GitHub 소개

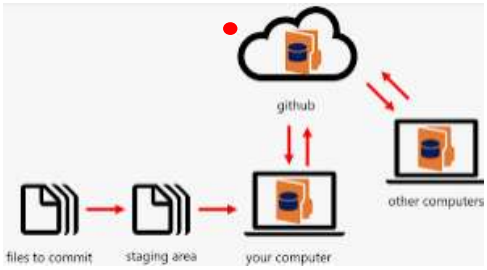
GitHub의 개념
기본적인 사용법
팀 개발에서의 사용법

1. 소개

- 엔지니어의 필수(Git), 팀-개발을 위한 필수(GitHub)

1.1 버전관리

- 의미 : 소스코드를 비롯한 파일 변경내용 (버전)을 관리하는 것
  파일을 추가하거나 변경의 이력 정보를 관리
  과거의 변경사항확인, 특정 시점의 내용으로 복원이 가능
- 효과 : 개발생산성 향상, SW오류 적극대처, 개발내용 팀원간 연계



https://www.ahamedsafnaj.com/2019/08/introduction-to-git-github.html

1.2 Git, Repository 그리고 GitHub

- Git : 분산형 버전 관리시스템 ← 현재 팀 개발의 대부분 채택
  개별 컴퓨터에 저장소 만들고 개발
- Repository (저장소)
  버전관리에 의해 관리되는 파일 및 히스토리 정보저장 영역
- Git에서의 동작
  (1) 개별 컴퓨터내 저장소(Local Repo.)에서의 작업 후,
  (2) 원격지 서버 등의 저장소(Remote Repo.)로 통합관리
- GitHub
  원격저장소 기능포함, 팀 개발기능을 제공하는 web서비스
  소프트웨어 개발 프로젝트를 위한 소스 코드 관리 서비스
  소스코드 버전 관리 및 열람, 버그추적, SNS 기능 보유
  (저장소기능 + 코드리뷰기능 + 의견공유기능)

➔ GitHub와 유사한 서비스 예 = Bitbucket과 GitLab 등

1.3 GitHub 사용 이유

(1) 무료의 개인 저장소 제공

소스코드를 관리하는 단위를 2개의 저장소

일반이용자에게 공개되는 Public 저장소

공용관계자로 제한 할 수 있는 개인 저장소

소스코드 비공개 원할시 개인저장소 선택(이전 GitHub는 유료)

Bitbucket과 GitLab에서는 무료

2020 년 4 월부터 GitHub도 무료 개인 저장소(무제한)

(2) 오픈소스 커뮤니티에 활성화된 SNS 기능

많은 오픈 소스가 관리되며, 많은 오픈 소스 커뮤니티가 형성됨

지역 사회에서 누가 얼마나 기여하고 있는지의 시각화 기능

개발 멤버의 지식을 공유하는 기능 (Wiki, GitHub Pages)

소스 코드의 업데이트 등에 관한 논의 기능 (Issues, Code와의 연동)

(3) 고급 기능의 확충

GitHub는 2018 년에 Microsoft에 인수된 기업

인수 후에도 GitHub를 독립 회사로 유지할 의향

다양한 고급 기능들 보유

MS의 Visual Studio Code의 무료 온라인이용(Codespaces)

SW개발에서 공개까지의 흐름을 지원(GitHub Actions)

2. Git와 GitHub의 기본 사용법

- Git 사용을 위한 단계 :

(1) Git 설치

(2) Git의 초기 설정

(3) GitHub 초기 설정(계정)

(4) 원격저장소 생성

(5) 로컬저장소 생성

(6) 로컬저장소에 Commit

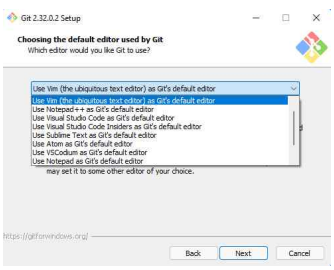
(7) 원격저장소에 Pull

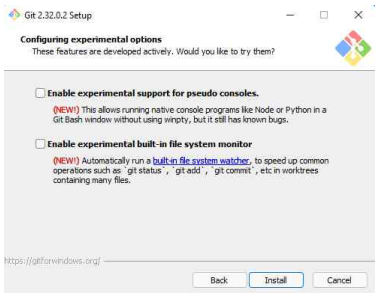
2.1 Git 설치

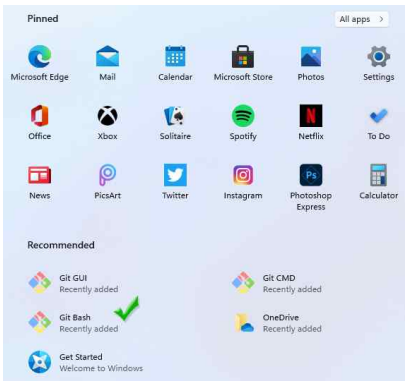
- Mac의 경우 : 기본설치 되어있음

- **Windows의 경우 : 설치 프로그램을 다운로드 & 설치**

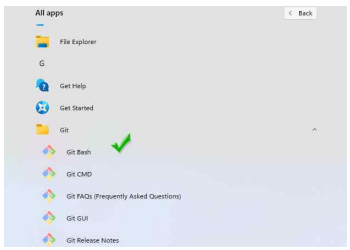
- Windows의 경우 예: https://gitforwindows.org Git-2.32.0.2-64-bit.exe







or



- 설치가 완료되면 Git Bash를 시작

  Git Bash는 Windows에서 Git 명령을 위한 도구

  Git Bash 시작 후 Git 버전 정보 확인

  $ git --version

  git version 2.32.0.windows.2

(이하 Git Bash와 같은 명령 줄로 소개, GUI 도구도 사용가능 함)

2.2 Git 의 초기설정

- Git에 의한 소스코드의 변경내역 확인 : 확인 정보 필요

  (작업자 식별정보로 아이디와 이메일주소 등록작업)

(1) 사용자명

$ git config --global user.name 사용자명 (*)

(2) 이메일주소

$ git config --global user.email 이메일주소 (*)

(3) 설정내용을 화인

$ git config --list

core.autocrlf=true

core.fscache=true

...

user.name= 사용자명 (*)

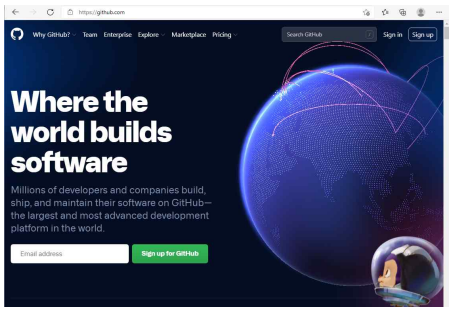
user.email= 이메일주소 (*)

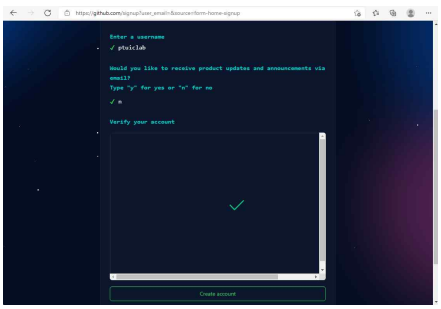
2.3 GitHub의 초기설정 : 신규계정(account) 개설

(1) GitHub에 액세스하여 이메일주소/ 사용자명/ 비밀번호 입력

예: (*)

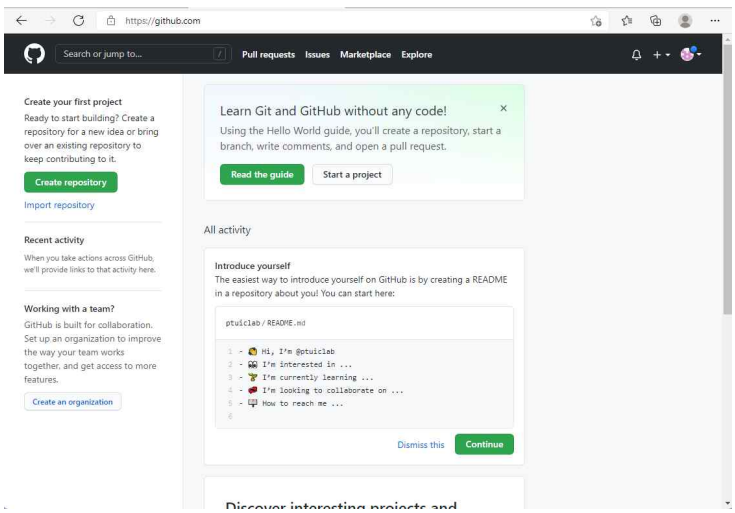
(2) "Sign up for GitHub"버튼을 클릭





(3) "Verify your account"설문응답 후 " Create account "클릭

(4) 인증메일 송신됨. 메일 수신/확인 후, 번호인증 실시후



➔ GitHub : 분산적인 SW의 변경관리 지원

- 소스코드를 비롯한 파일 변경내용 (버전)을 관리
- 파일을 추가하거나 변경의 이력 정보를 관리

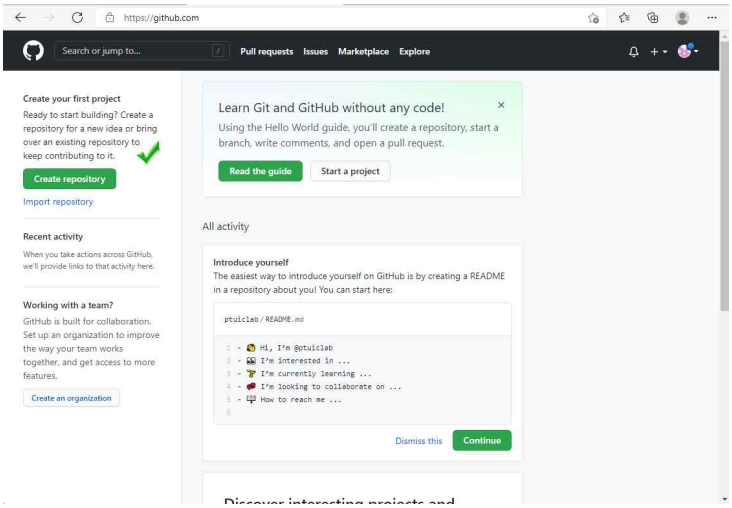
[참고] GitHub 이용을 위한 사전 지식 3가지

2.4 원격저장소(Remote Repository) 작성

(1) GitHub 액세스하여, 원격 저장소를 새로이 생성 :

메인화면에서 "Create Repository"버튼 클릭

저장소 생성화면으로 이동



(2) "Repository name"에 저장소이름을 입력

예) "PTnetsw"라는 저장소이름 입력

(3) 저장소 형식으로 "Public"또는 "Private"를 선택

2019년부터 Free 이용자도 "Private" 선택가능

(GitHub는 오랫동안 유료회원 만 "Private"를 선택가능 했음)

•"Public" 선택시 : 공개형 (다른 사용자가 볼 수 있음)

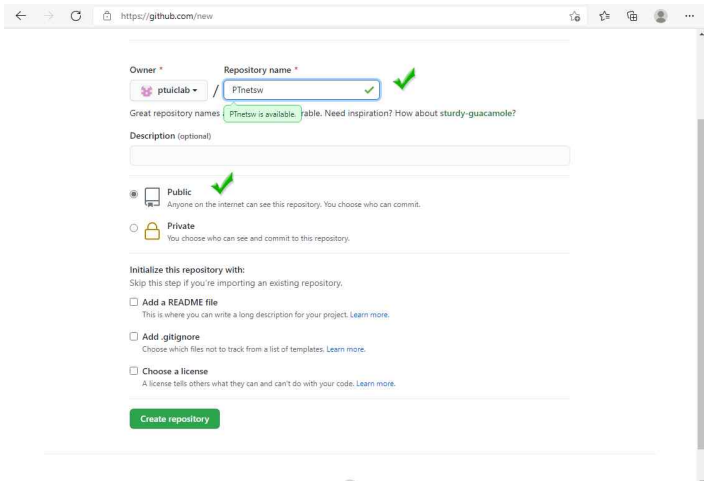
•"Private" 선택시 : 비공개형

(4) "Initialize this repository with a README"

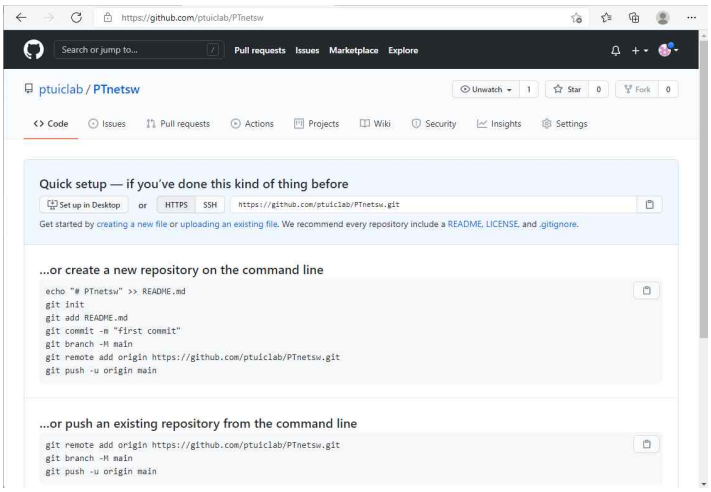
README파일(사용법설명)을 사전 준비된 경우 ON으로..

(5)".gitignore"나 "license "은 None으로 선택..

나중에 추가하거나 변경 가능함(None)



(6)"Create repository"버튼을 클릭하여 저장소 생성



2.5 로컬저장소(Local Repository) 작성 : 파일의 작성 및 편집

- Windows이면 PowerShell (Mac에서는 터미널)

(1) 예), 본인PC에 "PTnetsw"라는 디렉토리를 작성한 후...

$ mkdir PTnetsw

(2) PTnetsw디렉토리를 저장소로 초기화: git init 명령 입력

(명령은 PTnetsw 디렉토리에서 실시 해야 함)

$ cd PTnetsw

$ git init ← Git 저장소를 새로 만드는 명령

```
Initialized empty Git repository in C:/Users/Lee/PTnetsw/.git/
```

(버전관리를 하지 않은 기존 프로젝트를 Git 저장소로 변환,
새로운 빈 저장소를 생성하고 초기화에 사용함)

➔ git init 명령을 실행시 현재 디렉토리를 Git 저장소로 변환

(3) "PTnetsw"디렉토리 아래에 "index.html"라는 파일 작성

(텍스트편집기 이용가능, 아직 저장소에 추가되지 않은 상태)

$ vi index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <title> Hello </title>
</head>
<body>
      <p> Hello World ! </p>
</body>
```

(4) 현재의 상태를 확인 : 인덱스에 등록되지 않은 파일로 표시됨

$ git status

```
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html
nothing added to commit but untracked files present (use "git add"
to track)
```

2.6 로컬저장소에 커밋 : 파일생성/변경/삭제정보 index에 추가

- 생성한 파일을 로컬 저장소에 파일을 추가하는 과정 :

  ← 2개 명령어 사용 : add , commit 명령어

- 커밋 = 파일 추가/변경을 로컬 저장소에 반영하는 작업
- 인덱스= 커밋을 위해 변경내용을 임시로 저장하는 위치

  (인덱스에 추가된 파일만이 커밋 대상이 됨 !!)

(1) Git의 "인덱스(=Staging area)"에 "index.html"을 추가

$ git add index.html ← 이후, tracking&관리 가능

```
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
```

(2) 인덱스에 추가된 파일들을 커밋 : 변경사항의 반영처리과정

$ git commit -m "[Add] index"

```
[master (root-commit) cd4c559] [Add] index
 1 file changed, 10 insertions(+)
 create mode 100644 index.html
```

➔인덱스내의 파일(index.html)이 → 로컬 저장소에 추가됨)

➔ -m은 "커밋 설명메시지", 추후 커밋 내용을 확인에 도움

$ git log ← 변경이력 확인

```
commit cd4c559737fffd2c394fbd8a0711161174771bf5 (HEAD -> master)
Author: ptuiclab <ptuiclab@gmail.com>
Date:   Sat Sep 18 22:17:43 2021 +0900

    [Add] index
```

2.7 원격저장소(Remote Repository)에 푸쉬(Push) 하기

- GitHub상의 원격저장소에 푸쉬하기(로컬변경사항을 원격에 반영):

← 2개 명령어 사용 : remote add, push 명령어

(1) 연결등록(로컬저장소<->원격저장소) : 로컬저장소(PTnetsw)에서..

$ git remote add origin https://github.com/ptuiclab/PTnetsw.git

➔ 원격저장소의 정보가 로컬저장소에 등록됨

(2) 변경사항 반영 : 로컬저장소에서 원격저장소로 푸시(반영)

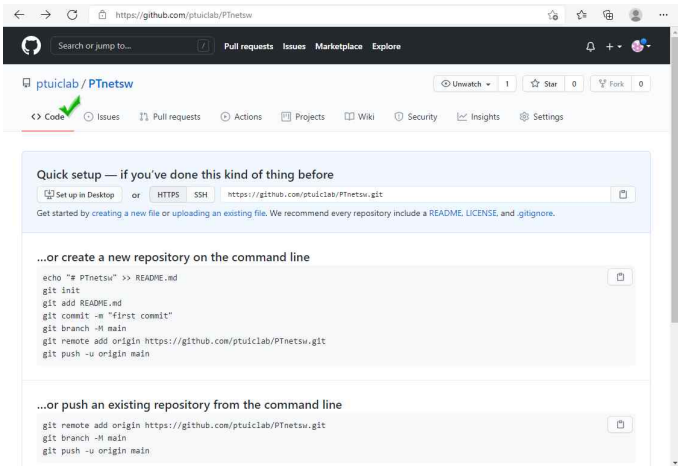
$ git push origin master ← default branch

➔ GitHub에서도 반영됨을 확인가능 (+원격저장소에 반영)

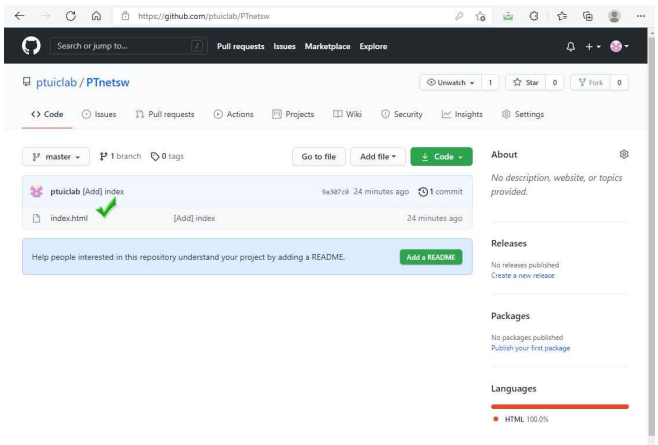
```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 328 bytes | 328.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ptuiclab/PTnetsw.git
 * [new branch]      master -> master
```

➔ GitHub 사용자 ID/PW요구시 계정등록시 설정 ID/PW

[GitHub 화면] 확인 :



[GitHub 화면 – "code" 클릭] 확인 :



(GitHub(Web) "Code" 클릭으로 다운로드 가능)

-GitHub의 기초 사용순서 재정리 (git init 은 1회만 실행)

1) Github에 저장소를 작성 (git init)
2) 파일의 작성, 편집을 할
3) 파일의 생성 / 변경 / 삭제를 git 인덱스에 추가 (git add)
4) 변경 사항을 로컬 저장소에 커밋 (git commit)
5) 로컬 저장소를 밀어 원격 저장소에 반영 (git push)

➔ 작은 작업 단위로 커밋(4)을 하고,
상당부분 작업이 완성되었을 때 푸시(5) 함

➔ 커밋시 커밋 메시지작성 (반드시 추천): 변경내용 이해에 도움

3. 팀-개발에서의 Git와 GitHub 사용법

- Git과 GitHub을 팀의 다른 멤버와 개발을 할 때의 기본 사용법
  ➔ 실제 SW개발 현장에서 자주 이용되는 브랜치 (branch)
- Git은 팀으로 SW개발시 사용하는 브랜치(branch) 구조 있음
  ➔ 동시에 이루어져야 하는 여러 버전의 관리 구조 지원
- 브랜치의 기초 사용 예: ← 저장소의 clone 이후
  - 브랜치 생성, 이동
  - 브랜치에서 개발 작업
  - 브랜치로 Push
  - 브랜치 Merge
  - 브랜치에서 Pull
  - 브랜치 Delete

3.1 저장소(Repository)를 clone 하기

- (가정) 원격저장소에 로컬변경사항이 반영되어있고,
  ➔ 새로운 인력이 개발과정에 참여시 :
  기존의 원격저장소에서 소스코드를 획득 가능
  ➔ 본인의 새로운 추가 작업을 실시하는 경우:
  GitHub(Web) "Code" 클릭으로 소스 다운로드 가능

(1) 새로운 디렉토리(예: "PTnetsw1")를 생성

```
$ cd ..                          ← /c/Users/Lee 됨
$ mkdir PTnetsw1
$ cd PTnetsw1
$ git clone https://github.com/ptuiclab/PTnetsw.git
Cloning into 'PTnetsw'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1
```

$ cd PTnetsw/

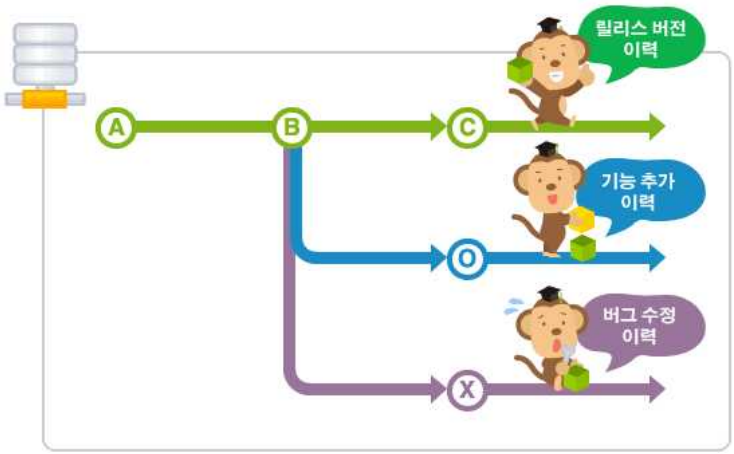
```
Lee@DESKTOP-SVOHT2P MINGW64 ~/PTnetsw1/PTnetsw (master)
```

$ ls

index.html

(원격저장소에서의 "가져오기 작업" = 클론 (clone) 이라함)

3.2 브랜치(Branch)의 작성, 이동



A:commit1, B:commit2, C:commit3, O:commit4, X:commit5

https://backlog.com/git-tutorial/kr/stepup/stepup1_1.html

- 브랜치를 생성하려면 : git branch 명령을 사용

예: 현재 작업중인 브랜치에 "feature1"브랜치 생성하는 경우

(디폴트 브랜치= "master")

$ cd ~/PTnetsw1/PTnetsw/ ← 예제 준비환경

$ pwd

/c/Users/Lee/PTnetsw1/PTnetsw

(1) master에서 feature1 브랜치를 생성하기

$ git branch feature1

(2) "feature1" 브랜치에서 작업을 위하여 브랜치로 이동하기

$ git checkout feature1

```
Switched to branch 'feature1'

Lee@DESKTOP-SVOHT2P MINGW64 ~/PTnetsw1/PTnetsw (feature1)
```

(3) 현재 로컬저장소의 브랜치 정보표시하기

(현재 로컬 저장소에 "master", "feature1"브랜치 표시됨)

$ git branch

```
$ git branch
* feature1                    <-"*"는 작업을 수행하는 현재 지점
  master
```

3.3 해당 브랜치(Branch)에서의 커밋(commit) 과정

- 예: feature1 지점에서의 작업 처리(commit)

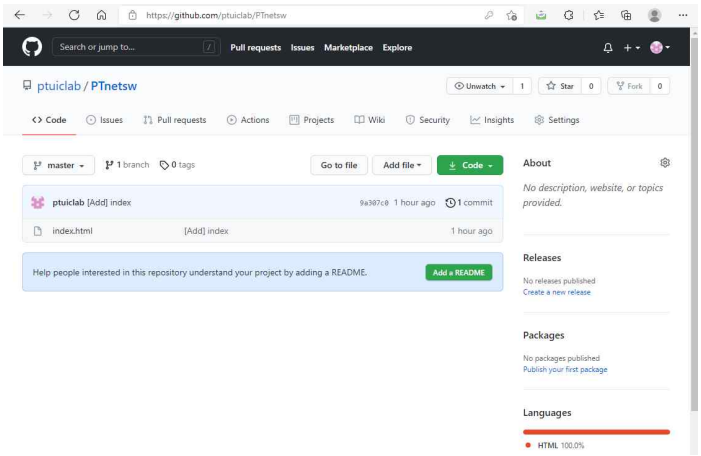
(1) login.html라는 파일 작성

(2) 만든 login.html을 feature1 브랜치에서 커밋하기

$ git add login.html

$ git commit -m "[Add] Login "

```
[feature1 7defae6] [Add] Login
 1 file changed, 12 insertions(+)
 create mode 100644 login.html
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (feature1)
```



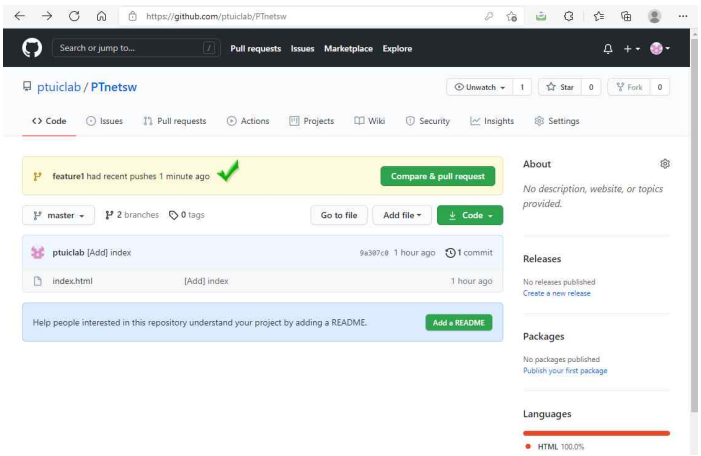
3.4 원격저장소로의 푸쉬(Push) 과정

(3) 원격 저장소 (GitHub)에 feature1 브랜치를 반영(=푸쉬)

$ git push origin feature1

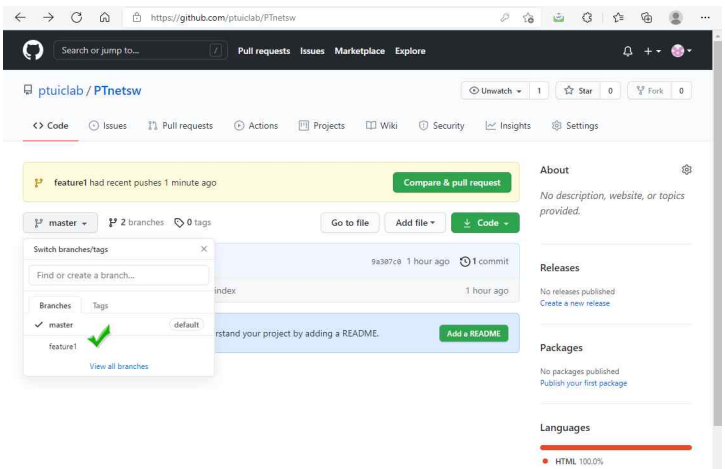
```
...
remote:
remote: Create a pull request for 'feature1' on GitHub by visiting:
remote:     https://github.com/ptuiclab/PTnetsw/pull/new/feature1
remote:
To https://github.com/ptuiclab/PTnetsw.git
 * [new branch]      feature1 -> feature1
```

[GitHub(Web)] 화면 Reload 시

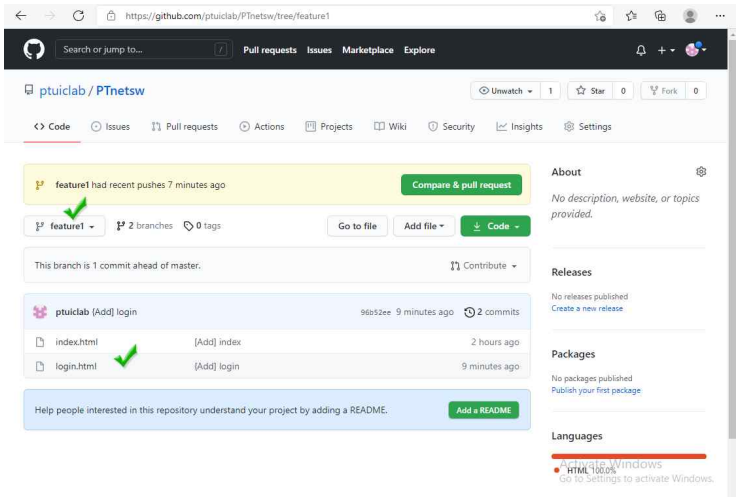


(참고:2-7절과 달리 push할 위치가 feature1으로 되어있음)

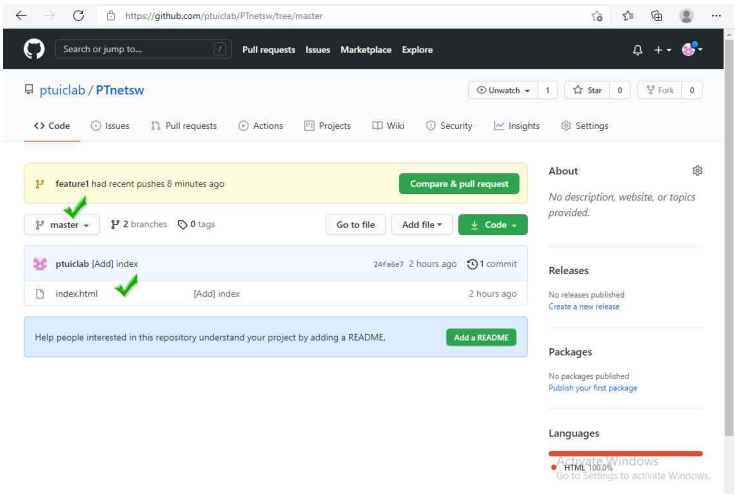
- GitHub reload시 2개 브랜치(master와 feature1) 존재함 확인



- feature1 branch선택시: 변경내용이 Push되었음을 확인가능



- master branch 선택시: 변경내용 미-반영 확인 (Push 대상아님)



3.5 브랜치의 병합(Merge) 과정 : 코드 리뷰(Review)후

병합 = "특정 브랜치의 변경사항을 다른 브랜치로 통합하는 과정

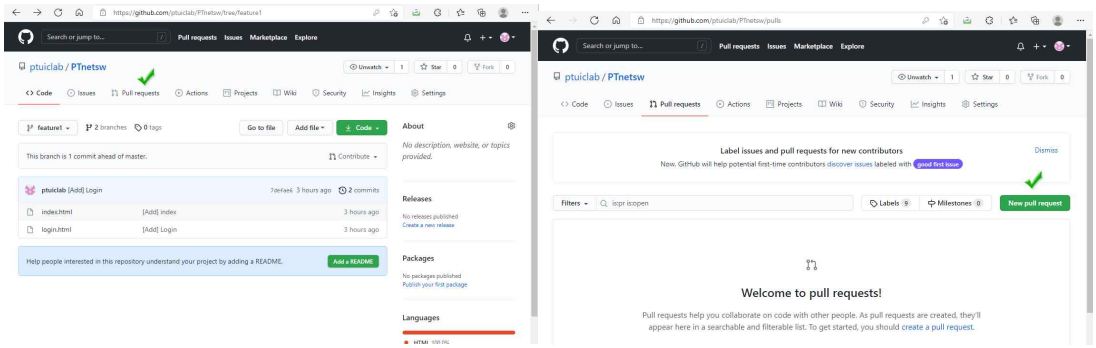
- (가정) 해당 브랜치(예:feature1)에서의 개발 작업이 완료되면,

  ➔ 메인브랜치(예: master)로 변경사항을 통합

- 병합시 코드리뷰(Code review) 가능
  - GitHub(Web) 상에서 검토의뢰(Pull 요청기능) 가능
  - 필요시 수정요청 가능(리뷰시 Comment이용)

(1) 검토 의뢰 : Reviewer에게 검토의견 요청

feature1 브랜치 열고, "Compare & Pull Request"클릭

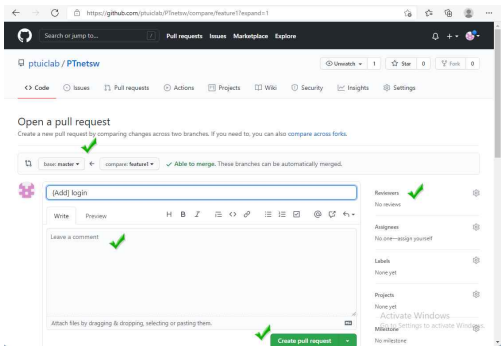


(2) 다음과 같은 Pull Request(예)를 작성

- ✓ 작성 후 "Create pull request"버튼을 클릭
- ✓ 병합대상 브랜치(Sorce, Dest.) ← Able to merge
- ✓ 검토하는 사람 : ← Reviewer (참가자들)

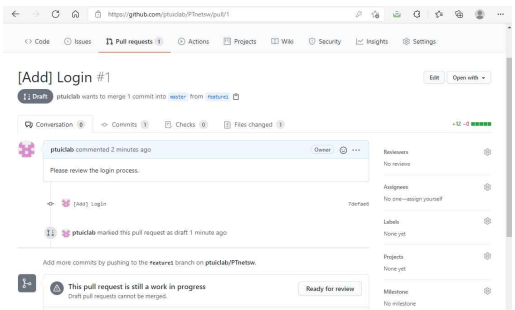
.github/CODEOWNER (해당 저장소 내)

- ✓ 검토 내용 ← Leave a comment
- ✓ 검토/병합할 대상 소스코드 ← 하단에 표시됨



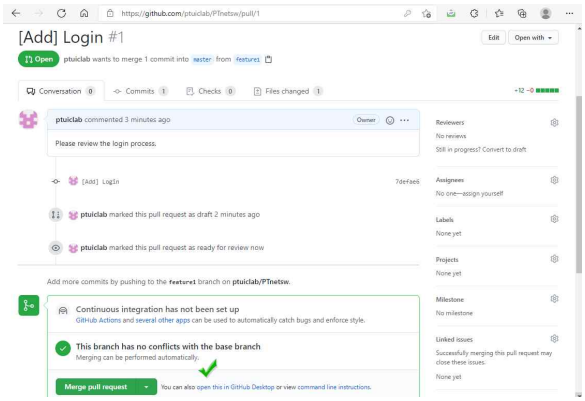
(3) 검토 실시 : Pull Request를 받은 검토자(Reviewer)는...

- ✓ 의견 존재시 소스코드상에 주석추가& 개발자에게 수정요청

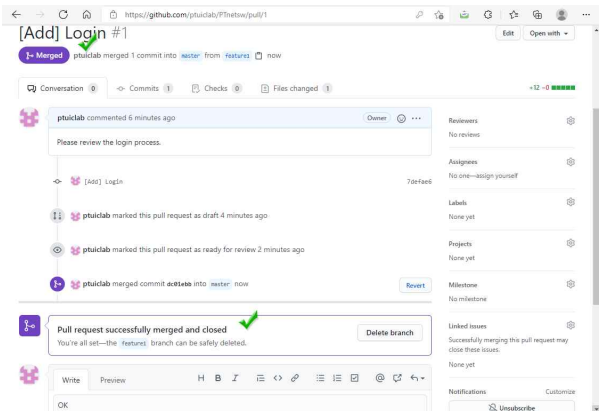


- ✓ OK시, "Merge pull request"클릭(검토 완료)

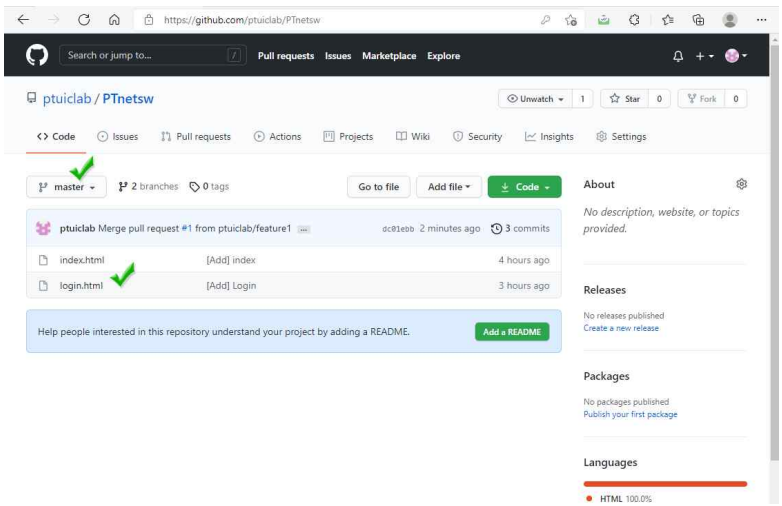
이때, 브랜치병합 & feature1의 내용이 master에 반영됨



Confirm Merge 시...



➔ **병합이 완료되어, login.html가 master에도 표시됨 확인**
("ptuiclab / PTnetsw" 클릭 하여 확인)



-실제의 리뷰에서는

- 개발기능 및 버그수정 등 다수 소스코드에 Pull requests가능
- 여러 번의 지적/변경요청이 발생함

3.6 로컬저장소로 Pull하기 (원격저장소에서)

- 원격저장소의 master에는 병합처리가 되어도...
  개별 로컬저장소의 master에는 아직 병합처리 이전의 상태임
- 추후 계속적인 개발 작업을 위해서는
  병합내용(login.html)이 로컬저장소의 master에도 반영 필요
  "Pull"작업으로 원격저장소에서의 변경내용을 가져오기 가능
- 2 단계의 처리 : 브랜치변경 + 변경내용 Pull

(1) 브랜치를 master로 전환

(현재, 아직 방금 병합한 login.html가 확인되지 않음)

```
$ git branch
* feature1
  master
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (feature1)
$ ls
login.html
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (feature1)
```

$ git checkout master ← master로 전환

```
Switched to branch 'master'
D       index.html
Your branch is up to date with 'origin/master'.

Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (master)
$ git branch
  feature1
* master
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (master)
```

(2) 원격 저장소의 master로 최신 변경 내용을 가져옴(pull 명령)

$git pull

```
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 624 bytes | 624.00 KiB/s,
done.
From https://github.com/ptuiclab/PTnetsw
```

```
   9a307c0..dc01ebb  master     -> origin/master
Updating 9a307c0..dc01ebb
Fast-forward
 login.html | 12 ++++++++++++
 1 file changed, 12 insertions(+)
 create mode 100644 login.html
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (master)
```

(참고) 이제 로컬저장소의 master 에도 최신 내용이 반영되었음

```
$ ls -l
total 1
-rw-r--r-- 1 Lee 197121 170 Jul 26 23:08 login.html
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (master)
$ git log
commit dc01ebbd94c3a5945d185d391caab7508f35dbbd (HEAD ->
master, origin/master, origin/HEAD)
Merge: 9a307c0 7defae6
Author: ptuiclab <87886499+ptuiclab@users.noreply.github.com>
Date:   Mon Jul 26 22:57:49 2021 +0900
    Merge pull request #1 from ptuiclab/feature1
    [Add] Login
commit 7defae6435604deb80f0904dba8d52247c2a680c
(origin/feature1, feature1)
Author: ptuiclab <ptuiclab@gmail.com>
Date:   Mon Jul 26 19:37:30 2021 +0900
    [Add] Login
commit 9a307c01fd19d3b96b8d4aa9f74731cd8b5e79b9
Author: ptuiclab <ptuiclab@gmail.com>
Date:   Mon Jul 26 18:49:33 2021 +0900
    [Add] index
Lee@DESKTOP-SV0HT2P MINGW64 ~/PTnetsw1/PTnetsw (master)
```

3.6 브랜치(Branch)의 삭제

- 사용하지 않는 지점은 제거 가능

➔ 작업이 완료된 경우에도 일반적으로 남겨 둠

(실제 개발현장에서는 잘못 작성되었을 경우 제외)

- 브랜치 삭제 명령

```
$ git branch -d feature1
$ git branch            ← 결과확인
```

➔ 이렇게... 다른 개발자와의 내용협의/검토/확인하며, SW의 협업/공동개발이 진행 됨(Git/GitHub이용!!)

[참고용 데이터]

- index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
        <meta charset="UTF-8">
        <title> Hello </title>
</head>
<body>
        <p> Hello World! </p>
</body>
</html>
```

- Repository 위치 : khlee16/PTnetsw1

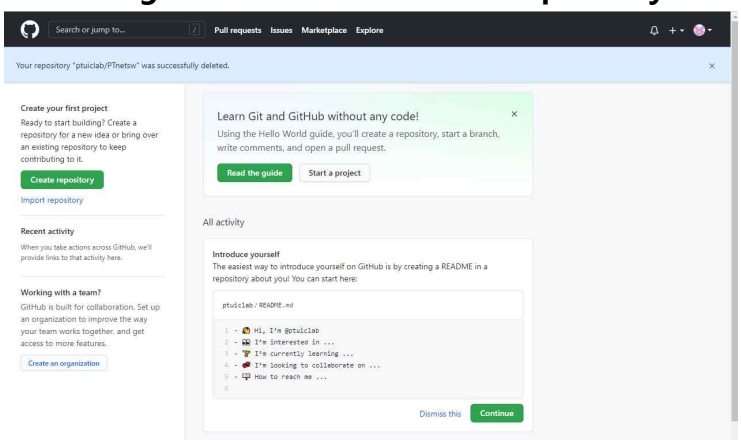
[저장소내 파일삭제]

- GitHub에서 파일클릭 -> 삭제

[원격저장소 초기화]

- “PTnetsw” repository에서 -> Your repositories

  -> Danger Zone -> Delete this repository



- Setting 메뉴에서...

[참고] GitHub 사용을 위한 기초

I. 용어 정리

(1) 로컬 저장소와 원격 저장소

- 저장소 : 파일이나 디렉토리의 상태를 저장하는 장소
  변경이력 관리용 디렉토리 등을 저장소의 관리하에 두기위함
  디렉토리 내의 파일 등에 대한 변경 내역 기록가능
- 저장소의 2 종류 :
  개인 컴퓨터에 있는 "로컬 저장소"
  서버 등 네트워크에 있는 "원격 저장소"
- 기본 동작흐름
  로컬 저장소에서 작업, 작업 결과를 원격 저장소에 Push.

(2) 커밋(commit)과 푸시(push)

- 커밋 (commit)
  파일을 추가하거나 변경 내용을 저장소에 저장하기
- 푸시 (push)
  파일을 추가하거나 변경 내용을 원격 저장소에 업로드 하기

(3) 브랜치(branch)

- 여러 버전 (병렬)관리를 위해 브랜치 (branch)라는 기능
  SW개발은 유지보수와 새 기능추가 및 버그수정 등 동시진행
- 브랜치는 SW개발 역사의 흐름을 분기하여 기록 하기 위함.
  분기지점은 다른 지점의 영향을 받지 않음
  (따라서, 같은 저장소에서 각 개발을 해 나갈 수 있음)

(4) 인덱스(index)

- Git 관리가 필요 없는 파일도 존재
  저장소 관리하의 디렉토리에 임시로 사용하는 파일 등
- Git : 버전 관리하려는 파일이나 폴더를 "인덱스"라는 대장에 등록
  필요한 경우에만 버전 관리를 할 수 있도록 함

-따라서, 새로운 폴더/파일의 생성시, 색인의 생성 필수(커밋 이전에)

II GitHub 주요 사용 명령어 예

(1) git status

저장소의 상태를 확인하기 위해 사용하는 명령어

현재 브랜치의 이름과 추가/변경된 파일 및 디렉토리 목록을 표시

git status

(2) git add

파일이나 디렉토리를 인덱스에 추가하는 데 사용

추가시 메타문자로 여러 대상을 지정할 수도 있습니다.

git add [file_name]

(3) git commit

인덱스에 추가된 파일이나 폴더내용을 저장소에 기록하는데 사용

메시지를 지정하려면 -m 옵션 이용

또한 -a 옵션은 수정파일을 검색 & 인덱스에 추가도 실행됨

git commit -am "A first commit"

(4) git branch

지점에 대해 다양한 작업을 수행하기 위해 사용하는 명령어

- git branch [branch-name] : 지점 만들기
- git branch : 브랜치 목록보기
- git branch -d [branch-name] : 지정한 지점 삭제

(5) git checkout

로컬 저장소의 브랜치를 전환 할 때 사용하는 명령어

git checkout [branch-name]

(6) git log

로컬 저장소의 커밋 히스토리를 탐색에 사용하는 명령

-n 옵션 내역보기 수를 지정 가능

git log -n 10

(7) git grep

저장소의 파일 내용에서 검색하고자 할 때 사용하는 명령어

특정단어가 포함된 파일 검색

git grep "검색 단어"

(8) git clone

기존 원격 저장소를 로컬에 가져오기 위하여 사용하는 명령어

git clone [url]

(9) git remote

원격 저장소를 조작하는 데 사용하는 명령어

- git remote : 원격 저장소의 이름목록 표시
- git remote -v : 원격 저장소에 대한 자세한 목록보기
- git remote add [name] [url] : 원격 저장소 추가
- git remote rm [name] : 원격 저장소 제거

(10) git reset

로컬 저장소의 커밋을 취소하기 위하여 사용하는 명령어

커밋오류 수정 누락이 있을 때 사용

git reset -soft HEAD ^

(11) git merge

현재 브랜치에 다른 브랜치의 변경사항을 가져올 때 사용하는 명령

(브랜치 bug-fix를 master 브랜치에 병합)

git checkout master

git merge bug-fix

(12) git pull

원격 브랜치의 변경 사항을 가져오기 위해 사용하는 명령어

(로컬의 master 브랜치에 원격 origin의 master 브랜치 가져옴)

git checkout master

git pull origin master

END