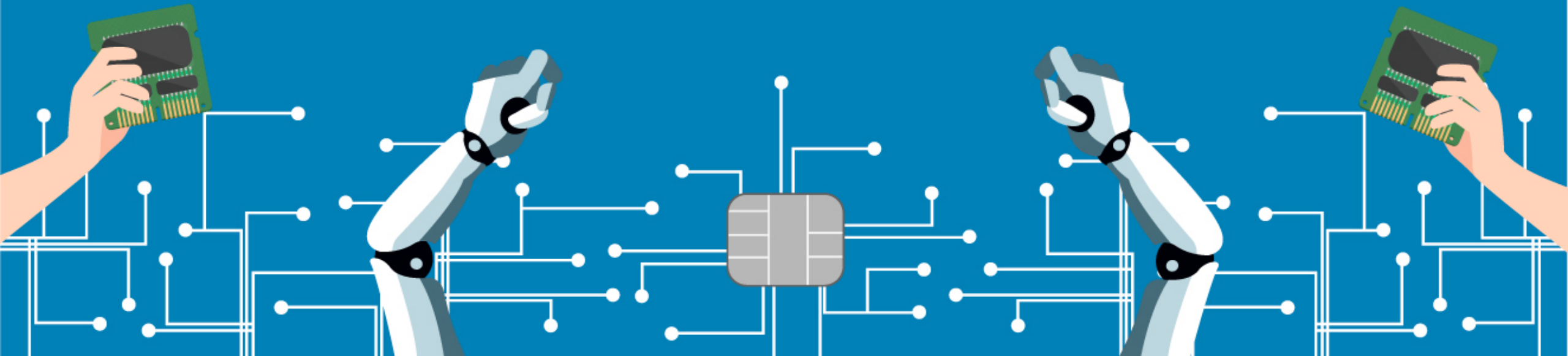
Kuggle

2020_02 Kuggle 정규세션_W2

2020.09.15
발표자: 정성준



CONTENTS

01 깃 시작하기

깃
깃 설치하기
리눅스 명령어

02 버전 관리

깃 저장소 만들기
버전 만들기
커밋 내용 확인하기
버전 단계별 파일 상태 확인
작업 되돌리기
브랜치

03 백업

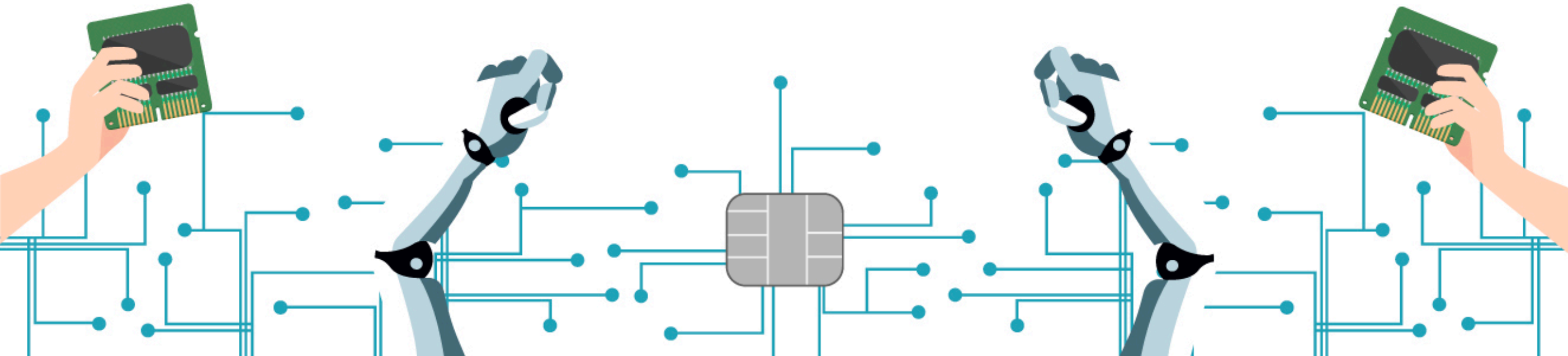
깃허브
원격저장소 연결
원격저장소 사용
SSH접속

04 협업

원격저장소 함께 이용
원격 브랜치 정보 확인
협업 기본
협업 브랜치 사용

05 깃허브 활용

오픈소스
블로그 만들기



CONTENTS

01. 깃 시작하기

- 깃
- 깃 설치하기
- 리눅스 명령어

01 깃 시작하기

깃

- git은 linux를 개발하는 과정에서 생긴 문서관리자이다.
- git의 핵심기능은 버전관리(version control), 백업(backup), 협업(collaboration) 세가지로 나뉜다

01 깃 시작하기

깃

1. 버전관리
   - 컴퓨터로 문서작성시 '다른이름으로 저장' 하며 '초안', '수정', '최종','진짜최종...' 등으로 문서 이름을 더럽히지 않고 문서버전관리를 가능하다
   - 깃은 문서를 수정할 때마다 언제 수정했는지, 어떤 것을 변경했는지 편하고 구체적으로 기록하기 위한 버전 관리 시스템이다.
2. 백업하기
   - 깃허브(github)등 깃의 원격저장소에 깃 파일을 백업할 수 있다.
   - 원격저장소의 종류는 여러가지..
3. 협업하기
   - 깃허브와 같은 원격저장소를 이용해 팀원들간 협업이 가능하다.

01 깃 시작하기

깃

- version control -> backup -> collaboration순으로 배워야 이해가 쉽다.
- 깃을 이용할 수 있는 깃 프로그램은 여러 종류가 있다(예: github desktop, tortoisegit-window, source tree 등)
- 커맨드 라인 인터페이스(Command Line Interface)는 리눅스 명령어에 익숙해야 하기 때문에 GUI를 사용하는 깃프로그램보다 사용하기 어렵지만 익숙해지면 다양한 기능을 사용할 수 있기 때문에 개발자 대부분은 CLI를 사용한다.

01 깃 시작하기

깃 설치

Window git 설치

- 옆의 절차에 맞춰서 설치하세요
- Git bash는 git을 CLI를 window에서 사용할 수 있는 프로그램입니다

1. 웹브라우저에서 https://git-scm.com 사이트로 접속해 운영체제에 맞는 프로그램 내려받기
2. Select Components - 기본값 선택
3. Choosing the default editor used by Git - Use Vim 선택
4. Adjusting your PATH environment - Get from the command line ... 선택
5. Choosing HTTPS transport backend - OpenSSL 선택
6. Configuring the line ending conversions - ..Checkout Window style, commit unix-style.. 선택
7. Configuring the terminal emulator to use with GIt Bash - Use Window.. 선택
8. Configuring extra options - 기본값 선택
9. 설치완료 후 Git Bash 프로그램 실행 후 git + enter 후 옵션이 나타나면 완료

01 깃 시작하기

자주 쓰는 linux 명령어

자주 쓰는 linux 명령어

- pwd : 현재위치 경로
- ls : 디렉토리 파일, 디렉토리 목록확인(list segments)
  - -l : 상세보기
  - -h : 용량 단위 표시, 그냥하면 바이트 단위로만 표시
  - -a : 숨김파일도 표시
  - -t : 추가된 순으로 표시
  - -r : 역순으로 표시
- cd 디렉토리명 : 디렉토리 이동(change directory)
  - . : 현재 디렉토리
  - .. : 상위 디렉토리
  - ~ : home 디렉토리
  - mkdir 폴더명 : 폴더 생성(make directory)
  - -r : 디렉토리도 삭제
  - -p: 존재하지 않는 디렉토리의 하위디렉토리도 생성가능(mkdir -p dir/dir2)

01 깃 시작하기

자주 쓰는 linux 명령어

자주 쓰는 linux 명령어

- mv 이동할파일명 이동시킬위치 : 파일 이동(move)
- cp file cfile(경로표시) : file을 cfile로 복사(copy)
  - cp -f file cfile : 복사할 때 cfile이 존재할 경우 지우고 강제로 복사
  - cp -R dir cdir : 디렉토리 복사, 모든 파일과 하위 디렉토리를 복사
- vim 파일명 : 파일이 존재하는 경우 -파일 수정, 파일이 없는 경우 - 파일생성
- touch 파일명 : 파일의 용량이 0인 파일을 생성, 날짜 변경하는 명령어
  - touch filename : filename 파일 생성
  - touch -c filename : filename의 시각을 현재 시각으로 변경
- rm filename: filename삭제 (remove)
  - rm -f filename : fname파일 강제 삭제
  - rm -r dirname : dirname폴더 삭제

01 깃 시작하기

자주 쓰는 linux 명령어

자주 쓰는 linux 명령어

- cat fname : 터미널에 fname 내용 출력
  - cat fname1 fname2 : 두 파일 내용 이어서 출력
  - cat fname1 fname2 | more : fname1, fname2 페이지별 출력
  - cat fname1 fname2 | head : fname1, fname2 첫 10줄만 출력
  - cat fname1 fname2 | tale : fname1, fname2 끝 10줄만 출력
- '>' '>>' : redirection, 스트림의 방향을 조정
  - cat fname1 fname2 > fname3 : fname1 fname2를 fname3에 저장
  - cat fname4 >> fname3 : >> 왼쪽 명령의 결과를 fname3에 추가
- alias : custom command 만들기
  - alias new = 'command' : command가 new로 대체됨
  - alias : 현재 alias 목록 출력
  - unalias new : new라는 alias 해제

01 깃 시작하기

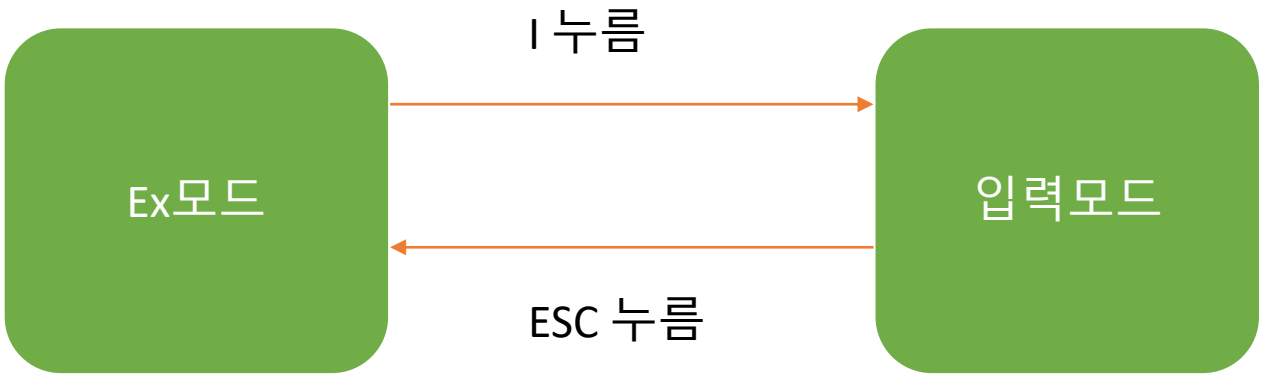
Git 설정

- Git 최초 설정
- 설정파일
  - etc/gitconfig 파일 : git config --system (시스템 전체 설정파일)
  - ~/.gitconfig, ~/.config/git/config 파일 : git config --global (현재 사용자에게만 적용)
  - .git/config : git config --local (현재 사용중인 저장소만 적용)
- 사용자 정보 설정
  - git config --global user.name = username
  - git config --global user.email = useremali
  - --unset: 설정 삭제

01 깃 시작하기

자주 쓰는 linux 명령어

Vim 사용하기



- Vim은 터미널에서 사용할 수 있는 리눅스 대표 편집기이다.
- 키보드만으로 편집기를 사용해야 한다
- i(insert) 또는 a(add)를 눌러 입력모드로 들어가야 내용을 입력할 수 있다.
- ex모드 명령어
  - :w or :write - 편집 중이던 문서 저장
  - :q or :quit – 편집기 종료
  - :wq – 편집 저장 후 종료
  - :q! – 저장하지 않고 바로 종료
  - /검색어 + enter – 검색어 검색
  - 등등

CONTENTS

02. 버전관리

- 깃 저장소 만들기
- 버전 만들기
- 커밋 내용 확인하기
- 버전 단계별 파일 상태 확인
- 작업 되돌리기
- 브랜치

02 버전 관리

깃 저장소 만들기

깃 초기화 – git init

- 따라하면서 해봅시다!!

1. mkdir dirname : 폴더 만들기
2. cd dirname : dirname 으로 이동
3. git init : 깃 저장소 초기화
4. ls –la : .git 폴더가 만들어졌는지 확인!

02 버전 관리

깃 저장소 만들기

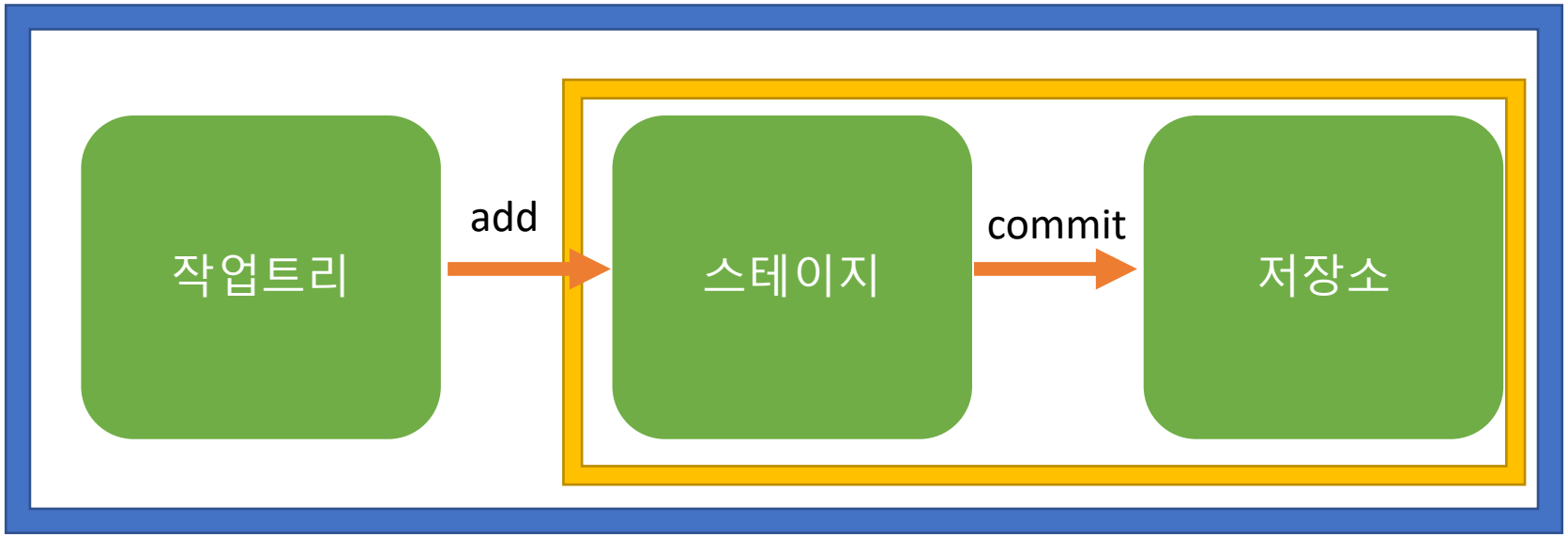
버전 만들기- git init

- 깃은 작업 디렉터리(작업 트리) 영역. 스테이지(stage,staging area) 영역, 저장소(repository)개념을 이용해 버전을 관리한다.
- 작업 디렉토리는 우리가 만드는 폴더를 생각하면 된다.
- 스테이지와 저장소는 '.git'폴더안에 존재 한다.
- 스테이지는 버전으로 만들 파일이 대기하는 곳이다.
- 저장소는 스테이지에 있던 파일들을 버전으로 만들어 저장하는 곳이다.

02 버전 관리

깃 저장소 만들기

버전 만들기- git init



- 스테이지와 커밋이 동작하는 과정은 다음과 같다.

1. 작업 트리에서 문서를 수정
2. 수정한 파일 중 버전으로 만들고 싶은 것을 스테이징 영역에 추가(git add filename..)
3. 스테이지에 있던 파일을 저장소로 커밋(git commit)

02 버전 관리

- 커밋 내용 확인하기

커밋 내용확인 – git log

- 버전을 관리하기 위해서는 지금까지 어떤 버전을 만들었는지 확인할 수 있고 버전마다 어떤 차이가 있는지 확인할 수 있어야 한다.

- **git log : commit hash(git hash), author(작성자), date(날짜), 커밋 메세지, 최신 버전 등 커밋 로그 정보 확인가능**
- **로그메시지가 많으면 enter로 다음 화면을 볼 수 있고 Q 를 누르면 로그 화면을 빠져나온다.**
- **자주쓰는 옵션**
  - **--stat : 커밋에 관련된 파일도 같이 출력**
  - **--oneline : 커밋 로그를 커밋당 한줄로 출력**
  - **--branches : 모든 branch의 커밋 로그 정보확인가능**
  - **--graph : branch의 분기과정을 graph로 나타내줌**

02 버전 관리

- 커밋 내용 확인하기

변경 사항 확인 - git diff

- git diff : 작업 트리에 있는 파일과 스테이지에 잇는 파일을 비교하거나, 스테이지에 잇는 파일과 저장소에 있는 파일을 비교해서 수정한 파일을 커밋하기 전에 최종적으로 검토할 수 있다.

- 존재하는 파일 수정 -> git diff -> 가장 최신의 버전과 어떻게 다른지 확인 가능

02 버전 관리

버전 단계별 파일 상태 확인

버전 단계별 파일 상태 확인

- 깃에서는 버전을 만드는 단계별로 파일 상태를 다르게 표시한다
- tracked / untracked file
  - tracked : 깃이 수정여부를 추적하는 파일,깃은 한 번이라도 커밋, 스테이징한 파일의 수정 여부를 계속 추적
  - untracked : 깃이 수정여부를 추적하지 않는 파일, 한번도 커밋되지 않은 파일
- unmodified / modified / staged - tracked 파일에서
  - unmodified : 수정 없는 파일
  - modified : 수정된 파일
- staged : commit이 가능한 파일

untracked -> unmodified -> modified -> staged

->스테이징

->->->->->->->->파일수정

->->->->->->->스테이징

<-<-<-<-<-<-<-<-<-<-<-<-<-<-커밋

02 버전 관리

버전 단계별 파일 상태 확인

.gitignore 파일로 버전관리에서 제외

- 버전 관리 중인 디렉터리 안에 버전관리를 하지 않을 특정 파일 또는 디렉토리를 .gitignore파일을 만들어 버전관리에서 제외시킬 수 있다.
- 파일명, 폴더명, 확장자(ex .png) 등 입력 가능
- .gitignore파일은 .git 폴더가 있는 최상위 폴더에 존재해야 한다.

02 버전 관리

작업 되돌리기

작업 되돌리기

- 스테이지에 올렸던 파일을 내리거나 커밋을 취소하는 등 각 단계로 돌아가는 방법
- HEAD는 현재 작업하고 있는 Branch를 가리키는 포인터 역할
- git checkout : 작업트리에서 수정한 파일 되돌리기
- git reset HEAD filename : 스테이징 되돌리기
- git reset HEAD^ : 최신 커밋 되돌리기
- git reset 커밋해시 : 특정 커밋(커밋해시를 기진 커밋)으로 되돌리기
- git revert : 커밋 삭제하지 않고 되돌리기

02 버전 관리

작업 되돌리기

실습

실습하기

1. git checkout
    1. 파일 수정, modified상태 만들기
    2. git checkout filename
    3. git status
2. git reset HEAD filename
    1. 파일을 수정하고 스테이징
    2. git reset HEAD filename
    3. git status
3. git reset HEAD^
    1. 파일 수정하고 커밋
    2. git reset HEAD^
    3. git status
    4. 파일 내용확인
5. git reset 커밋해시
    1. 여러개의 커밋
    2. 돌아가고 싶은 커밋의 커밋해시 복사(git log)
    3. git reset 커밋해시
    4. git log
    5. 파일 내용확인
6. git revert
    1. 여러개의 커밋
    2. 돌아가고 싶은 커밋의 커밋해시 복사(git log)
    3. git revert 커밋해시
    4. git log
    5. 파일 내용확인

02 버전 관리

Branch

Git과 Branch

- 깃을 처음 시작하면 master라는 branch가 만들어 진다.
- 브랜치를 통해 기존에 저장한 파일을 master브랜치에 그대로 유지하면서 기존 파일 내용을 수정하거나 새로운 기능을 구현할 파일을 만들 수 있다.
- 개발할 때 코드를 통째로 복사해서 다른 내용의 코드를 만들어야 하는 상황을 복사하지 않고 가능하게 만들어 준다.

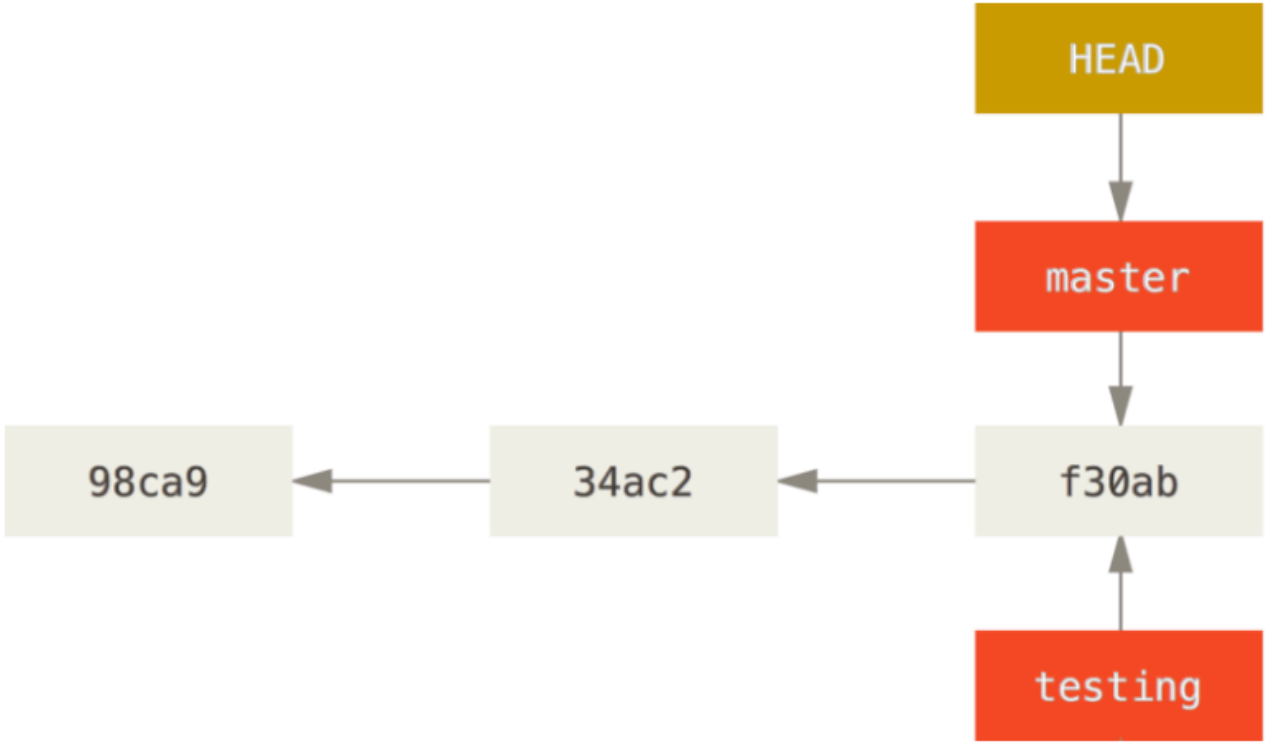
02 버전 관리

Branch

Branch 만들기

- git branch : 브랜치 목록 확인
- git branch 브랜치이름 : 브랜치이름의 브랜치를 만듦

```
git branch testing
git checkout testing
```

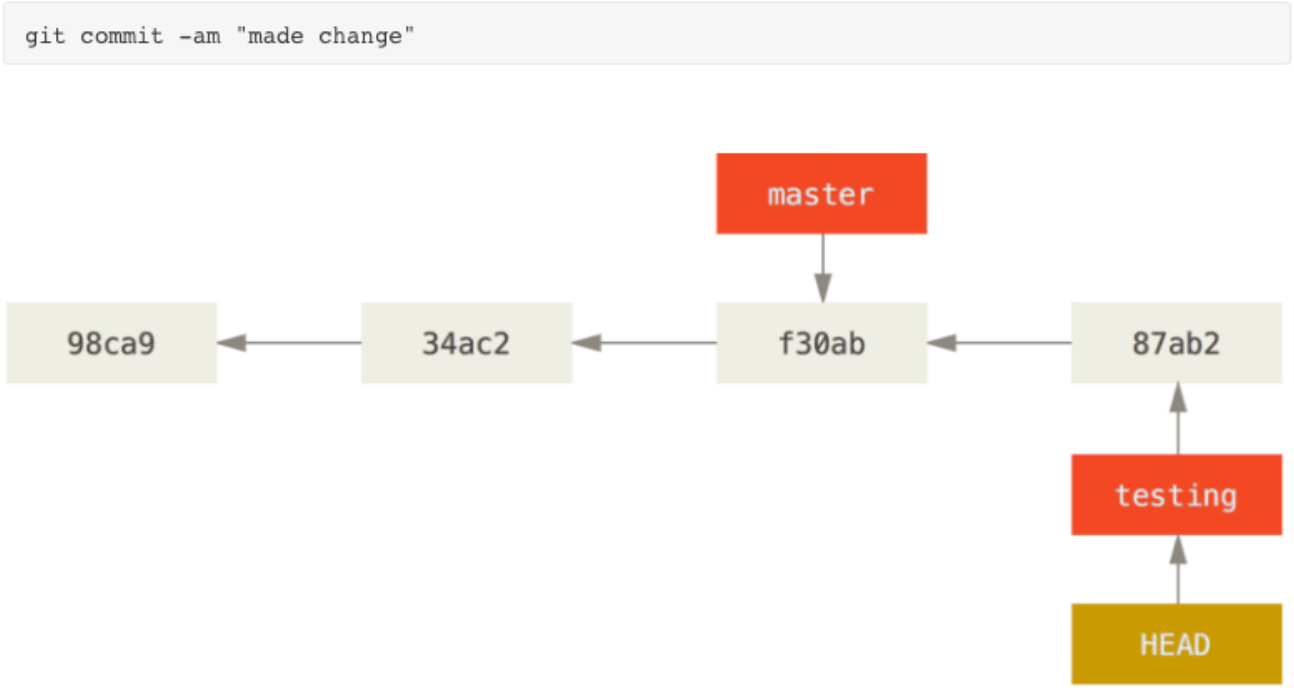


02 버전 관리

Branch

Branch 이동

- git checkout : 브랜치 사이 이동하기
- git checkout -b branchname : branchname을 만들고 이동 (git branch branchname + git checkout branchname)



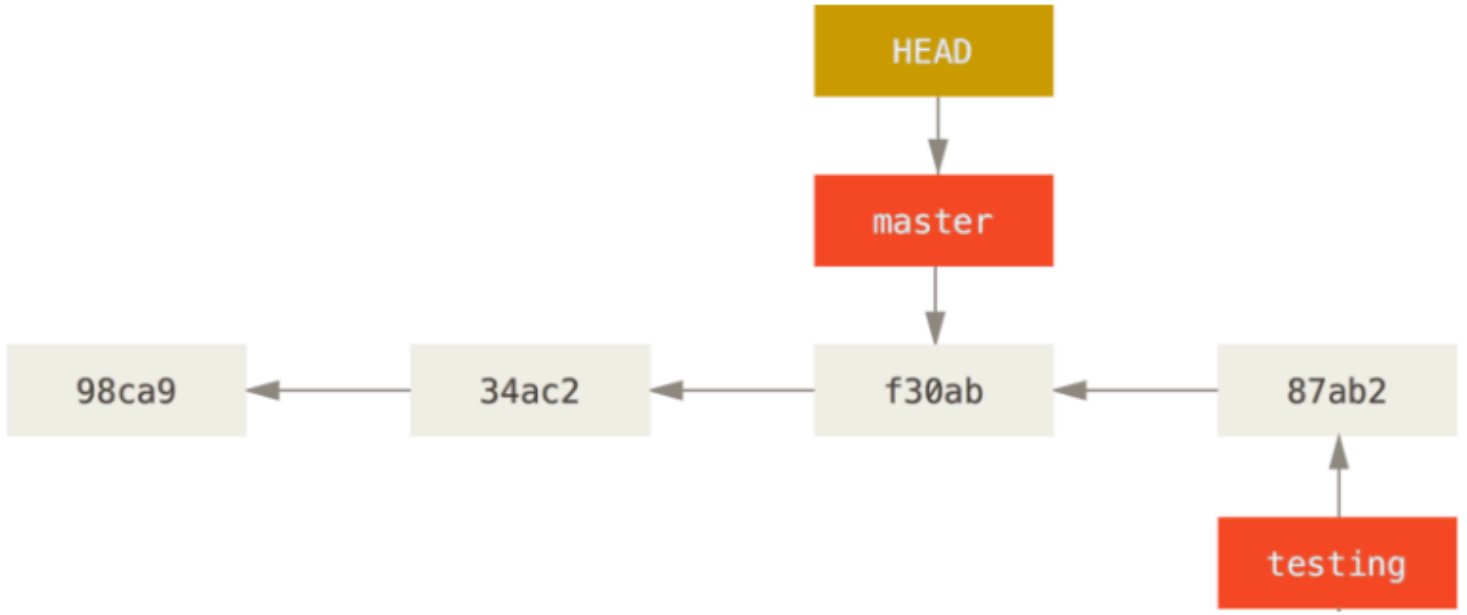
02 버전 관리

Branch

Branch 이동

- git checkout : 브랜치 사이 이동하기
- git checkout -b branchname : branchname을 만들고 이동 (git branch branchname + git checkout branchname)

```
git branch testing
git checkout testing
git commit -am "made change"
git checkout master)
```



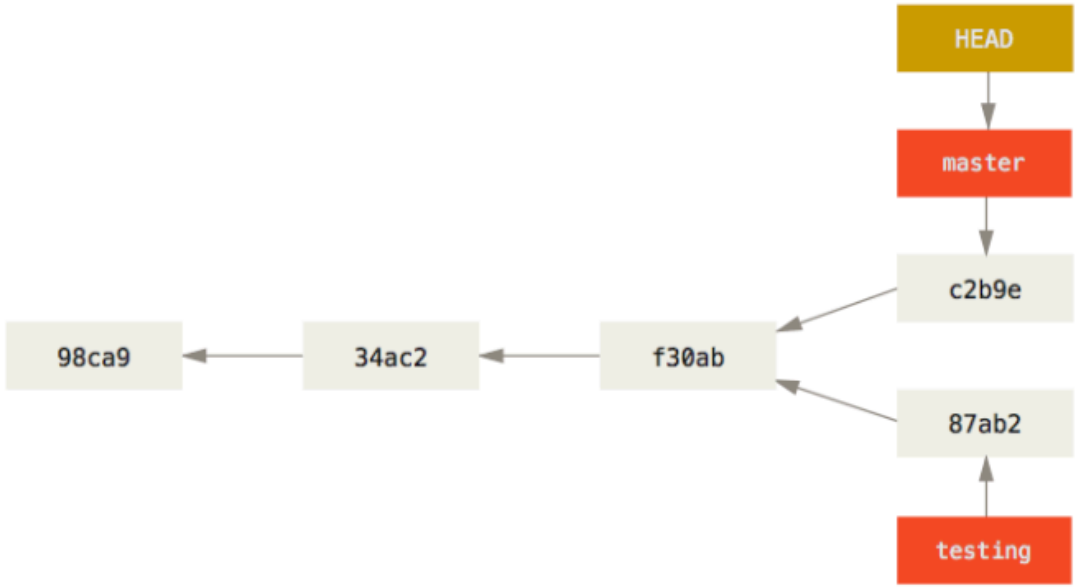
02 버전 관리

Branch

Branch 이동

- git checkout : 브랜치 사이 이동하기
- git checkout -b branchname : branchname을 만들고 이동 (git branch branchname + git checkout branchname)

```
git branch testing
git checkout testing
git commit -am "made change"
git checkout master
git commit -am "made other change"
```



02 버전 관리

작업 되돌리기

실습

실습하기

1. manual 폴더만들고 깃 저장소 초기화
2. 커밋 3번 하기
3. 브랜치 여러개 만들기

02 버전 관리

Branch

Branch 정보 확인하기

- git log --oneline --branches : 각 브랜치의 커밋을 함께 볼 수 있다.
- git log --oneline --branches --graph : 각 브랜치의 커밋간의 관계를 보기 쉽게 그래프 형태로 표시
- git log master..testing : master branch에는 없고 testing branch에만 있는 커밋을 보여준다.
- git log testing..master : testing branch에는 없고 master branch에만 있는 커밋을 보여준다.

02 버전 관리

작업 되돌리기

실습

실습하기

1. 위의 실습상황에서 다른 브랜치로

   이동 후 커밋하기

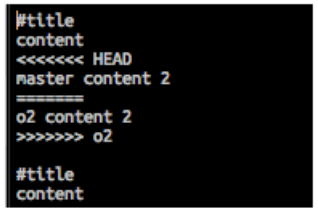
2. 브랜치간의 차이 확인하기

02 버전 관리

Branch

Branch 병합하기

- 각 브랜치에서 작업하다 어느 시점에서 브랜치 작업을 마무리하고 기존의 브랜치와 합쳐야하는 상황 발생한다.
- git merge branchname : HEAD가 가리키는 branch와 branchname을 병합
- 병합할 때 파일 수정 위치에 따라 충돌이 발생할 수 있다.
  1. 서로 다른 파일 병합
  2. 같은 파일 다른 위치 수정 후 병합
  3. 같은 파일 같은 위치 수정 후 병합
     - 충돌 발생



<<<<<<< HEAD

현재 HEAD가 가리키고 있는 브랜치의 내용

==========

병합할 브랜치의 내용

>>>>>>>> branchname

- 충돌이 발생하면 양쪽 브랜치의 내용을 참고하며 수정 후 구분선을 삭제후 다시 저장
- 저장한 파일을 커밋하면 완료
- 병합 및 충돌 해결프로그램 사용가능
  - p4merge, meld, kdiff3 등
  - p4merge설치법

02 버전 관리

Branch

Branch 삭제하기

- 병합 후 더 이상 사용하지 않는 브랜치는 삭제가능하다
- git branch -d branchname: git branch의 -d 옵션을 사용하면 삭제가능

02 버전 관리

작업 되돌리기

실습

1. 폴더만들고 깃 초기화
2. 마스터브랜치에서 텍스트파일 만들고 커밋
3. 다른 브랜치 만들기
4. 마스터브랜치에서 텍스트파일 수정
5. 다른브랜치로 이동후 같은위치 수정
6. 마스터브랜치로 이동후 병합
7. 충돌발생, 파일 수정후 다시 커밋 후 충돌해결
8. 병합된 브랜치 삭제

02 버전 관리

Branch

Branch 관리

브랜치에서 checkout reset

- git reset 커밋 해시 : HEAD가 기리키고 있는 branch를 커밋 해시가 가리키는 커밋으로 이동시킨다.
- branch안에서만 커밋을 이동할 수 있을 뿐만 아니라 branch간에도 커밋을 이동할 수 있다.

수정 중인 파일 감추기, 되돌리기

- git stash : modified file들을 잠시 unmodfied파일로 감춰둘 수 있다
  - 현재 작업중인 파일들과 관련 없는 작업을 따로 할 때 실수로 관련 없는 파일들이 커밋되는 것을 방지할 수 있다.
  - 현재 시점의 modified file들을 unmodified file로 바꾼다
- git stash pop : stash 목록에서 가장 최근의 항목 목록에서 삭제 후 되돌린다
  - git stash list : stash 목록 출력
- git stash apply : stash 목록에서 가장 최근의 항목 목록에서 삭제 하지않고 되돌린다
- git stash drop : stash 목록에서 가장 최근의 항목을 삭제한다.

CONTENTS

03. 백업

- 깃허브
- 원격저장소 연결
- 원격저장소 사용
- SSH접속

03 백업

깃 허브

Github 백업

- 깃 허브(Github)는 깃을 관리할 수 있는 원격저장소를 제공하는 대표적인 서비스이다
- 깃 사용, 지역 저장소 백업, 협업프로젝트, 개발 이력 기록, 다른 사람의 소스 사용, 오픈소스 참여 등을 할 수 있다.

03 백업

깃 허브

Github 백업

- local, remote 연결하기
- git remote add orgin 원격저장소주소 : .로컬 저장소에서 실행하면 원격저장소 주소를 가진 원격저장소와 연결된다.
- git remote -v : 연결확인

03 백업

깃 허브

Github 백업

- **원격 저장소에 파일 올리기, 내려받기**
- git branch --set-upstream-to=origin/<원격브랜치> <로컬브랜치> : '<원격브랜치>' 브랜치가 리모트의 '<로컬브랜치>' 브랜치를 ('origin'에서) 따라가도록 설정.
  - 원격 저장소에서 여러개의 브랜치가 있고 브랜치끼리 연결시킬때 사용
- git push : 연결된 원격저장소에 파일 올리기
  - git push <원격저장소> <로컬브랜치>
  - 처음 push는 git push -u orgin master와 같이 -u 옵션을 붙여야 한다.
- git pull : 연결된 원격저장소의 파일 내려받기
  - git pull orgin master : orgin(원격저장소주소 줄임말)의 내용을 master 브랜치로 가져온다.
  - 깃에서 기본 브랜치를 master라고 하는 것 처럼 기본 원격 저장소는 orgin이라는 이름을 사용한다
- push전에 pull을 해줘서 원격저장소와 커밋상태를 맞춰줘야한다.
- 깃허브 사이트에서 직접 커밋할 수도 있다

03 백업

깃 허브

Github 백업

깃허브 SSH원격 접속하기

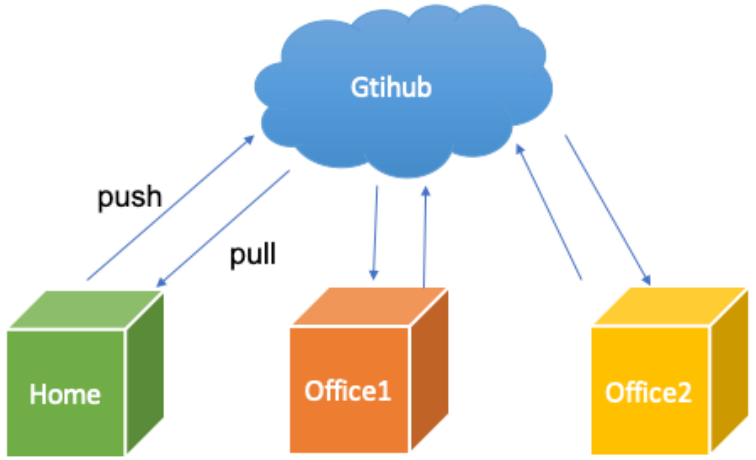
- SSH 접속은 프리이빗 키와 퍼블릭 키를 이용해 깃헙 서브에 해당 컴퓨터를 인증하는 방식의 접속방법이다.
- SSH 접속을 이용하면 번거로운 로그인 등을 피할 수 있다.

1. ssh-keygen 으로 ssh키 생성하기, 키 생성경로 메모해두기
2. 메모해둔 경로로 둘어가 퍼블릭키 내용 복사
3. 복사한 내용을 github 계정 setting의 SSH and GPG keys 에 들어가 add ssh key에 붙여넣고 등록
4. 이제 퍼블릭키를 등록한 컴퓨터는 ssh 주소를 이용해 로그인 없이 언제든 깃헙 서버에 접속할 수 있다.
5. 연결은 위와 같은 방법(git remote add orign 주소) 명령어를 사용하면 된다.

03 백업

깃 허브

Github 백업



여러 컴퓨터에서 원격저장소 함께 사용 - 같은 계정으로

- 원격저장소 복사
  - git clone 원격저장소주소 로컬저장소이름: 원격저장소 주소를 로컬저장소에 복사한다.
  - 로컬저장소이름을 가진 폴더가 없다면 생성되고 있다면 그 안에 생성된다.
  - 로컬저장소를 지정해주지 않으면 현재폴더에 복사된다.
- 여러 컴퓨터에서 같은 원격 저장소를 복사하여 같이 사용할 수 있다.

03 백업

깃 허브

Github 백업

원격 브랜치 정보 가져오기

- 원격 저장소를 로컬저장소와 합치기전에 원격저장소에 어떤 변화가 있었는지 확인할 필요가 있다.
- 이런 경우에는 원격저장소의 정보만 가져올 수 있다.
- 원격저장소와 연결 후 git log명령어 결과내용에 orgin/master, HEAD->master가 생긴다
  - HEAD -> master : 해당 커밋이 지역 저장소의 최종 커밋
  - orgin/master : 헤당 커밋이 원격 저장소의 최종 커밋
- git fetch : 원격 저장소 정보 가져오기
  - 원격 저장소의 내용을 지역 저장소와 바로 합치지 않고 정보만 가져와서 검토할 수 있다
  - 원격 저장소에서 가져온 정보는 FETCH_HEAD 브랜치로 가져온다.

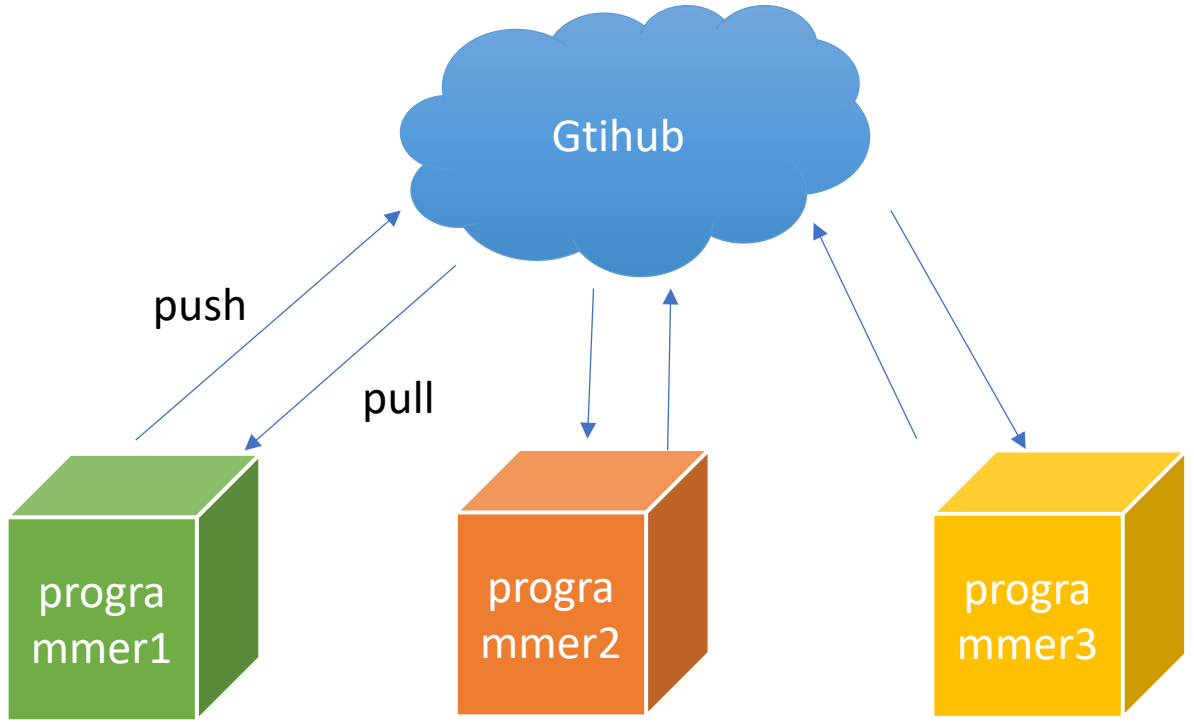
CONTENTS

04. 협업

- 원격저장소 함께 이용
- 원격 브랜치 정보 확인
- 협업 기본
- 협업 브랜치 사용

04 협업


Kuggle


Gtihub
push
pull
progra
mmer1
progra
mmer2
progra
mmer3

04 협업

깃 허브

다른 사용자와 협업하기

- 깃헙 무료계정은 최대 3명의 공동작업자를 추가할 수 있다 – 다른 계정

1. 원격저장소의 팀장 계정이 공유할 원격저장소 `settings -> manage access 에서 공동작업자 추가가능
2. 팀장 계정 작업환경구성
    - git config --local user.name 사용자이름
    - git config --local user.email 사용자이메일
3. 팀원 계정이 원격 저장소 복제
    - git clone 원격저장소주소
    - 복제한 저장소의 첫 커밋을 하는것이아니라면 git pull 먼저 실행
    - 커밋후 원격저장소에 업데이트 : git push -u origin master

04 협업

깃 허브

협업에서 브랜치 사용

- git push origin <브랜치> : orgin에 <브랜치>브랜치를 푸시한다.
  - 웹브라우저 깃헙에서 확인하면 브랜치가 추가된 것을 확인할 수 있다.
- pull requests로 푸시한 브랜치 병합
  - 푸시한 브랜치는 pull request를 통해 병합해야 원격 저장소에 반영된다.

1. 로컬 저장소에서 새 브랜치를 푸시한다
2. 웹브라우저로 깃헙의 원격 저장소에 접속해 pull request 버튼을 누르고 new pull request 를 누르면 새로운 pull request가 생성된다.
3. 생성된 pull request는 collaborator중 한명이 승인해야 원격저장소 master브랜치에 병합할 수 있다.

CONTENTS

05. 깃허브 활용

- 오픈소스
- 블로그 만들기

감사합니다