Cogsys Winter Code Hack:
End-to-end pipeline with collaboration

Daehyun Cho

Cognitive System Lab

A.I. Dept, Korea Univ.

Updates

- 추가된 서버들 세팅 완료. 계정이 필요하시면 파드립니다.
- 이번 겨울 일정 픽스
- 다음 A.I 관련 일정

CognitiveSystems

- 연구도 좋지만, 코드를 잘 짜는 것도 만만치 않게 중요함
- 특히 전공특성상 우리의 상황은
  - 정부에서 쓸데없이 갑자기 밀어줘서 개나소나 인공지능 하는 중임
  - 심지어 학부 아니고 석/박사들도 발에 채이는 중
  - 요즘 박사도 코테를 봐야한다는 소문이 있음
- 어쨌든 개발직군에 더 가까운 전공에 소속되어 있기 때문에,

  우리는 연구도 연구지만 개발역량을 무시할 수 없는 개발과 연구 사이 개구자...? 연발자..? 하여튼 -

CognitiveSystems

- 딥러닝 모델 개발은 아래와 같은 프로세스들을 포함함.

- 새로운 모델 / 파이프라인 짜기 (new code!)
- 짜놓은 파이프라인 위에 실험을 추가하기
- 실험을 관리하기
- 실험을 모니터링하기
- 유지보수하기 (for future works)

- 하지만 대다수의 연구자들은 이것을 그렇게 소중히 여기지 않음
- 사실 우리 교수님만해도 이렇게까지 진행하시지 않는데, 교수님이야 이미 교수님이고….
  우리는 우리 밥그릇 챙기려면 적어도 저 프로세스가 내 머릿속에 완벽히 정리되어 있어야함

CognitiveSystems

- 이 프로세스를 어떻게 학습할까?
- 하나부터 누가 떠먹여주면 좋겠지만, 그런 일은 세상에 없다.
- 만약 그런 일이 있다면 알려준 사람한테 아주 고마워 해야 한다. 개발자의 세계는 냉정한걸… ~~내거 짜기 바쁨~~

  (해주길 기대하는 것 자체가 금지임)

- 사자우리에 밀어 넣는 것만큼 확실한 방법이 있을까 싶어서

  이번 겨울에는 처음부터 코드를 짜서 대회에 제출하고 성능을 올리는 전체적인 프로세스를 직접해보는 과정을 실습

전체적인 계획

1. 대회 참여하기 & GitHub Repository 개설하기D
2. Pytorch DataLoader 구축하기 & Code Review하기
3. Pytorch Model 개발하기 & 제출해보기
4. 여러 가지 실험을 하기 위해 세팅하기 (w/ Hydra) & wandb logging하기
5. Pytorch-Lightning으로 갈아타보기
6. Accuracy 100% 만들기

이 과정을 조금 불친절하게 진행할 예정

CognitiveSystems

지향

✅ 알잘딱깔센

✅ Collaborations

✅ 조금이라도 모르면 구글링 (선무당이 사람잡음)

지양

❌ **Copy & Paste**: 단순한 syntax정도는 괜찮은데, 전체 파이프라인을 베껴오는 건 취지를 아주 많이 벗어남

❌ 나의 온 시간을 투자하기: 짬짬히... 틈틈히 진행하는 것을 권장 (교수님한테 혼나기 싫음)

CognitiveSystems

TODO List

- ✅ Join Kaggle Competition
- ✅ Connect local directory to remote GitHub Repository and add README.md
- ✅ Invite collaborators
- ✅ Create .gitignore
- ✅ Split branch from main and push
- ✅ Make a pull request and get a code review
- ✅ Add flake8 workflow in github

CognitiveSystems

대회 참여하기

Start Kaggling!

CognitiveSystems

1 Digit Recognizier

CognitiveSystems

Kaggle

- Kaggle은 모두가 아는 대회 플랫폼
- 여기서 아주 쉬운 Computer vision 대회를 참여할 예정이다.
- 사실 말이 competition이지, 이 대회는 Kaggle에서 Playground용으로 내놓은 Toy-competition이라 대회 기한이 없다.
- https://www.kaggle.com/competitions/digit-recognizer
- **링크를 타고 가서 Join Competition 해보자 (Do it yourself)**

1 Looking at Data

Kaggle

- 먼저 대회의 특성을 이해하는 것이 중요하다.
- Kaggle에 있는 온갖 탭을 눌러서
  이 대회가 뭐하는 대회인지 파악해보자.

다음 질문에 답해보자

1. 뭐하는 대회인가?
2. 대회를 평가하는 지표는 무엇인가?
3. 데이터는 어떤 형태로 제공되는가?

Dataset Description

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 × 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

```
000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
 |   |   |   |  ...  |   |
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783
```

The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column.

Your submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit you predict. For example, if you predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then your submission file would look like:

Files
3 files

Size
128.13 MB

Type
csv

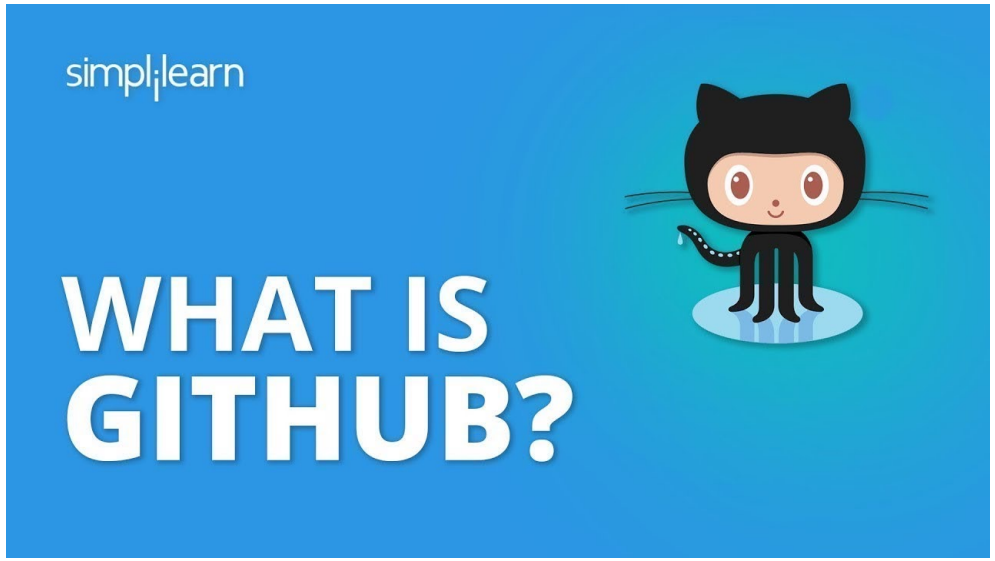
GitHub Repository

Start Collaboration

CognitiveSystems

What is GitHub

- 코딩 하다보면 바뀌는 내용이 정말 많다
- 내가 두 개 정도 기능을 바꿨다고 '인지'하지만, 실제 코드 히스토리를 보면 바꾼 게 한 두개가 아님을 알 수 있다.
- 당연히 과거의 내가 맞는 경우도 많기 때문에 이 때의 히스토리를 불러와야 하는데, 이걸 직접 관리하는 게 만만치 않다.
- 이렇게 과거의 버전들을 다 기록하고 관리하는 시스템을 Version Control System (VCS)라고 부르는데,
  이 솔루션 중 하나가 가장 대표적인 GitHub이다.
- 다른 사람과 공동개발을 할 때 수뇌부 역할을 한다.



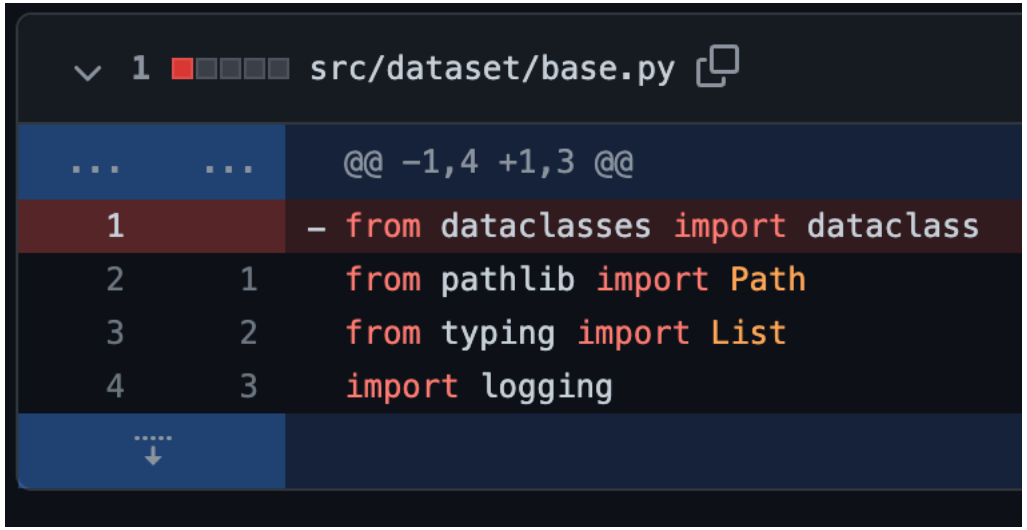
Code History

Pytorch-lightning

- GitHub은 “**변경 내역**”을 기록한다.
- 아래와 같이 특정 ”commit”에서 다음의 라인이 제거됨을 알려준다.
- 전체 히스토리를 다 가지고 있기 때문에 내가 최근에 한 멍청한 짓을 과거의 내가 와서 도와줄 수 있다.
- 아까도 말했지만 날 도와주는 사람은 없다. 그렇기 때문에 더더욱 github code history가 중요해진다.
- 실제로 github 덕분에 연구가 터질 뻔한 걸 무려 4-5번 정도 막은 경험이 있다…. 없었다면 큰일날뻔

  (e.g. 갑자기 하드가 날아갔어요! → 코드가 다행히 웹에 있음,

  뭔가 바꿨는데 갑자기 훈련이 안됩니다! → 과거의 돌던 코드가 있으니 거기로 돌아가면됨)



2 Make your own repository

Pytorch-lightning

- 자 이제 가서 팀 Repository를 만들자 (Do it yourself)
  - 레포 만들어야하는데 뭔 옵션이 이렇게 많나요? → 마시면서 배우는 술게임 같은 거니까 맘대로 조작해보세요
- 무조건 개인레포에는 나만 작업할 수 있다. 타인의 레포에 작업을 하는 방식은 크게 두 가지가 있는데
  - Collaborator로 등록해서 같은 레포 내에서 작업
  - 레포를 Fork 따서 원본 레포에 PR을 넣는 방법 (보통 라이브러리에서 내가 발견한 에러 같은 것들을 수정한 사안을 반영할 때 사용)
- 여러 명이서 한 팀으로 작업한다면, 내 팀원에게 Collaborator Invitation을 보내보자! (Hint: Settings tab)

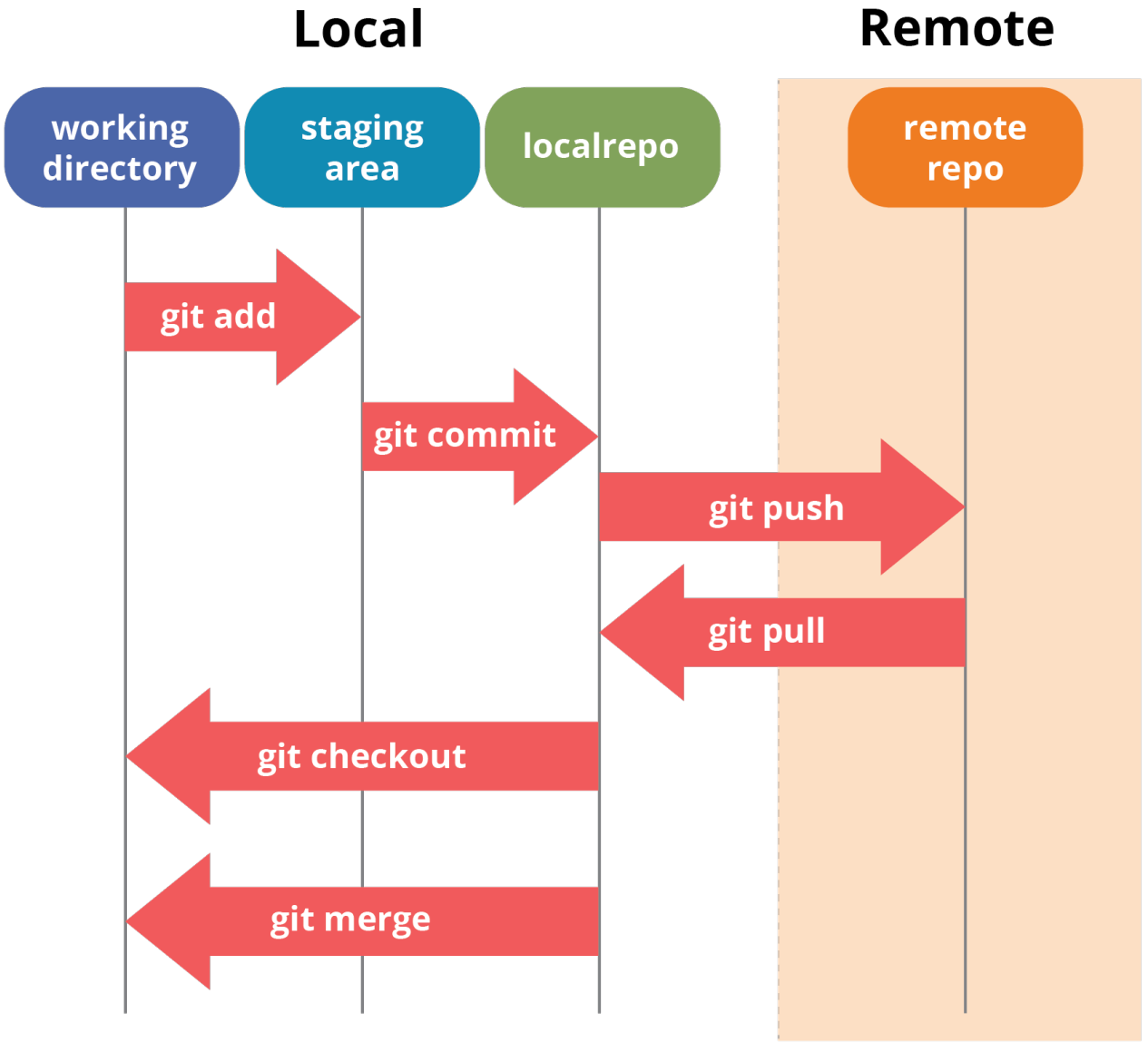
GitHub을 쓰다 모르겠다. 다음 중 취해야할 액션은?

1. 아무나 붙들고 물어본다
2. 적절한 keyword로 구글링해본다
3. Documentation을 찾아본다.

정답은 적절도로 나열하면 2- 3- 1입니다. 2, 3 다해봐도 모르겠을 때 1을 합니다.

가장 기본적인 사용법을 알아보자

- 기본개념은 ”Local”과 “Remote”이다.
  - Local은 내가 현재 내 컴퓨터에서 작업하고 있는 내역
  - Remote는 GitHub 서버에서 관리되고 있는 내역
  - 공동작업의 경우, 다른 사용자는 Remote에서 내역을 가져오기 (=pull)하기 때문에 Remote가 중심부역할이다.
- Git의 ABC는 add - commit - push 그리고 pull이다.
  - Local 에서 변경사항이 생긴 파일을 add
  - Commit은 add한 파일들을 한 데 묶어주는 역할.
  - Push는 Remote에 변경사항을 반영하는 역할.
  - 작업 중간중간 pull을 통해 다른 사람의 변경사항을 반영해주는 것이 중요하다
    - 미뤘다가 몰아서 하면 conflict 파티가 날 수 있다.



README.md 파일을 만들어 우리의 레포가 어떤 레포인지 설명을 작성한자. (흔히 대문이라 부른다)

1. 뭘 어떻게 쓰라는거지? → 레포는 널리고 널렸다. 좋은 예시를 벤치마킹하도록 하자
   - https://github.com/rwightman/pytorch-image-models

     파이토치에서 CV에 많이 사용되는 모델들이 들어있는 timm 라이브러리의 레포. 여기에는 업데이트 이력에 대해 자세히 나와있다.
   - https://github.com/CompVis/stable-diffusion

     핫하디 핫한 Stable Diffusion 레포. 어떤 연구에 대한 코드인지, 어떻게 사용하는지 등이 담겨있다.

2. .md 파일은 뭐지? → markdown의 약자. 코드로 글을 쓰는건데 노션 사용법이 markdown 사용법이다. 명령어가 쉽기 때문에 금방 익힐 수 있다.

   이 또한 용례를 보고 싶으면 위의 레포들의 README.md의 Raw file을 살펴보면 된다.

   https://ko.wikipedia.org/wiki/%EB%A7%88%ED%81%AC%EB%8B%A4%EC%9A%B4
3. GitHub은 마이크로소프트가 인수하면서 기능이 굉장히 많이 추가됐다. 오른쪽 사이드탭에 많은 정보들을 담을 수 있는데 탐색해보면 좋다.

작성을 완료했으면 Remote에 반영해보자.

.gitignore

GitHub

코딩하다보면 잡스러운 부산물들이 많이 생긴다.

- 중간중간 .png, .csv 같은 중간작업물
- C-compile의 결과로 등장하는 __pycache__ 파일들
- Mac 유저의 경우 .DS_Store, Window 유저의 경우 desktop.ini 라는 파일이 몰래몰래 암세포처럼 생긴다.

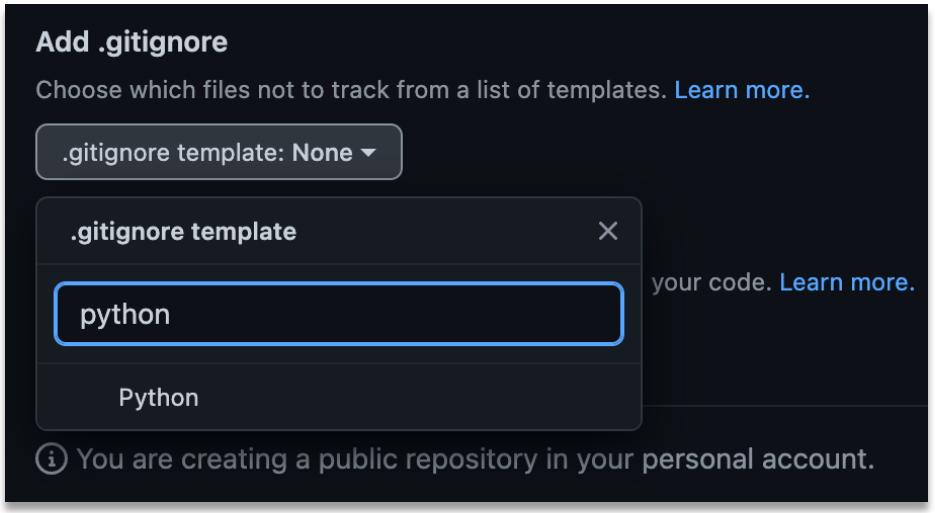
이 작업물들까지 레포에서 관리할 필요가 없다. 이런 파일들을 git에서 무시하도록 도와주는 `.gitignore` 라는 파일이 있다.

- 가령 jupyter notebook 같은건 레포에서 관리하고 싶지 않으면, .gitignore에 *.ipynb를 넣어주면 된다.
- Github이 사용자가 사용하는 언어에 따라 .gitignore를 제공해준다. 귀찮은 __pycache__ 같은것들이 기본적으로 포함되어 있다.

.gitignore를 레포에 추가해주자

- 사실 눈치 챈 사람이 있다면 아까 **새로운 레포를 만들 때**

  .gitignore 파일을 처음에 미리 추가할 수 있다는 것을 알 수 있다.



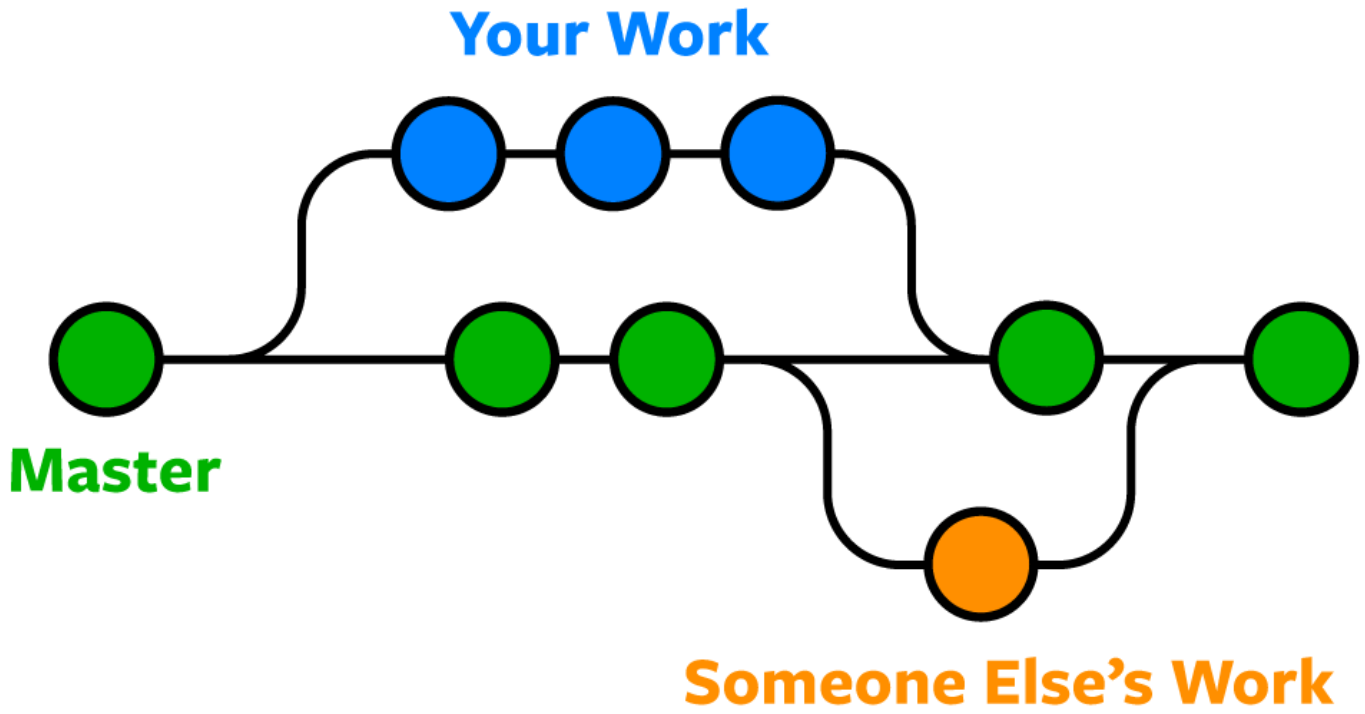
코딩 히스토리를 알 수 있다는 장점은 이제 알 것 같은데, 이게 공동작업이랑 무슨 관련이 있는지 궁금할 수 있다.

GitHub의 또 다른 파워는 branch라는 기능이다.

만약 하나의 히스토리 위에서만 모든 파일이 관리된다고 치면, 누군가 파일을 수정했을 때, 내가 그 파일을 쓰고 싶지 않을 수 있다.

이럴 때 서로 합의를 통해 파일을 고쳐나가야 하는데, 하나의 히스토리에서만 이 이력을 가져오면 당연히 충돌이 생기거나 남이 쓴 코드만 써야한다.

이를 방지하기 위해 나만의 히스토리를 따로 만들어서 어느 정도 모두가 사용 할만하다 싶으면 다시 큰 흐름에 다시 합치는 기능이 branch이다.



Branch

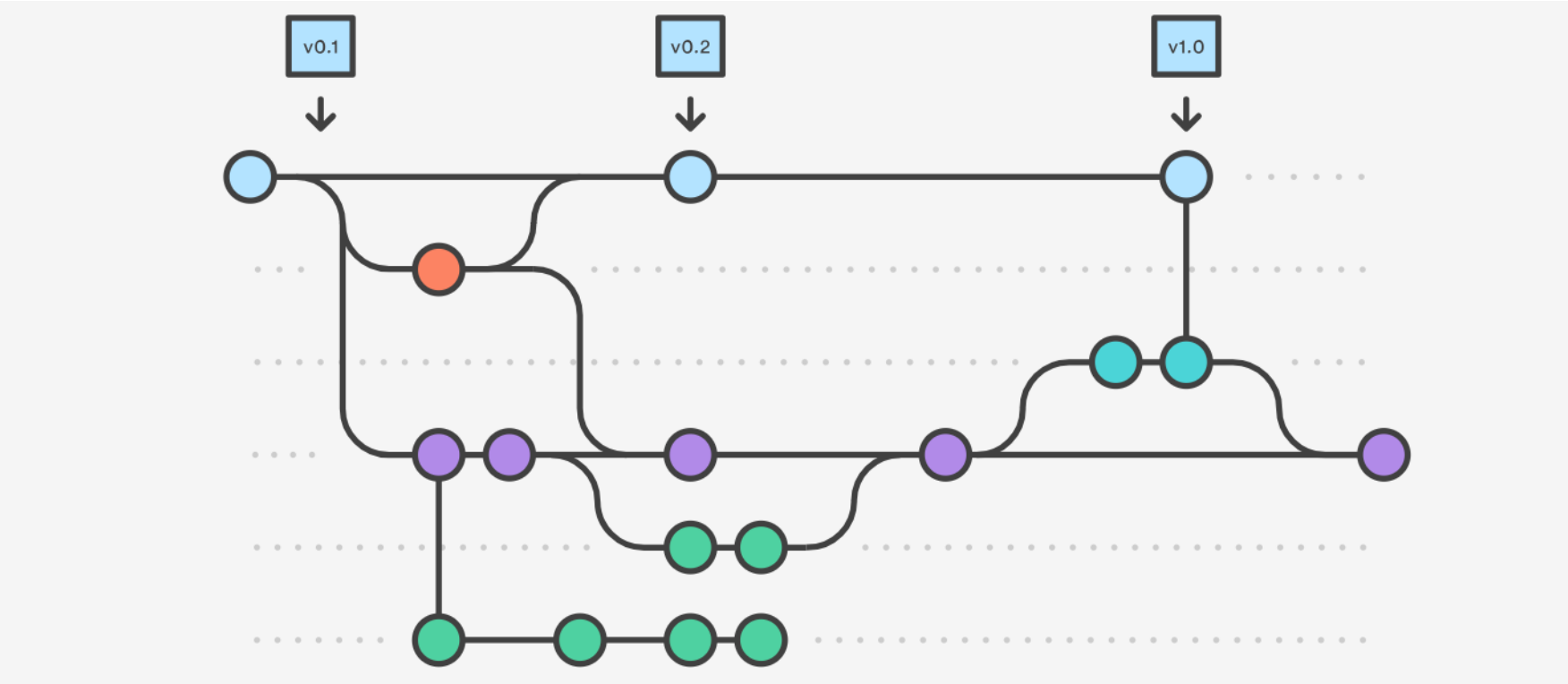
GitHub

가장 기본적인 사용법은 git branch (branch이름) 을 통해 branch를 딸 수 있다.

이 때 내가 현재 어느 브랜치에 있느냐가 중요하다. (아래 그림 참조)

다른 브랜치로 이동하는 것은 git checkout (branch이름)으로 진행할 수 있다.

이 때 현재 브랜치에 commit하지 않고 modified 된 내역이 있으면 checkout할 수 없다. 항상 checkout 전에 commit을 해줘야한다.



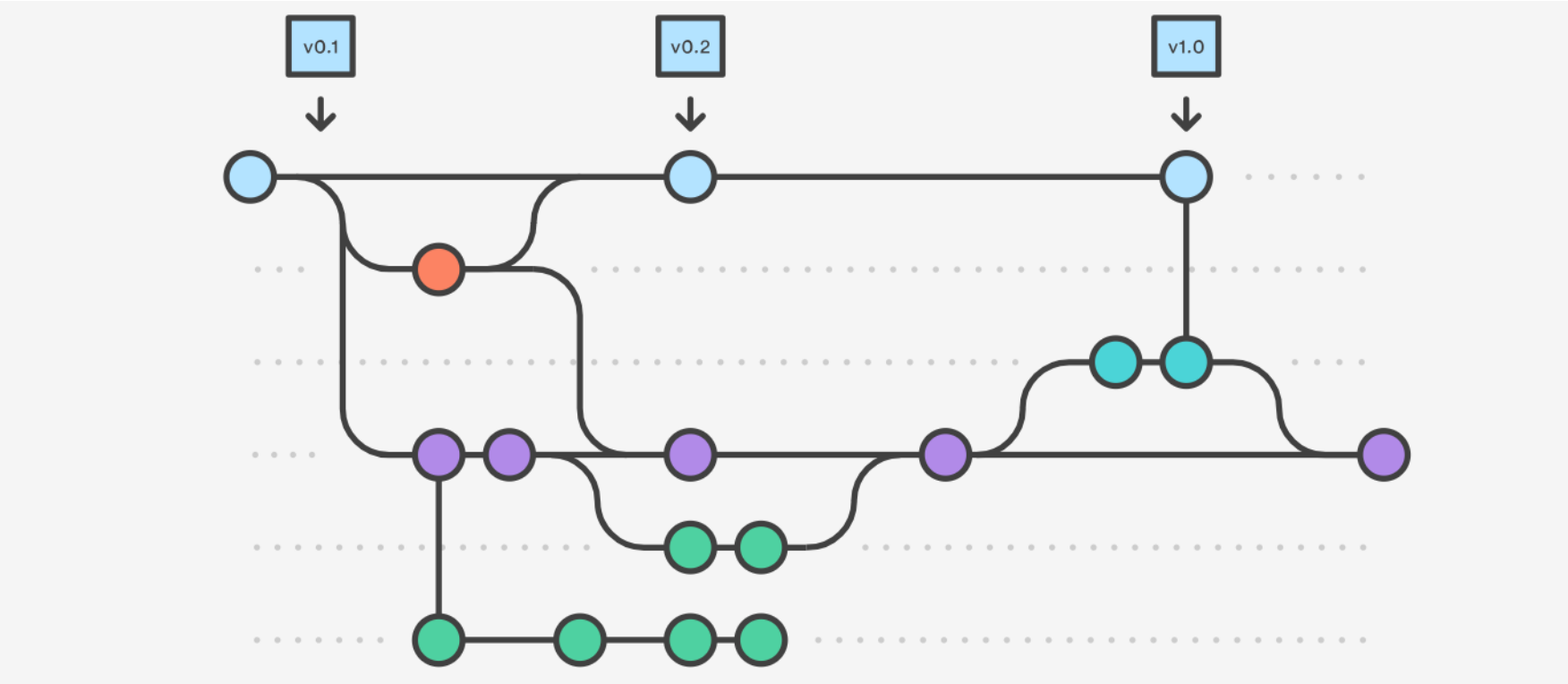
Branch

GitHub

오늘은 실습하지 않고 이런 개념이 있다는 것만 알아두면 된다.

Git branch는 git에서 제일 핵심 기능이라고 해도 무방할만큼 중요하기 때문에 documentation을 한 번 읽어보는 것을 추천한다.

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches



2 Options

GitHub

- Add-commit-push를 두 번이나 진행해봤다.
- 이 세 명령어는 추가적인 옵션을 통해 사용자가 좀 더 편하게 이용할 수 있게 한다.
- 대표적인 예로 git add를 하되, 내가 “새롭게” 만든 작업물을 제외하고 “변경한” 작업물만 추가하고 싶을 수 있다.
- 이럴 때는 git add -u를 통해 변경한 작업물만 add할 수 있다.
- 여러 옵션은 굳이 다 알 필요는 없는데, " 내가 지금 치고 있는 명령어가 너무 불편한데 더 간편한 게 없을까?” 하고 찾아보면 된다.
- 내가 불편하면 보통 다른 사람도 불편하다. 어지간하면 누가 만들어놨다.

https://git-scm.com/docs/git-add

Thank you 🙏

Daehyun Cho

1phantasmas@korea.ac.kr

CognitiveSystems