트웨어 개발 커뮤니티의 관리인(steward)으로 본다. 그리고 이에 따르는 책임을 진지하게 받아들이고 있다"라고 말했다.

이런 생각은 깃허브의 기술 담당 SVP인 제이슨 워너도 비슷하다. 그는 "내부적으로 항상 사용하는 용어는 관리(stewardship)다. 우리는 그 책임을 매우 진지하게 받아들인다. 무엇을 더 할 수 있고, 이런 논의를 어떻게 발전시켜 나갈 수 있을까? 깃허브는 업계는 물론 전 세계적으로 중요한 위치에 있다. 따라서 모든 소프트웨어 개발과 개발자를 생각해야 한다. 또한 전체 소프트웨어 개발 기술을 어떻게 발전시킬지 소프트웨어 관점에서도 많은 고민을 하고 있다"라고 말했다.

깃허브는 현재 회사의 한계를 인식하고 있는 것으로 보인다. 유니버스 행사 기간 내내 깃허브는 소프트웨어 개발과 자체 데이터에 초점을 두고 그 사이에 발생하는 공백은 마켓플레이스 플랫폼이 채우도록 한다는 점을 여러 차례 강조했다.

아발로스는 "뭔가 큰 문제에 대처하는 것이 깃허브의 책임이라고 생각하지 않는다. 그러나 커뮤니티 내에 또는 우리가 보유한 데이터 내의 스택에 존재하는 요소와 툴을 더 신속하게 발굴할 기회가 있다면 이 역시 깃허브가 앞으로 해야 할 중요한 역할이라고 믿는다"라고 말했다.

흥미로운 것은 코딩 이외 부분에서도 깃허브가 활발하게 사용된다는 점이다. 마케팅과 영업을 포함해 깃허브의 직원 대부분은 자신의 업무에 깃허브를 사용한다. 다른 기업도 마찬가지다.

아발로스는 "변호사, 정책 입안자, 저널리스트, 과학자에게 깃허브는 무엇인가? 지금껏 접하지 못한 업계에 깃허브를 지원하고 협업을 촉진하기 위해 깃허브가 역할을 해야 한다. 단, 현재의 초점은 소프트웨어 개발 프로세스를 더 쉽게 만들고 접근성을 높이는 것이다. '의사가 사용할 수 있는 깃허브'를 언제 개발해 내놓을지 같은 문제에 매달리면 안 된다. 이런 문제는 데이터 과학자나 저널리스트가 더 쉽게 깃허브를 사용할 수 있는 마켓플레이스와의 통합이 이뤄지고 나면 검토할 수 있다"라고 말했다.

코딩의 미래

깃허브의 CEO 크리스 완스트래스는 깃허브 유니버스 기조연설을 통해 코딩의 미래는 '아예 코딩이 없는 세상'일 수도 있다고 말했다. 기계가 기계를 코딩하는 시대가 온다는 의미일까? 실제로 구글의 오토ML(AutoML) 같은 시스템은 사람보다 더 효과적인 머신러닝 코드를 만드는 것이 목표다. 그러나 깃허브는 코드에 자동화를 추가하는 것과 소프트웨어 개발자를 로봇으로 대체하는 것은 분명히 다르다고 강조한다.

깃허브 데이터 과학 부문 엔지니어링 관리자인 미주 한은 "확실히 말하지만, 코딩 로봇을 만드는 것이 아니다. 미래의 코딩은 지금보다 더 자연어 중심으로 이뤄질 것이다. 즉 미래의 개발



Credit : Ben Yuko Honda/flickr

자가 지금처럼 한 줄 한 줄 코드를 작성하지는 않을 것이다. '메뉴 표시줄이 있고 음악을 재생하는 안드로이드 앱을 만들어 줘'라고 말하는 형태가 될 것이다. 이를 위해 깃허브가 나서서 템플릿을 만드는 것도 고려할 만하다"라고 말했다.

아발로스는 이를 인력 대체가 아니라 소프트웨어를 만드는 사람에게 유용한 '요소로써의' 코드 자동화라고 표현했다. 그는 "사람에게 계속 집중하고 사람이 쉽게 작업을 할 수 있도록 하고 더 효율적인 개발 프로세스를 만들어, 사람이 발판, 즉 많은 프로그램에 있는 인프라에 시간을 소비할 필요가 없도록 하는 방법을 진지하게 고민하고 있다"라고 말했다.

MS의 깃허브 인수, 개발자가 우려해야 할까?

Tom Macaulay | Computerworld UK

지난 6월 마이크로소프트가 세계 최대 코드 저장소 기업 깃허브(GitHub)를 인수한다고 발표한 이후, 오픈소스 커뮤니티를 중심으로 우려의 목소리가 나오고 있다.

8,500만 코드 저장소와 2,800만 사용자를 보유한 깃허브의 매각 금액은 75억 달러다. 마이크로소프트는 회사의 성장을 가속화하는 한편, 마이크로소프트 플랫폼을 사용하는 개발자를 늘리기 위해 깃허브를 인수했다. 앞으로 깃허브는 마이크로소프트의 애저 클라우드와 통합돼 미래 성장 전략의 핵심 역할을 할 것으로 보인다. 이번 인수는 통상적인 조건과 규제 검토를 거쳐 올해 말 완료될 예정이다.

마이크로소프트의 CEO 사티아 나델라는 블로그 포스트를 통해 이번 인수의 배경을 설명했다. 그는 "무엇보다도 우리는 개발자 라이프 사이클의 모든 단계에서 개발자의 역량을 강화하도록 지원할 것이다. 아이디어 단계부터 협업, 클라우드 배치에 이르는 모든 단계다. 깃허브는 개발자가 참여할 수 있는 개방형 플랫폼으로 유지된다. 개발자는 계속해서 자신의 프로젝트에 원하는 프로그래밍 언어, 도구 및 운영체제를 사용할 수 있고 어떤 클라우드, 어떤 기기에서든 배포할 수 있다"라고 말했다.

이어 "두 번째로 우리는 깃허브의 직접 판매 및 파트너 채널과 마이크로소프트 글로벌 클라우드 인프라 및 서비스에 대한 액세스를 가속화할 것이다. 마지막으로 마이크로소프트의 개발자 도구와 서비스를 새로운 고객에게 제공한다. 가장 강조할 부분은 우리가 이번 계약에 따르는 책임을 충분히 인식하고 있다는 사실이다. 깃허브 커뮤니티의 청지기가 되기 위해 최선을 다할 것을 약속한다. 깃허브 커뮤니티는 개발자 우선 정신을 유지하고 독립적으로 운영되며 개방적인 플랫폼으로 계속 남을 것이다"라고 덧붙였다.

깃허브의 다음 행보는?

깃허브는 2015년 기준 20억 달러로 평가됐다. 그러나 수익성이 좋은 기업은 결코 아니었다. 개인용 무료 계정을 운영하는 수익 모델로 기업용 유료 계정 판매에만 의존했기 때문이다. 마이크로소프트는 현재 깃허브에 1,800개의 저장소를 보유하고 있다. 이 코드 공유 사이트에서 가장 활발히 활동하는 기업이다. 이번 인수를 통해 마이크로소프트는 광범위한 개발자를 자사 플랫폼으로 유인할 수 있을 것으로 기대하고 있다.

그러나 이번 인수에 대해 일부 개발자는 우려하고 있다. 나델라가 2014년 마이크로소프트에 합류한 이래 오픈소스 프로젝트와 개발자를 끌어들이기 위한 활동을 활발히 펼치고 있지만, 마이크로소프트가 그 이전에 오픈소스에 보였던 적개심은 여전히 개발자의 뇌리에 남아 있다. 나델라의 전임자였던 스티브 발머는 2001년 "리눅스는 암"이라고 말하기도 했다. 단 마이크로소프트는 그 이후 회사의 닷넷 프로그래밍을 오픈소스화하고 리눅스 재단에 합류하는 등 태도를 바꿨다.

이에 대해 나델라는 블로그를 통해 "마이크로소프트는 오픈소스에 옮긴하고 있다. 우리는 오픈소스 세계에 들어왔으며 이제는 가장 활발하게 활동하며 기여하고 있다. 우리의 개발자 도구와 프레임워크 일부도 이미 오픈소스화했다. 우리의 최근 행보와 미래 비전으로 마이크로소프트를 판단해 달라"라고 말했다.

그러나 나델라의 바램과 달리 개발자의 반발은 이미 가시화되고 있다. 깃허브의 경쟁사인 깃랩(GitLab)으로 갈아타려는 움직임이 나타나는 것이다. 실제로 마이크로소프트의 깃허브 인수가 알려지자 깃랩의 일일 저장소 용량이 10배나 늘어났다. 마이크로소프트의 노력과 나델라의 약속이 과연 깃허브 개발자의 마음을 돌릴 수 있을까?

설계를 통한 보안

자동화와 깃허브의 ‘관리인’ 역할에 대한 이야기를 듣고 있으면 자연스럽게 다음 질문이 떠오른다. 바로 보안이다. 깃허브 플랫폼에 호스팅되는 코드의 안전을 보장하는 것은 깃허브의 역할일까? 깃허브는 로봇으로 사람을 대체하지 않는다는 점을 강조하지만, 한편으로는 보안을 소프트웨어 개발의 중심에 두기를 원한다. 실제로 보안은 사람이 실수할 가능성이 매우 큰 영역이기도 하다.

워너는 “일반적으로 보안은 소프트웨어를 개발한 이후의 문제로 취급되는 경향이 있다. 그러나 개발 과정에서 최선의 보안 접근 방법을 자동으로 결정하거나 최소한 자동화된 제안을 통해 보안을 강화할 수 있다. 이런 부분은 잠재력이 매우 크다. 코드 생성 시점에서 보안을 자동화한다면 전체 업계가 어느 정도나 혜택이 될까? 매우 흥미로운 주제이며, 개인적으로는 더 안전한 인터넷과 더 안전한 소프트웨어 환경으로 이어질 것으로 믿는다”라고 말했다.

이런 작업의 시작은 코드 종속성 그래프다. 이를 사용하면 프로젝트에서 코드의 종속성이 얼마나 많은지(미주 한에 따르면 많게는 100개가 넘는다), 주기적으로 업데이트되는 코드는 무엇인지, 어느 코드가 안전하지 않은지 등을 볼 수 있다.

미주 한은 “선제적 보안에 대한 우리의 비전은 명확하다. 특정 프로젝트가 취약하다면 그 프로젝트에 아예 종속성을 두지 않는 것이다. 예를 들어 개발자가 코드 편집기에서 작업할 때 해당 프로젝트에 대한 종속성이 생기지 않도록 사용하지 말 것을 조언한다. 더 나아가 좋은 대안과 이를 프로젝트에 통합하는 방법까지 제안할 수도 있다”라고 말했다. 깃허브는 코드에서 알려진 취약점을 찾아, 가능한 경우 수정을 제시하는 보안 알림 기능도 이미 도입했다.

다음 1억 번째 개발자

또한, 아발로스는 소프트웨어 개발자 육성 프로세스의 속도를 더 높여야 한다고 지적했다. 그는 “5년, 10년, 15년 후는 고사하고 지금의 글로벌 비즈니스 요구 사항을 충족하는 소프트웨어 개발자도 부족한 실정이다”라고 말했다. 이를 위해 깃허브는 인공 지능으로 소프트웨어 개발의 편의성과 접근성을 높이는 방법을 찾고, 이전까지 생각해본 적 없는 회사의 각 부분에서 개발자를 찾는데 도움을 제공하려 노력하고 있다.

아발로스는 “기업 내에서 개발자를 발굴하는 방법을 적극적으로 제안할 것이다. 현재 자신의 업무와 무관한 분야의 직원이 깃허브의 루비 또는 파이썬 프로젝트를 진행 중인 경우가 있고, 특정 프로젝트를 맡겨야 하는데 적임자가 기업 내에 있는지 알 수 없는 경우도 있을 것이다. 이를 통해 현재 기업이 처한 개발자 부족 상황을 어느 정도 보완할 수 있다”라고 말했다.

미주 한도 “요즘은 개발자라고 해서 꼭 컴퓨터 과학을 전공할 필요는 없다. 많은 새로운 개발자가 온라인으로 코딩 강좌를 듣는다. 부동산 중개업자로 일하면서 앱 또는 프로젝트에 관심이 많은 사람도 있다. 코딩하는 사람이 늘어나는 것은 긍정적이다. 깃허브는 이들이 안전하고 성능이 좋고 다른 사람이 이해하기도 쉬운 코드를 작성하도록 지원할 것이다”라고 말했다. 