Git(1)

2014년02월10일 이훈구

-목 차 -

KSNET
케이에스넷

1.개요

(1) 개요

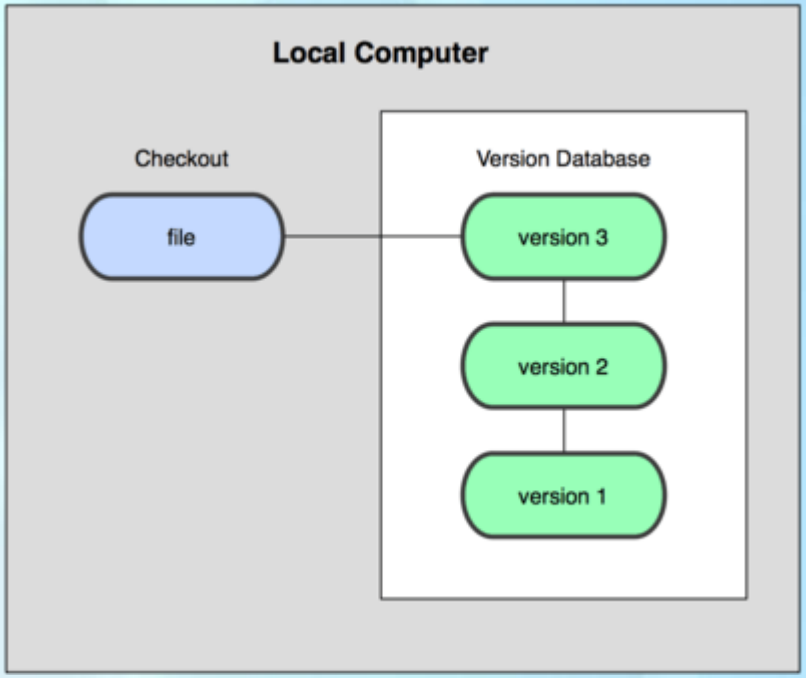
- 버전관리 시스템은 소스코드의 중요한 변화들을 기록하는 행위
- 어떤 문제가 발생했을 때 문제의 맥락을 파악 할 수 있도록 지원
- 변화에 실패 했을 때 과거의 상태로 쉽게 전환 가능
- 실패에 대한 부담을 덜어지고 좀 더 자신있는 개발을 촉진하는 효과
- 버전관리는 백업, 협업과 같은 중대한 장점을 제공
- 문서나 코드의 변경사항을 저장해서 과거의 상태를 열람, 복원
- 협업시에 협업자들의 변경사항을 자동으로 붙여주고, 충돌을 방지

(2) 역할

- 프로그램의 변경 이력을 관리하는 소프트웨어
- 작업일지
- 협업
- 백업

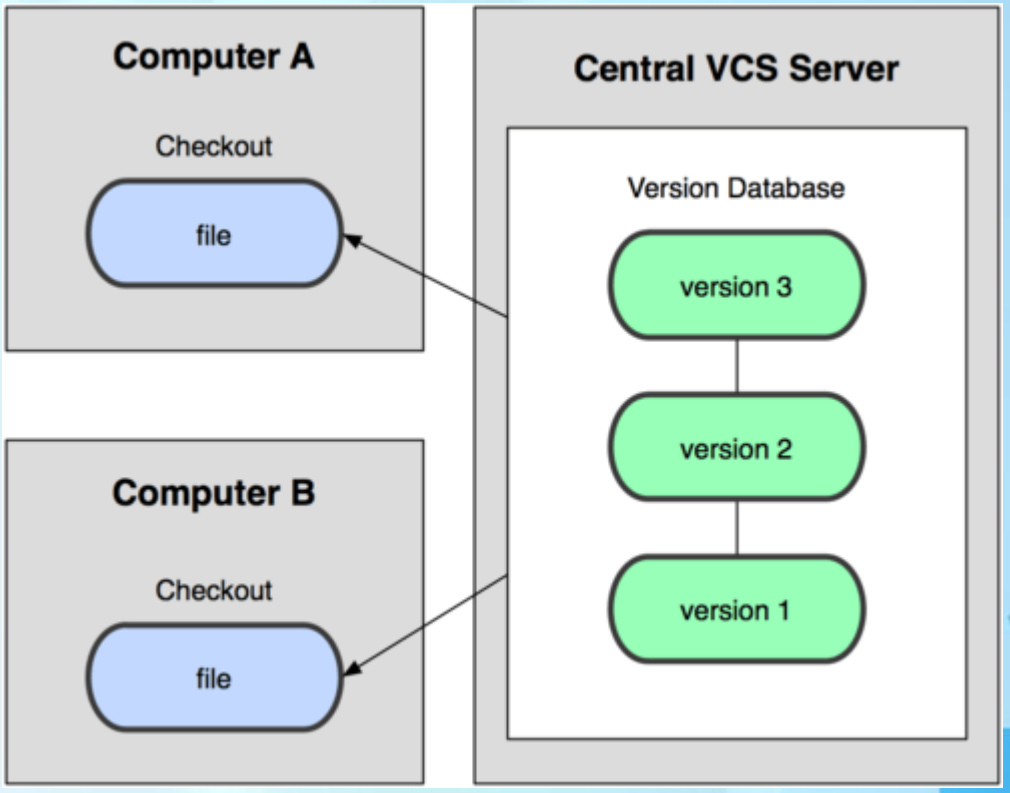
2.버전관리

(1) 로컬버전관리시스템



2.버전관리

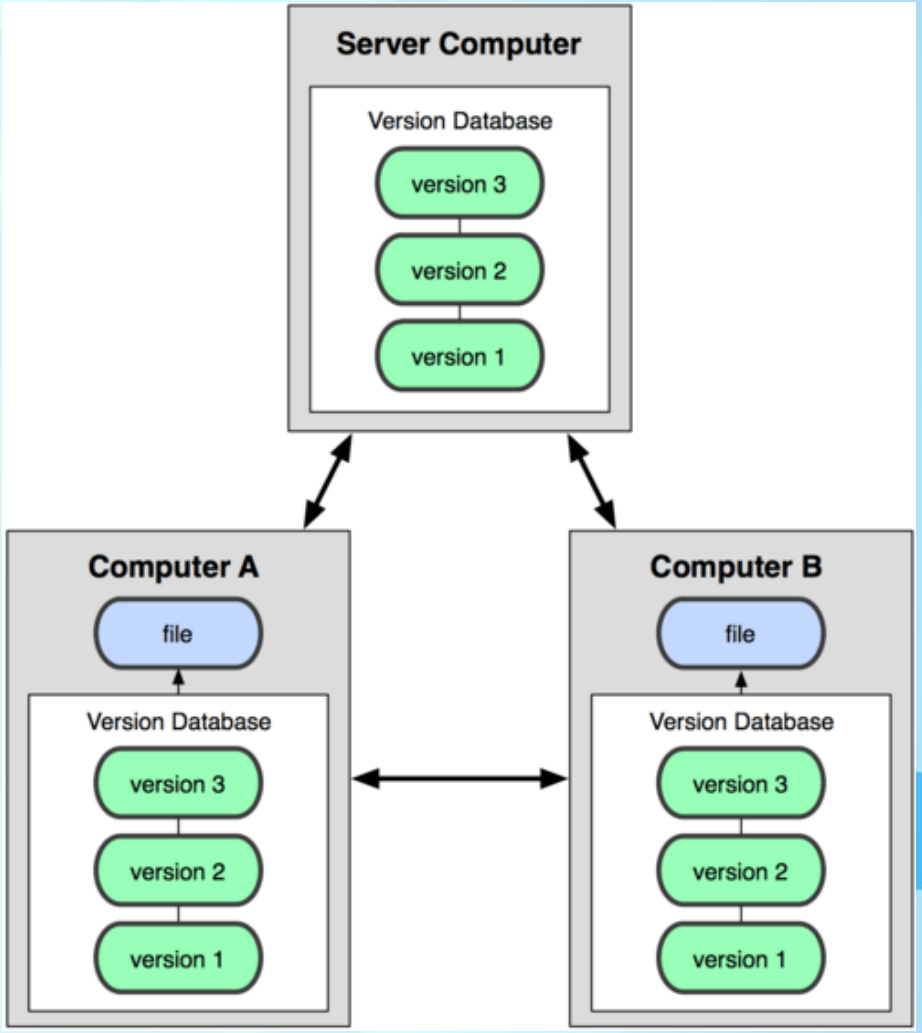
(2) 중앙집중식버전관리시스템(CVCS: Center Version Control System)



※ CVS, Subversion, Perforce

2.버전관리

(3) 분산버전관리시스템(DVCS: Distributed Version Control System)



※ Git, Mecurial, Bazaar, Darcs

3.Git

(1) 역사

리눅스커널관리에서 시작

2002년 BitKeeper(DVCS) 사용 시작

2005년 BitKeeper 결별

2005년 리눅스창시자 리누스 토발즈 개발

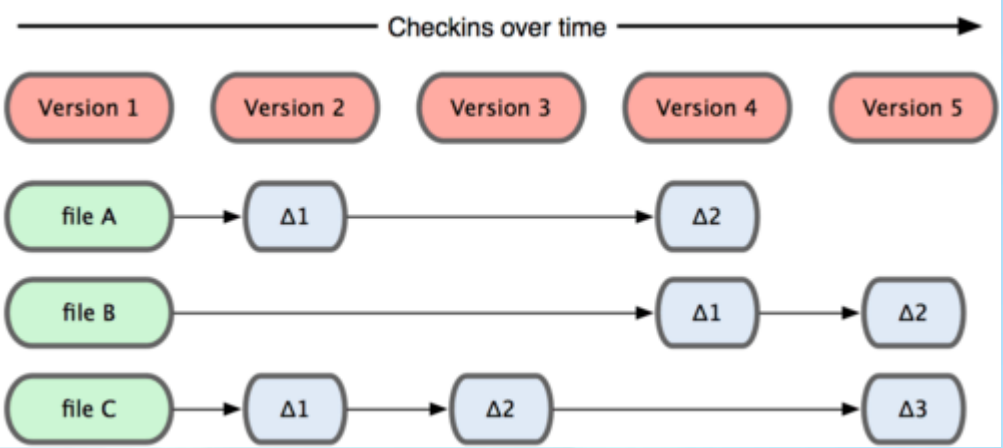
(2) 주요기능

- 빠른 속도
- 단순한 구조
- 비선형적인 개발(수천 개의 동시 다발적인 브랜치)
- 완벽한 분산
- 리눅스 커널 같은 대형 프로젝트에도 유용할 것(속도나 데이터 크기 면에서)

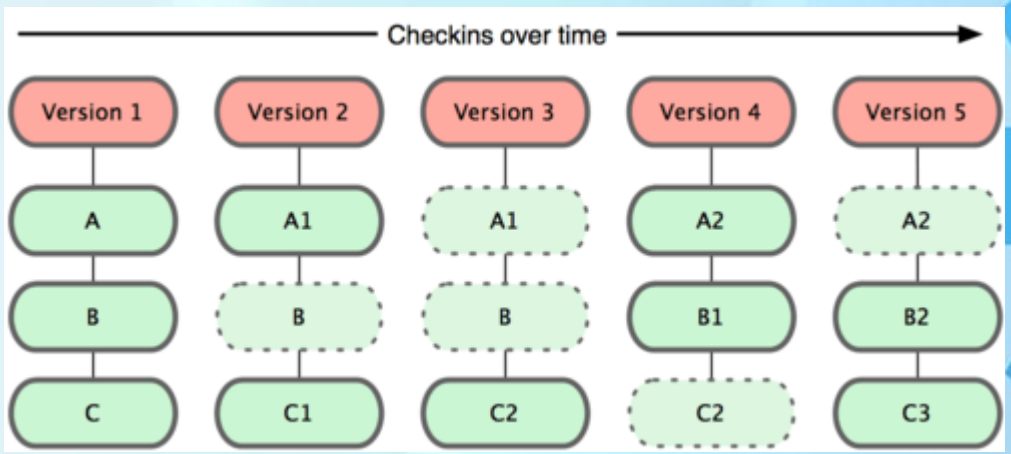
3.Git

(3) 버전관리방식

- CVS, Subversion, Preforce, Bazaar : 델타 방식



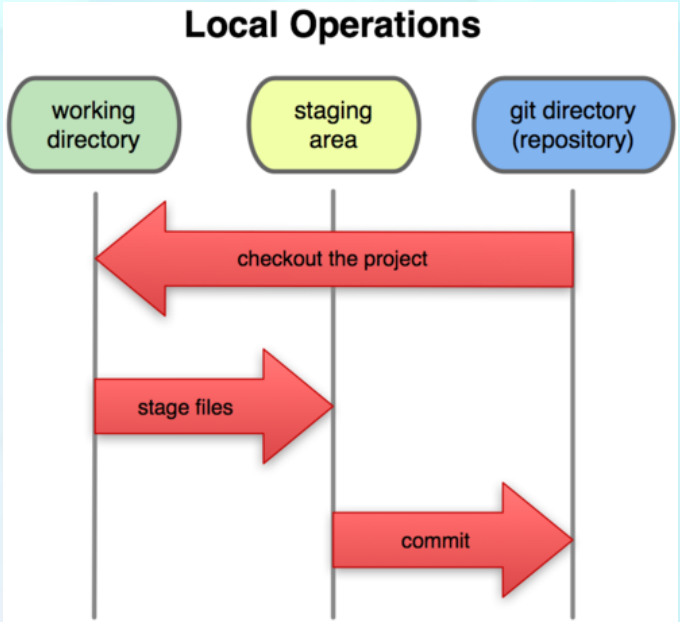
- Git : 시간 단위로 스냅샷을 저장

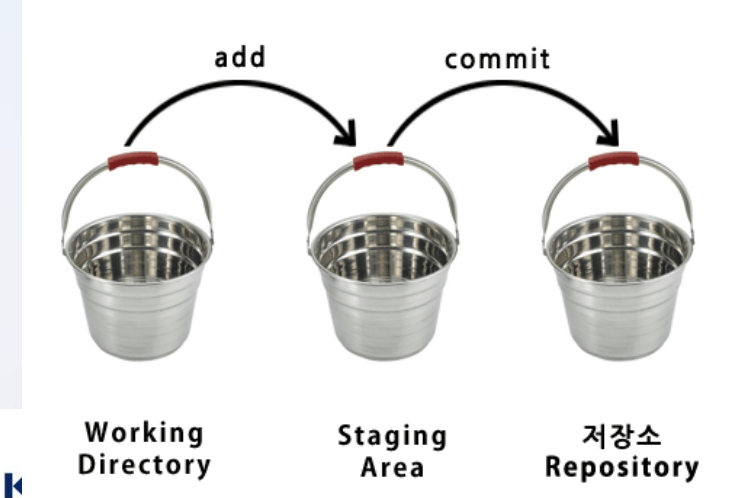


KSNET
케이에스넷

3.Git

(4) 흐름



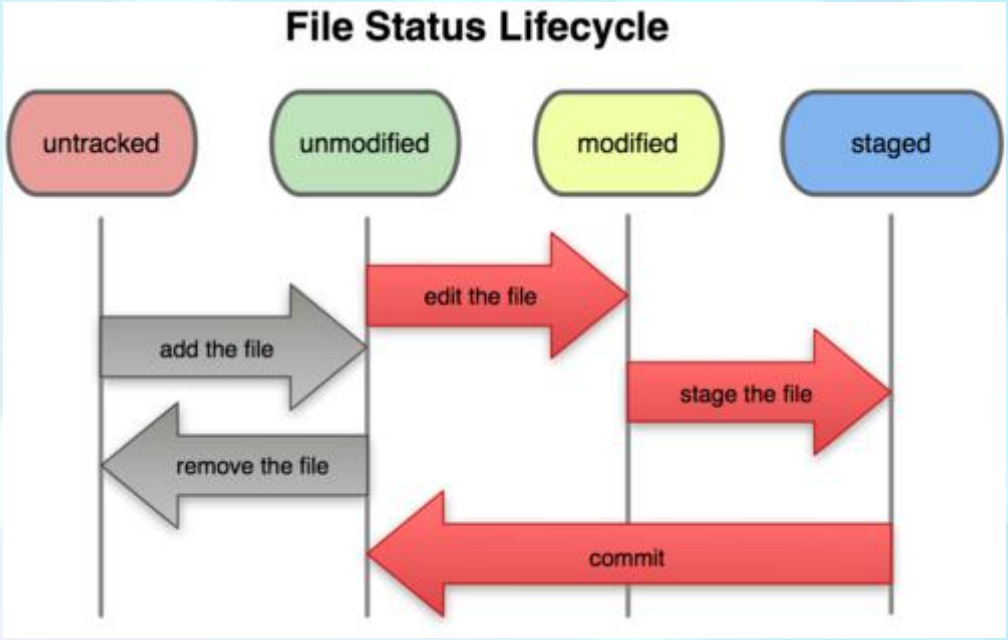


- Working directory
  - 개발 작업 디렉토리
- Staging Area
  - Commit 을 하기 위한 작업 공간
- Git Directory(repository)
  - Commit후 버전관리를 위한 공간

케이에스넷

3.Git

(5) File Status Lifecycle



- Untracked
  - 파일
- Unmodified
  - 최신상태
- Modified
  - 수정된 상태
- Staged
  - 수정 후 Commit 되기 전 상태

4.설치

- 다운로드 :
  http://www.git-scm.com
  http://msysgit.github.io

- 설정
  git config --global user.name “Peter LEE"
  git config --global user.email foxworld@gmail.com

- 참고문서(Pro Git)
  http://www.git-scm.com/book

KSNET
케이에스넷

5.시작

(1) 기본기능

- Git Bash 클릭한다. (Git Bash는 Git의 콘솔화면)
- 작업 디렉토리(working directory)를 만든다.

```
$ git init
```

- 작업할 파일을 생성 한다

```
$ touch helloworld.html    or    $ vi helloworld.html
```

- 파일을 스테이징 영역(staging area)에 추가(add)한다.

```
$ git add helloworld.html
```

- 커밋(commit)한다.

```
$ git commit –m 'first commit'
```

- 파일의 상태 확인

```
$ git status
```

6.History 조회

(1) History 조회

$ git **log**
: 최신순으로 커밋 로그를 조회

$ git **log -p -2**
: 최신순 커밋 2개를 보여주면서 각 커밋 간의 차이점을 diff로 조회

$ git log **--pretty=format:"%h %s" --graph**
: 커밋간의 변경점을 그래프로 조회

$ git log **--pretty=oneline**
: 로그를 한줄로 조회

$ git log **--since=2.weeks**
: 지난 2주간의 커밋을 조회

$ git log **--path1 path2**
: path1, path2의 커밋이력을 조회

$ gitk
: 커밋이력을 GUI 응용프로그램으로 조회

6.History 조회

(2) Commit의 History 조회

$ git log **-p HEAD..FETCH_HEAD**

: fetch 명령을 실행한 후에 사용.
fetch를 하면 실제 로컬 저장소에는 변경사항이 반영되지 않음
로컬저장소에 원격저장소의 변경사항을 반영하기 전에 이 명령을 실행해서 원격 저장소와 로컬 저장소 사이의 차이점을 비교해보고 문제가 없다면 merge.

- 옵션

-(n) 최근 n 개의 커밋만 조회
--since, --after 명시한 날짜 이후의 커밋만 검색
--until, --before 명시한 날짜 이전의 커밋만 조회
--author 입력한 저자의 커밋만 조회
--committer 입력한 커미터의 커밋만 조회

7.복원

(1) 복원

- revert
  : 이미 commit된 상태를 특정한 시점으로 복원한다. 복원된 내용을 새로운 커밋으로 발행함
- reset
  : 스테이징이나 커밋을 취소할 때 사용됨, 커밋되지 않음.
- checkout
  : 브랜치를 변경하고, 특정 브랜치의 내용으로 현재 브랜치의 파일을 변경함

7.복원

(2) 명령어

구분	명령어	working directory	staging area	repository directory	기타
파일	checkout -- 파일	취소	유지	유지	
	checkout HEAD 파일	취소	취소	유지	
	reset -- 파일	유지	취소	유지	
전체	reset commit id	유지	취소	commit id 이후의 커밋 취소	commit id에 해당하는 커밋은 유지된다.
	reset HEAD^	유지	취소	최신커밋 취소	커밋은 했지만 push하지 않은 경우 유용 HEAD^는 최신커밋을 포함 두개의 커밋을 의미
	reset HEAD~2	유지	취소	최신커밋 -2번 취소	
	reset --hard HEAD~2	취소	취소	최신커밋 -2개 취소	-- hard는 working과 staging 모두 취소
	reset --soft HEAD~2	유지	유지	최신커밋 -2개 취소	--soft는 working과 staging 모두 유지
	reset --hard ORIG_HEAD			병합한 커밋을 취소	ORIG_HEAD는 위험한 작업에 대한 포인터로 push나 merge가 여기에 해당됨
	revert HEAD	거부	거부	최신 커밋 취소	커밋을 이미 push한 경우
	reset --hard HEAD	취소	취소	유지	신규파일에 영향없음
	checkout HEAD	취소	취소	유지	신규파일에 영향없음
	checkout -f	취소	취소	유지	신규파일에 영향없음
	clean -f -d	untracked	유지	유지	Untracked인 파일제거(디렉토리포함)

7.복원

(3) reset의 옵션

--soft : staging area(index) 보존, working directory 보존. 즉 모두 보존.

--mixed : staging area 취소, working directory 보존 (기본 옵션)

--hard : staging area 취소, working directory 취소. 즉 모두 취소.

(4) reset과 revert 차이점

- reset은 해당 커밋의 상태로 되돌리는 명령
- reset 뒤에 커밋 아이디를 지정하면 해당 커밋 이후의 변경점이 취소처리
- revert는 선택한 커밋이 취소되는 것이 아니라, 이전 커밋이 추가

8.작업의 분기 Branch

(1) Branch란?

- 개발과정에서 분기가 필요할때 사용
- 메인개발이 아닌 새로운 아이디어가 생겨서 메인개발을 수정하지 않고 추가로 생성해서 개발하기 위해 사용

(2) branch 명령어

$ git **branch** 브랜치명
: git branch change_font

$ git branch
: branch 종료 확인

$ git **checkout change_font**
: 현재 branch 에서 'change_font' branch 로 이동

※ 동일한 개발 작업을 하면 됨

8.작업의분기 Branch

(3) merge 명령어

$ git **checkout master**
: 현재 'change_font' 에서 master branch 로 이동

$ git **merge change_font**
: change_font branch 와 master branch를 병합하여 master에 적용

※ git 이 판단한 소스코드 동일 또는 위치가 같아서 병합이 이루어지지 않은 수 부분을 찾아서 수정한 후에 add , commit 을 하여야 함

$ git **branch –D change_font**
: branch가 필요없거나 적용할 수 없을시 삭제

9.중요한 작업의 저장(tag)

(1) tag란

- 프로젝트에서 중요한 시점들을 기록해두는 방법
- 보통 릴리지 또는 배포할 때 사용

(2) tag 종류

- Lightweight tag : 커밋 ID만을 보존하는 간단한 태그
- Annotated Tag : Tag를 만든 사람의 이름, 이메일, 날짜, 메시지등을 저장

9.중요한 작업의 저장(tag)

(3) 명령어

```
$ git tag
   : Tag 리스트를 보여줌
$ git tag -l 'v1.4.2.*'
   : 'v1.4.2'로 시작하는 태그들의 리스트를 조회
$ git tag v1.1
   : 현재의 HEAD를 태그 v1.1로 저장
$ git tag v1.2 9fceb02
   : 커밋 아이디 9fceb02를 v1.2 태그로 지정
$ git tag -a v1.2
   : annotated tag로 v1.2태그를 생성
$ git show v1.2
   : v1.2 태그에 대한 상세한 정보 조회
```

```
$ git push origin v1.5
   : git은 태그를 push할 때 자동으로 리모트로 저장않음 꼭 명시해야 함
$ git push origin --tags
  : 모든 태그를 원격저장소에 반영
```

10.협업의 참여

(1) 협업이란?

여러사람이 하나의 프로젝트를 개발하면서 발생할 수 있는 다양한 충돌상황에 대한 질서와 규범을 버전관리 시스템이 제공

(2) remote branch

리모트 저장소에 있는 브랜치

리모트 저장소는 협업에 참여하고 있는 작업자들이 접속할 수 있는 중립된 저장소

명령규칙은 (remote)/(branch) 형식

(3) Remote Server

https://github.com

https://www.gitlab.com

10.협업의 참여

(4) Github 설정

- github id 생성
- SSH keys 생성

```
$ ssh-keygen
```

- id_rsa.pub의 공개키를 github에 등록

(5) Github new repository 생성

- Repository name 입력
- public or private 선택(private 설정 시 월 일정금액 결제 필요)
- Initialize this repository with a README 체크(README.md 파일 생성)
- .gitignore 선택(버전관리 필요없는 파일 설정)
- Create Repository 클릭

10.협업의 참여

(6) 명령어

```
$ git clone git@github.com:foxworld/helloworld.git
   : Github

$ git pull origin master
  : Github

$ git push origin master
  : Github

$ git remote add origin git@github.com:foxworld/helloworld.git
  :                URL

$ git remote rm origin
  :                URL
```

※ origin

- origin은 리모트 저장소의 별명
- clone 실행시 자동으로 origin 으로 설정
- 리모트 저장소를 만들지 않았을 경우 add 로 설정하면 됨

10.협업의 참여

(7) 명령어(동기화)

$ git **fatch origin master**
: Github
: Repository 까지만 적용

$ git **pull origin master**
: Github
: merge
: Working directory 적용

$ git **push origin master**
: Github

10.협업의 참여

(8) 명령어(리모트브랜치)

$ git **checkout –b** 신규브랜치명
: 신규브랜치명으로 생성

$ git **push origin** 브랜치명
: Github

$ git **push origin :**브랜치명
: 리모트 브랜치 삭제

※ master

- master 는 최초 branch 명으로 자동 생성
- 기타 branch 생성시 직접 명을 만들면 됨

Q & A

감사합니다.

Thank you