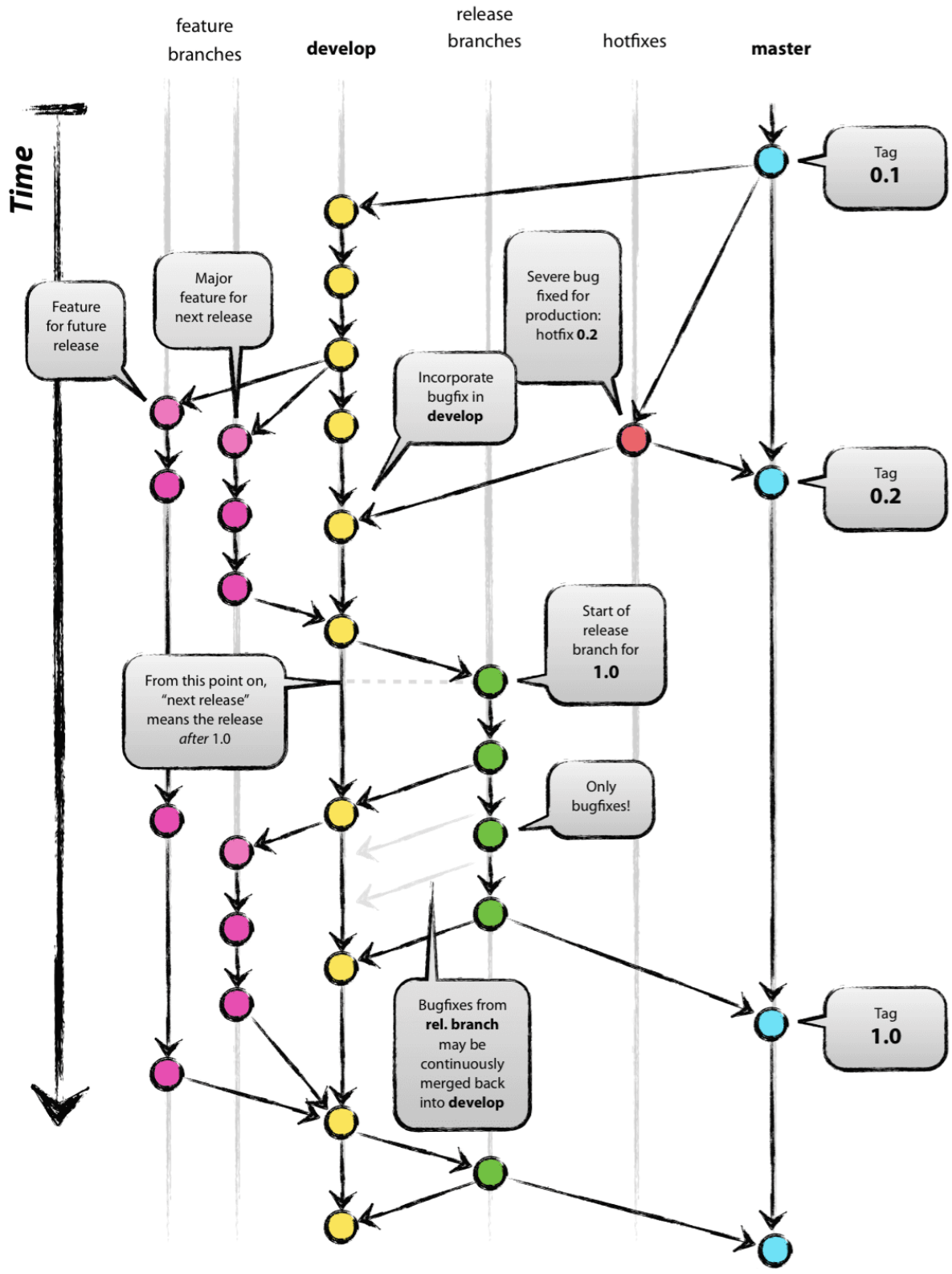
YungchAI workflow manual

1. Git-flow model

- 브랜치 구분

메인 브랜치(master, develop)	보조 브랜치(feature, release, hotfix)
항상 유지됨	일정 기간동안만 유지됨

- 브랜치 설명
    1. master : 제품으로 출시될 수 있는 브랜치
    2. develop : 다음 출시 버전을 개발하는 브랜치
    3. feature : 기능을 개발하는 브랜치
    4. release : 이번 출시 버전을 준비하는 브랜치
    5. hotfix : 출시 버전에서 발생한 버그를 수정 하는 브랜치

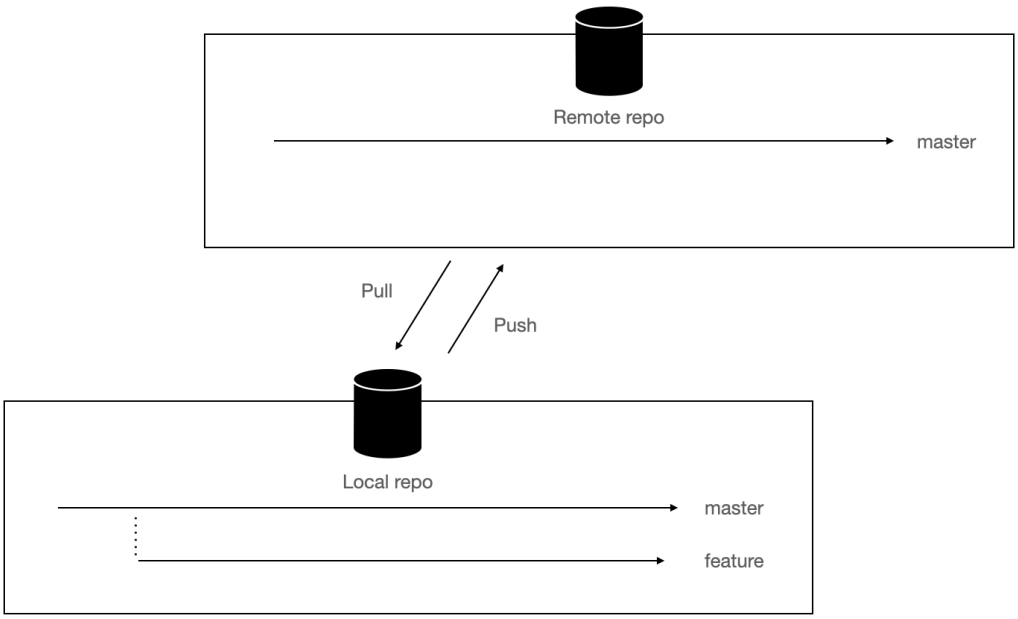


2. Git-flow in YungchAI

가. 프로젝트 구조

root
L ML_PROJ : Remote repo
L KSJ
  L ML_PROJ : Local repo
L KDY
  L ML_PROJ : Local repo
L JYJ
  L ML_PROJ : Local repo

나. Git-flow light



1) Remote repo

- Local repo에서 push한 branch를 QA를 마친 뒤 merge

2) Local repo

- Remote repo를 clone한 것으로 **반드시 feature 브랜치를 생성한 뒤** 개발을 진행해야 함
- feature branch naming convention

  > 💡 feature/[구현하고자 하는 기능 혹은 해결하고자 하는 문제]
  >
  > ex) feature/dev-fasttext or feature/fix-fasttext-bug

- commit message convention ** 자세한 예시는 튜토리얼 참고*

  > 💡 [TAG_NAME] TITLE
  >
  > CONTENT

  - TAG_NAME

    남기고자 하는 commit이 주로 무엇을 한 것인지 명시하며 대문자로 작성. 이때 가능한 동사로 작성

    - 목록

ADD	파일/ 기능 추가
DELETE	파일/ 기능 삭제
MODIFY	파일 수정
FIX	버그 해결
REFACTOR	리팩토링
ETC	그 외 상황은 임의로 작성하되 가능한 동사로 작성

  - TITLE

    commit의 주제를 간단히 작성

    ex) fasttext 개발, 모델 학습시 발생하는 버그 해결 등

  - CONTENT

    commit의 내용을 파일별로 자세히 작성

ex) fasttext.py : fasttext 클래스 구현 등

다. 역할 구분

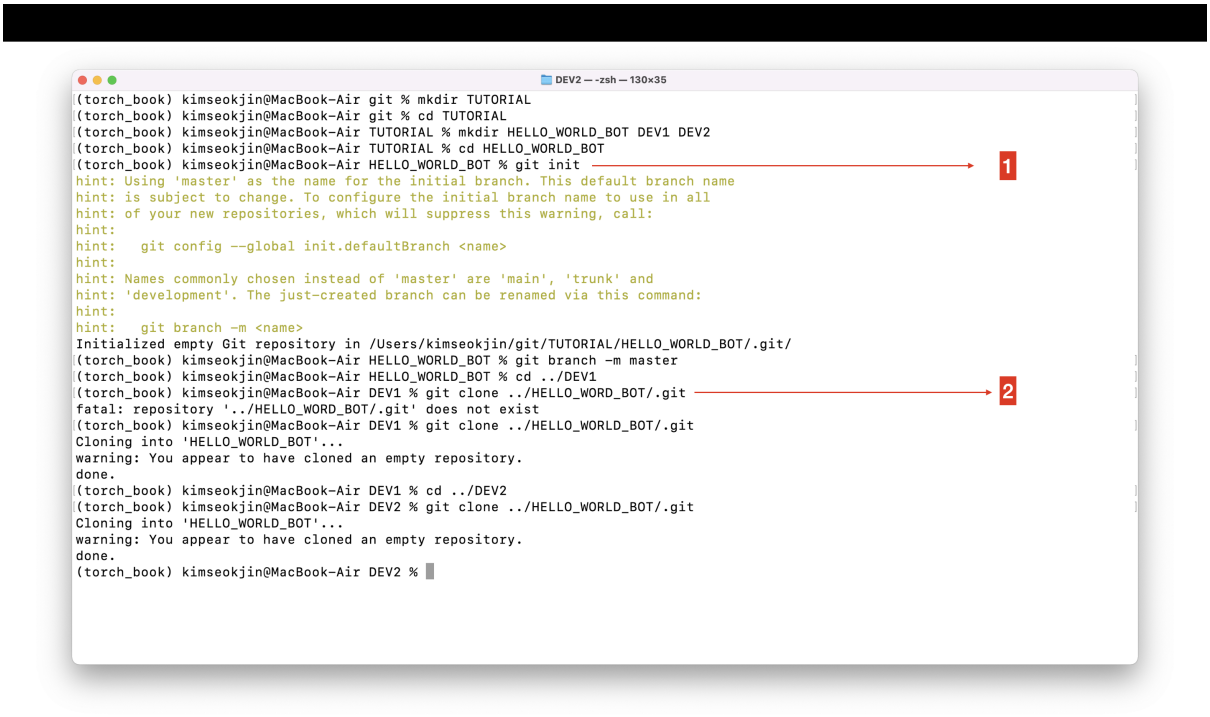
1. 관리자 : 김석진
    - 프로젝트 관리
    - 메인 브랜치(master) 담당
    - 기능 개발 등
2. 개발자 : 정예준, 김동연
    - 보조 브랜치(feature) 담당
    - 기능 개발 등

4. 튜토리얼 - Hello world Bot 개발 👋

가. 프로젝트 구성

아래와 같이 프로젝트를 구성한다.

💡
```
TUTORIAL
L🗂️ HELLO_WORLD_BOT          : Remote repo
L 🗂️ DEV1
   L 🗂️ HELLO_WORLD_BOT       : Local repo
L 🗂️ DEV2
   L 🗂️ HELLO_WORLD_BOT       : Local repo
```



- 주요 명령어에 대한 설명은 다음과 같다.
  1. Remote repo 생성
  2. Local repo 생성

나. Bot 개발

개발자(DEV1)가 local repo에서 bot을 개발한 뒤 remote repo로 push하는 일련의 과정은 다음과 같다.

1. feature branch 생성

   feature branch naming convention에 따라 ‘feature/dev-bot’이라는 branch 생성

```
(torch_book) kimseokjin@MacBook-Air DEV1 % cd HELLO_WORLD_BOT
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git checkout -b feature/dev-bot
Switched to a new branch 'feature/dev-bot'
```

2. 개발

   개발을 진행한 뒤 commit할 수 있도록 index에 staging

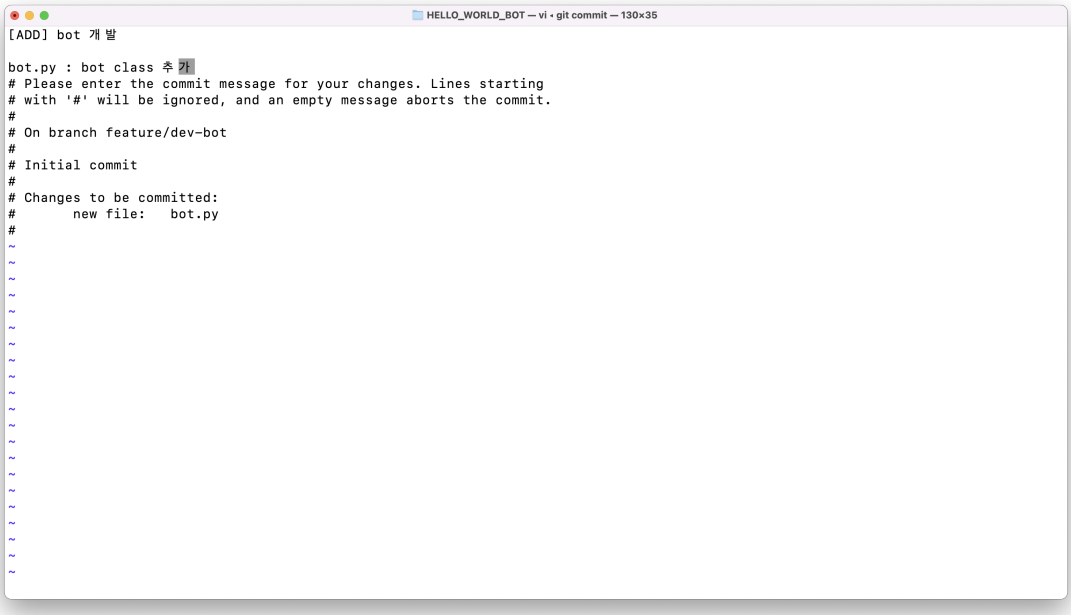
   **** index : commit할 파일들을 구분하기 위한 논리적인 저장소***

```
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % touch bot.py
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git status
On branch feature/dev-bot

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        bot.py

nothing added to commit but untracked files present (use "git add" to track)
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git add .
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git status
On branch feature/dev-bot

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   bot.py
```

3. commit 생성

   commit message convention에 맞게 commit 생성

```
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git commit
```



> 🚨 만약 commit을 잘못 생성했을 경우 `git commit --amend`를 통해 이전 commit 수정

4. origin으로 push

   **** origin : clone한 repo. 즉, remote repo***

```
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git push --set-upstream origin feature/dev-bot
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 249 bytes | 249.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To /Users/kimseokjin/git/TUTORIAL/DEV1/../HELLO_WORLD_BOT/.git
 * [new branch]      feature/dev-bot -> feature/dev-bot
Branch 'feature/dev-bot' set up to track remote branch 'feature/dev-bot' from 'origin'.
```

다. master 브랜치 통합

관리자가 개발자(DEV1)이 작성한 코드를 mater 브랜치에 통합하는 일련의 과정은 다음과 같다.

1. 관리자가 개발자에게 직접 설명을 듣고 QA 진행
2. QA를 마친 뒤 브랜치 통합

   이때 브랜치로 실행한 작업 확인 및 브랜치 관리를 위해 **non fast-forward 방식으로 merge 실행**

```
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git branch
  feature/dev-bot
* master
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git merge --no-ff feature/dev-bot
```

라. 프로젝트 최신화

개발자(DEV2)가 원격 저장소로부터 프로젝트를 최신화하는 일련의 과정은 다음과 같다.

1. 원격 저장소로부터 pull하여 프로젝트를 최신화

```
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % git pull
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false  # merge (the default strategy)
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 229 bytes | 229.00 KiB/s, done.
From /Users/kimseokjin/git/TUTORIAL/DEV2/../HELLO_WORLD_BOT/
 * [new branch]      master          -> origin/master
 * [new branch]      feature/dev-bot -> origin/feature/dev-bot
(torch_book) kimseokjin@MacBook-Air HELLO_WORLD_BOT % ls
bot.py
```

5. 참고

1. 우린 Git-flow를 사용하고 있어요, 우아한 형제들 기술 블로그

   https://techblog.woowahan.com/2553/

2. git flow model, 생활코딩

   https://www.youtube.com/watch?v=EzcF6RX8RrQ

3. 브랜치 통합하기, 누구나 쉽게 이해할 수 있는 Git 입문

   https://backlog.com/git-tutorial/kr/stepup/stepup1_4.html

4. 좋은 git 커밋 메시지를 작성하기 위한 8가지 약속, Uno's blog

   https://djkeh.github.io/articles/How-to-write-a-git-commit-message-kor/