www.git-scm.com 에서 설치파일 다운로드

- 설치과정 중 [default branch name] 을 정하는 단계에서 가급적 [**main**]으로 정하는 것이 요즘 추세.
- HTTPS transport backend 설정 [**Use the OpenSSL library** (기본권장)]

[0] Git Bash최초 설정

1. 사용자 정보

Git을 설치하고 나서 가장 먼저 해야 하는 것은 사용자 이름과 이메일 주소를 설정하는 것이다.
Git은 커밋할 때마다 이 정보를 사용한다. 한 번 커밋한 후에는 정보를 변경할 수 없다:

$ git config --global user.name JohnDoe [띄어쓰기가 있는 이름를 하려면 따옴표 “John Doe”]
$ git config --global user.email johndoe@example.com

---

[1] Git저장소 만들기

1. 윈도우 탐색기로 프로젝트폴더 생성 (C:\Users\Admin\MBCA\GitRepository)

2. 명령프롬프트에서 로컬 저장소로 폴더 이동 (C:\Users\Admin\MBCA\GitRepository)

```
> cd MBCA          ( MBCA 디렉토리로 이동 )

> cd GitRepository     ( GitRepository 디렉토리로 이동 )

> dir ( 디렉토리안에 파일과 폴더들 리스트보여주는 명령어 )
```

3. C:\Users\Admin\MBCA\GitRepository 폴더를 Git저장소 만들기

프로젝트를 Git으로 관리하고 싶을 때, 프로젝트의 디렉토리로 이동해서 아래와 같은 명령을 실행한다.

```
> git init
```

- 이 명령은 .git이라는 하위 디렉토리를 만든다.(윈도우탐색기로 .git폴더존재확인 : 숨김문서보기)
  .git 디렉토리에는 저장소에 필요한 뼈대 파일(Skeleton)이 들어 있다

- initialized 코멘트가 출력될때 디렉토리 경로의 뒤에 (master or main)라고 표시되는 것을 볼 수 있음.
  - 현재 저장소에서 작업중인 브랜치가 master(or main)라는 것을 표시하고 있는 것임.

4. 첫번째 파일 만들어보기. (Vim편집기 사용 - git설치시에 같이 있음)

```
> vim hello.py
```

- 편집기화면모드가 되면 [i]키를 누르기(글 삽입[작성]모드)

```
message='nice to meet you'
print(message)
```

- 입력이 완료되었으면[Esc]키를 누르고 :wq를 통해 작성 종료
- 그냥 :q! 를 쓰면 저장없이 종료

- 저장이 잘 되어 있는지 문서 보기.
  1) cat hello.py
  2) 윈도우 탐색기 창에서 문서 열어보기

4. 문서작업이 완료되면 git의 소스코드 관리상태를 확인해보기

```
> git status
```

-- 아직 git에 의해 hello.py 문서의 작업내역이 git 시스템의 관리번호에 적용(staged라고 함)되지 않았기에
빨간 글씨로 hello.py 이 보여질 것임.

-- 주의깊게 글들을 보면 (use "git add <file>..." .. 이런식으로 해야할 명령어가 있으니 다음 작업을 예상할 때 참고..)

5. hello.py문서를 git시스템의 관리파일에 추가하기

```
> git add hello.py
```

-- 잘되면 The file will have its original line endings in your working directory. 메시지 보임.

(* Windows 에서는 line ending으로 CR(Carriage-Return, \r)과 LF(Line Feed, \n)을 사용하고 Unix 나 Mac OS 는 LF 만 사용)
vim이 unix 계열이고 hello.py는 윈도우에 저장되므로 .. LF로 된 줄바꿈이 CRLF로 변경되었다고 표시되는 경고문구 있음.

6. git관리시스템에 잘 추가 되었는지 다시한번 상태 보기

```
> git status
```

-- 잘되었으면.. 녹색으로 파일내역 보일것임.

-- 여기까지 하면 현재 저장상태까지 잘 작업되었다는 것이지만 아직 완전하게 작업을 완료한 것은 아님.

7. git을 통해 문서작업 완료(commit)하기 [현재까지의 작업 일단 완료표시]

```
> git commit
```

-- 편집기가 보이면서 커밋 메시지를 작성하도록 함 : [i]키를 누른 후 커밋 메시지 작성

First commit

-- 메세지 작성이 완료되었으면 [Esc]키를 누르고 :wq를 통해 저장후 작성 종료

-- 정상적으로 완료되었다는 메세지가 보일 것임.

---

[3] 브랜치**(Branch) -** 일종의 문서 복사본 만들기

(기존에 완료된 곳에서 이어서 작성하다가 잘못되면 기존 성공파일로 못 돌아가는 문제 해결**)**

0. 브랜치의 리스트 보기

```
>  git branch
```

-- 현재 브랜치가 녹색으로 표시됨.

1. 새로운 브랜치 만들기

```
> git branch hotfix
```

2. 브랜치가 만들어 졌는지 확인하기

```
> git branch
```

-- 현재 브랜치가 녹색으로 표시됨

3. 새로운 브랜치로 이동

```
> git checkout hotfix    or   git switch hotfix
```

-- hotfix 브랜치로 이동.. 경로를 보면 (hotfix)로 파란색 글씨가 변경되어 있음.
-- Switched to branch 'hotfix' 메시지 보임.

4. hotfix 브랜치에서 문서수정작업

```
> vim hello.py
```

-- [i]키를 누르고 새로운 글씨 추가 'have a good day'. 작성완료 후 [Esc]키 누르고 :wq로 저장 및 나가기

```
> cat hello.py
```

-- 수정된 내역 확인하기

5. 작업한 내용에 대한 git시스템의 관리상태 확인

```
> git status
```

-- 빨간 글씨가 보이면 형상관리 중이 아닌 파일이 있는 것임.

6. 작업내역(hello.py)을 git시스템에 추가하기

```
> git add hello.py
> git status      --확인해 보기 , 녹색글씨가 보이면 추가된것임.
```

7. 작업의 내용이 완료되었으면 확정명령(commit)하기

```
> git commit
```

-- [i]키 누르고 커밋 메시지 작성, [Esc]키 누르고 :wq로 작성완료

8. master브랜치와 hotfix브랜치의 차이 확인하기

1) hotfix브랜치에서 hello.py문서 확인

```
> cat hello.py
```

2) master브랜치로 이동(checkout)후 hello.py문서 확인

```
> git checkout master
```

```
> cat hello.py
```

3) 실제 브랜치를 변경했을 당시의 윈도우 탐색기에서 hello.py 문서를 메모장으로 확인해보기

(문서는 1개인데 브랜치마다 다르게 보임)

9. master와 hotfix의 작업내용을 병합(merge)하기

1) master브랜치에서 hotfix 병합(merge)

```
> git merge hotfix
```

-- [i]키를 누르고 병합 코멘트 작성, [Esc]키를 누르고 :wq로 나오기

2) 병합내역 확인

```
> cat hello.py
```

** 새로운 작업은 다시 hotfix브랜치에서...
** master는 언제나 최종 완성본들만.....
*** 참고로 git 삭제는

```
> git branch -d hotfix
```

---

[4] 작업내용 되돌리기**(Roll Back)**

1. hotfix브랜치로 이동

```
> git checkout hotfix
```

2. hello.py 수정

```
> vim hello.py
```

-- [i]키 누르고 새로운 내용 추가.. [Esc]키 누르고 :wq로 나오기

3. 작업내용 git시스템 관리상태 확인 및 Roll Back 힌트 확인하기

```
> git status
```

-- 빨간글씨 위헤 (use "git checkout -- <file>.." 이라는 명령이 작업내역을 되돌리는 명령어)

4. 되돌리기 및 확인

```
> git checkout -- Hello.java
> cat Hello.java
```

5. 혹시 git시스템 관리 추가까지 했다면.??

1) hello.py 새로 수정. `> vim hello.py`

2) 깃시스템 관리 추가 `> git add hello.py`

3) 관리 상태 확인 및 힌트 `> git status`

-- (use "git restore –staged <file>..."이라는 명령이 add한 내역을 되돌리는 명령)

4) add 이전 상태로 되돌리기 및 상태 확인

```
> git restore –staged hello.py

> git status
```

6. 혹시 커밋까지 했다면??

```
> git reset --hard HEAD~1
```

HEAD is now at 1f58cad additional print "aaa" Merge branch 'hotfix'

(hotfix브랜치는 이제 이전 commit위치로 이동)

#새로운 폴더 만들기 —--------
mkdir 폴더명

각 프로젝트 폴더마다 git init 하는 것이 좋음.
- 상위폴더에 git을 하면 하위 프로젝트 전체에 대해 git관리를 해주기에 git status가 항상 다른 폴더도 표시됨.

git 저장소 해제하기!!!
-- 해당 폴더에서 .git폴더 삭제
---

[5] 다른 **git** 저장소를 복제해 오기

1. 새로운 폴더 생성 (C:\Users\Admin\MBCA\GitRepository2)

2. GitRepository2 폴더를 git 저장소로 초기 설정하기

```
> git init
```

3. 로컬 저장소 복제(clone)

```
> git clone /로컬/저장소/경로
> (예) git clone C:\Users\Admin\MBCA\Nice     - Nice 라는 폴더를 복제
```

4. 원격 서버의 저장소를 복제하려면 아래 명령을 실행하세요.

```
> git clone 원격/저장소/경로.git
```

---

[6] GitHub 원격저장소

작업내역을 오로지 로컬**PC**에만 있으면 다른 **PC**에서는 작업을 못하므로 온라인 저장소 **PC**라고 보면됨
(일종의 클라우드 서버)

www.github.com 회원가입 및 이메일 인증

1. First 프로젝트 저장소 만들기 // README.md도 같이 만들기

2. [create new file] 버튼을 통해 hello.py 문서 생성 및 commit

-- GitHub사이트는 기본적으로 작업할때마다 commit하도록 되어있어서 바로바로 관리(add)및 완료(commit)이 동시에 됨

3. [Branch: master or main] 버튼을 통해 새로운 hotfix 브랜치 추가 및 이동(checkout)

4. hotfix브랜치에서 hell.py를 선택하고 [연필]모양의 아이콘으로 문서 수정 및 커밋

5. 커밋 후 master브랜치와 hotfix브랜치를 이동하면서 hello.py 문서가 서로 다름을 확인

6. 병합(merge)하기
1) master브랜치로 이동(병합당하는 브랜치)
2) [Compare & pull request]라는 녹색버튼을 통해 branch비교 및 병합의 위해 가져오기 요청
3) hotfix의 commit지점 메세지가 보여지며 아래쪽에는 바뀌어진 내역이 비교되어 표시됨
-- 녹색음영으로 + 아이콘이 있는 것이 추가된것
-- 적색음영으로 - 된것이 제거된 것

4) [Create pull request] 녹색버튼으로 병합의 위해 가져오기 요청
5) 요청내역이 보이면서 녹색박스 안에 녹색버튼으로 [Merge pull request]버튼 으로 병합[merge]하기
6) Merge코멘트 작성 후 [Confirm merge] 녹색버튼 클릭
녹색들이 보라색으로 바뀌면 merge성공

7) 저장소로 이동하여 branch를 이동하면서 hello.py 문서 확인

[기타 **GitHub** 관련]

문서선택 후 [History]버튼 클릭하면 이전 Commit내역들을 볼 수 있으며 그중에 원하는 위치의 index번호를 선택하면 해당 commit당시의 문서를 볼 수 있음.

Gihub는 롤백기능을 찾을 수 없음.

Github에서 다른 사람의 주소가 있으면 문서 볼수 있음. -- 수정은 불가능 ---

수정하려면 [Fork]로 가져와서(clone) 내 저장소에 넣고 수정하기

[Upload files]버튼을 통해 파일업로드 가능

README.md 파일은 html문법에 맞게 작성하면 화면 꾸밀 수 있음.

contributor : 기여자

로컬 저장소**(**본인 **PC)**에서 원격 저장소**(GitHub)**에 파일들을 업로드 하는 방법

```
git init
git add -all
git commit -m "first commit"

git remote add origin https://github.com/mbca-aix/test.git
git push -u origin main
```

---

[7] Github Team 프로젝트

7.1 Collaborate : 하나의 원격저장소에서 여러 팀원이 같이 작업

1. 팀장이 저장소 하나 생성 (FirstTeam)
2. Setting >> Collaborators 메뉴를 선택 --> 협업하고자 하는 팀원의 email또는 github 아이디 검색
   -- [Add collaborator]선택 하면 해당 팀원의 email에 초청메일(invite) 발송
3. 팀원은 해당 메일을 열어 안에 있는 [accept invite] 녹색버튼 클릭

각 팀원들은 각자 같은 저장소에 대한 작업권한이 있음.

가급적 master를 직접 건드리는 방식보다는 branch를 통해 각자 작업 후 merge하는 방식 권장

각 branch에서 [New pull request]버튼을 눌러보면 화살표로 어떤 브랜치에서 어느브랜치로 병합되는지 표시됨

-- 만약 두 branch사이에 차이가 없다면 anything to compare 문구가 보여짐.

7.2 Contribute : 팀원 각자의 원격저장소에서 중앙 원격저장소에 지속적으로 업데이트

- PR 보내기
