



191125 Day 76

2019년 11월 25일 월요일 09:07

 : 주요 용어
 : 명령어

<VCS (Version Control System)>

버전 관리 시스템

프로젝트를 진행할 때, 각 버전 별로 관리하기 위한 시스템

<SVN (Subversion)>

CVCS (Centralized Version Control System) : 중앙 집중 버전 관리 시스템

하나의 중앙 서버로 모든 내용이 저장됨

중앙 서버가 다운되면 모든 내용이 날라가서 더 이상 작업이 불가능

<Git>

DVCS (Distributed Version Control System) : 분산 버전 관리 시스템

각 사용자의 서버로 내용이 분산 저장됨

중앙 서버가 다운돼도 모든 내용이 로컬 서버에 남아있어서 계속해서 작업 가능

Git이 SVN보다 기능 면에서 효과적이고, 팀원 간 소스코드 공유도 더 편리함

꼭 팀으로서 사용해야하는 것이 아니라 개인 프로젝트 목적으로 사용해도 됨

Working Tree : 프로젝트가 저장돼있는 폴더 (실제로 작업하는 공간)

Repository : Git이 제공하는 원격 저장소

Local Repository : Git이 제공하는 지역 저장소 (사용자 컴퓨터에 데이터를 저장해서 보관)

index : Working Tree와 Local Repository 중간에 있는 곳

[프로젝트의 데이터 수정 및 저장 과정]

프로젝트 폴더가 수정되면 Git은 그것을 인지함 (어떤 작업을 바로 수행하지는 않음)

-> 사용자가 Git에게 수정 내용을 index에 올려달라는 요청을 함 (**Staging** 과정)

-> index에 수정 내용이 들어감 (이 과정이 없으면 내용이 날라갈 수 있음)

-> index에 들어간 수정 내용을 Local Repository에 저장해달라는 요청을 함

-> Local Repository에 수정 내용이 저장되고 index는 비워짐 (**첫 번째 버전 생성**)

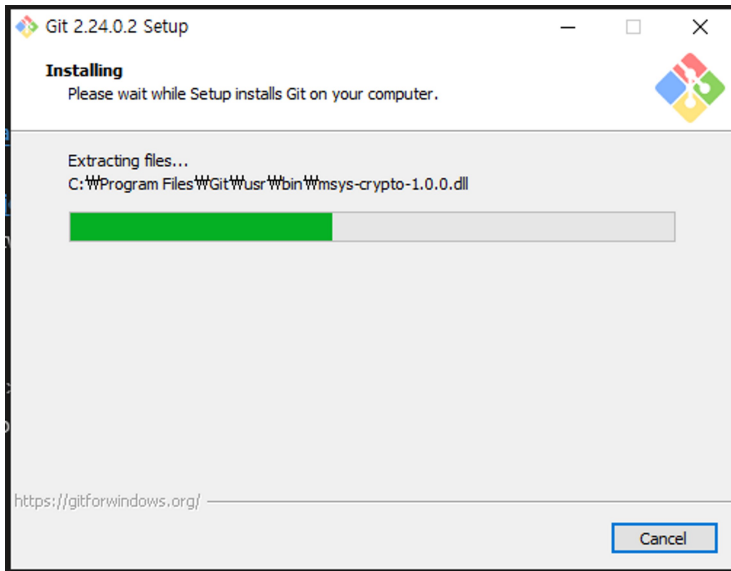
위 내용이 반복되면서 버전이 업데이트 됨

사용자는 아무 때나 원하는 버전을 사용할 수 있음

<Git SCM>

콘솔용 Git

2.24.0.2 - (Major Version).(Minor Version).(Update).(?)



<CMD (명령 프롬프트) 명령어>

prompt : 명령줄

dir (directory) : 현재 폴더에 있는 파일 및 폴더 목록

cd (change directory) : 폴더 위치를 변경

cd.. : 한 단계 위로 이동

cd 폴더명 : 현재 폴더에 있는 해당 폴더로 이동

cd W : 최상위 폴더로 이동

d : d 드라이브로 이동

폴더 이름 일부만 작성하고 tab 누르면 자동완성됨

<CMD에서 사용하는 Git 명령어>

git init(initialize) : git이 해당 폴더를 감시하도록 설정(Git이 동작하도록 하는 초기화 설정) -> .git이라는 숨김폴더가 생성됨

D:\디지탈컨버전스 8월\temp>git init

Initialized empty Git repository in D:/디지탈컨버전스 8월/temp/.git/

=====

공유되면 안되는 파일 및 폴더(설정 관련 내용)를 알려주기 위해 .gitignore.txt 라는 텍스트 문서를 만들어야 함

.gitignore.txt에 작성되는 내용중 #은 주석임

작성 후 .txt 확장자를 지워줘야 함

<https://www.gitignore.io/> 접속 (사용자들의 경험상 공유되면 안되는 내용을 알려주는 사이트)

-> java, eclipse, java-web 입력 후 생성 클릭

-> 해당 내용 복사해서 .gitignore.txt에 붙여넣기

-> *.MF (공유되면 안되는 설정 파일) 추가 작성

=====

git status : Git의 상태 확인

git add <파일명> : 파일을 index에 staging 시켜주는 명령어

git rm --cached <파일명> : 파일을 index에 staging 시킨 것을 취소하는 명령어

git config user.email "메일 주소" : 자신이 누구인지 알리기 위한 메일 작성 (진짜 메일이 아니어도 됨, 형식적인 과정)

git config user.name "이름" : 자신이 누구인지 알리기 위한 이름 작성 (진짜 이름이 아니어도 됨, 형식적인 과정)

git commit -m "메시지" (m의 의미 : message or comment) : index에 staging된 파일을 local repository에 저장하는 명령어 (새 버전 생성)

git log : repository에 저장된 commit-hash(commit point id), 사용자, 날짜, 메시지를 보여줌

```
D:\#디지탈컨버전스 8월#temp>git log
commit 900019791990f723065e9e6b19e58f12dc9c841f (HEAD -> master)
Author: sh <sp077@naver.com>
Date: Mon Nov 25 10:36:00 2019 +0900

    First Commit with gitignore
```

`git checkout <commit-hash>` : 해당 버전으로 이동 -> 새로운 임시 branch(분기점) 생성

```
D:\#디지탈컨버전스 8월#temp>git checkout 2eecd6
Note: switching to '2eecd6'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-c` with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to false

HEAD is now at 2eecd69 test01 is added

```
D:\#디지탈컨버전스 8월#temp>git status
HEAD detached at 2eecd69
nothing to commit, working tree clean
```

```
D:\#디지탈컨버전스 8월#temp>git log
commit 2eecd69ec32821a3470eebbfe25c63ba1c437584 (HEAD)
Author: sanghyun <sp077@naver.com>
Date: Mon Nov 25 14:21:11 2019 +0900

    test01 is added
```

test01 is added

```
commit d6b81144fc72188160dd052a2fb64287d18a1c96
Author: sanghyun <sp077@naver.com>
Date: Mon Nov 25 12:47:11 2019 +0900

    .gitignore.txt is added
```

HEAD detached at <commit-hash> : 해당 버전에서 새로운 임시 branch를 생성했다는 의미

HEAD : 현재 어떤 branch의 최신 버전을 가리키고 있는지 알려줌

가장 처음의 branch는 master branch라고 함 (**HEAD -> master**는 master branch의 최신버전을 가리키고 있다는 의미)

`git branch --list` : branch의 목록 조회

`git checkout master` : master branch로 이동 (임시 sub branch는 사라짐)

sub branch를 실제로 생성하여 사용하려면 이름을 부여해야함

branch를 사용하지 않는 방법도 있음 (Git 초보 사용자에게 권장)

[파일 복구 방법 01]

`git restore <파일명>` : commit 수행 전에 삭제된 파일 복구

commit 수행 후, 이전 버전에서 삭제된 파일을 살리고 싶으면 그 파일이 있던 버전으로 이동 (새로운 임시 branch 생성)
-> 해당 파일을 다른 폴더에 복사 -> master branch로 다시 이동 -> 복사한 파일을 다시 프로젝트에 추가

[파일 복구 방법 02]

`git checkout <commit-hash> <파일명>` : 이전 버전에서 삭제된 파일을 master branch상에서 한 번에 복구

[파일 복구 방법 03]

Local Repository가 삭제됐을 경우엔 Online Server Repository로부터 복구 가능 (GitHub 사용)

`git remote add <저장소 이름 (임의로 설정)> <저장소 url (GitHub의 repository url)>` : 원격 저장소를 추가

`git remote -v` : 현재 등록된 원격 저장소 목록

[로컬 저장소에서 원격 저장소로 보내는 과정]

Working Tree에서 Index로 **add** (staging 과정)

-> Index에서 Local Repository로 **commit**

-> Local Repository에서 Server Repository로 push

push : Local Repository -> Server Repository

pull : Server Repository -> Local Repository

git push <원격 저장소 이름> <local branch>:<remote branch> : 로컬 저장소에 있는 branch를 서버 저장소의 branch로 전송

git pull <원격 저장소 이름> <local branch>:<remote branch> : 서버 저장소에 있는 branch를 로컬 저장소의 branch로 전송

push는 자격 정보(로그인)가 필요하고, pull은 누구나 언제든지 가져올 수 있기 때문에 자격 정보는 필요 없음

git reset --hard <commit-hash> : 해당 지점 이후의 history를 지우고 해당 지점으로 이동 (commit point id가 삭제되지는 않음)

=====
collaborating : 다른 사용자들에게 repository에 대한 동등한 권한을 부여함

- > 서로 다른 사용자가 같은 파일을 push하면 conflict (충돌) 발생
- > server는 가장 먼저 push한 사용자의 파일만 승인해줌
- > 다른 사용자들에게는 conflict 발생 메시지를 보냄
- > 다른 사용자들은 우선 pull로 최신 버전을 받은 후에 수정해서 다시 push 해야함
- > 여러 사용자들이 동시에 같은 파일에 대한 작업을 하면 매우 복잡해짐

pull request : 다른 사용자들에게 repository를 fork해줌 (복사본을 보냄)

- > 사용자들은 각각 복사본으로 작업을 수행하고 소유자에게 pull request를 보냄 (본인이 수정한 내용을 pull 해달라는 요청)
- > 소유자는 내용을 검토한 후 수정 내용을 pull 함
- > 소유자가 자리를 비운 경우에 request가 계속해서 밀릴 수 있음
- > 팀원들은 자신의 repository만 갖고 있는 것이 아니라 팀장의 repository도 갖고 있어야 함 (다른 팀원들이 수행한 업데이트를 계속 받아오기 위해)