Git

다음 몇가지 문제에 대해 생각 해봅시다.!

“오늘 드디어 프로젝트가 끝난 후 ver 1.0을 출시 했다.
그런데 팀장님이 내일부터 새 기능이 추가된 Ver 1.1
업데이트 준비 하란다.“

- 코드 버전 관리를 효과적으로 해야한다.-

“우리회사 기획자는 기획이 너무 자주 바뀐다. 한달 전 만든
공유기능을 필요 없다고 해서 얼마전 다른 기능으로 바꿨는데
갑자기 공유기능을 다시 만들어 달라고 한다.
코드 백업도 안해 놨는데….어쩌지“

-개발 히스토리를 저장하고 코드 백업기능-

“지금까지는 혼자 개발을 해왔는데 큰프로젝트를 맡다보니
여러사람과 함께 개발해야될 일이 생겼다.
각 기능별로 분담시켜 모듈단위로 개발해야 하는데 효과적으로
코드 공유할 수 있는 방법이 없나?“

-협업을 위한 코드 공유, 관리 tool-

문제 해결책이 뭐가 있을까요?

What is Git

깃(Git /gɪt/[1])은 프로그램 등의 **소스 코드 관리를 위한 분산 버전 관리 시스템이다**. 기하학적 불변 이론을 바탕으로 설계됐고, **빠른 수행 속도에 중점**을 두고 있는 것이 특징이다. 최초에는 리누스 토르발스가 리눅스 커널 개발에 이용하려고 개발하였으며, 현재는 다른 곳에도 널리 사용되고 있다.

깃의 **작업 폴더는 모두, 전체 기록과 각 기록을 추적할 수 있는 정보를 포함하고 있으며, 완전한 형태의 저장소**이다. **네트워크에 접근하거나 중앙 서버에 의존하지 않는다.**

- wikipedia -

버전 관리 시스템

* 버전 관리 시스템 : 프로젝트에 포함된 파일의 변경사항을 추적할 수 있도록 돕는 방법론이나 도구.

  - 사용자가 변경한 모든 내용을 추적 관리 가능한 저장소(repository)를 통해 버전관리.
  - 사용자는 변경된 내용을 저장소에 저장(commit)
  - 하나의 중앙 저장소가 있고 모든 사용자는 이 저장소에 변경사항을 전송.
  - 중앙 저장소에 접근가능한 모든 사람들이 변경사항을 조회가능하고 최신버전을 복사해서 작업할수 있다.

분산 버전 관리 시스템

* 분산 버전 관리 시스템 : 각 사용자가 프로젝트의 전체 이력을 관리할수 있는 자신만의 저장소를 가지고 있다.

  - 기존 버전관리 시스템과 같이 중앙 저장소를 통해 협업 가능
  - 사용자는 변경된 내용을 개인 저장소에 저장(commit)
  - 개인 저장소에 있는 변경된 내용을 중앙 저장소에 저장(Pushing)
  - 중앙 저장소에 있는 최신 내용을 개인저장소로 가져와 최신버전을 유지가능(pulling)
  - 네트워크 연결이 안되있는 상황에서도 저장소에 저장 가능하고 네트워크 연결시 중앙 저장소와 동기화 실시
  - 개인 저장소와 중앙 저장소 모두 프로젝트 전체 이력을 가지고 있어 네트워크 연결이 안되 있는 상황에서도 변경이력 조회 가능

Git의 특징

1. 빠른 속도와 성능
   - git의 모든 동작은 local에서 실행되기 때문에 매번 네트워크 연결을 할 필요가 없어 타 버전 관리시스템에 비해 속도가 빠르다.
2. 분산 작업
   - 중앙 저장소를 통해 한 프로젝트를 여러명의 개발자가 동시에 작업을 분담하여 진행할 수 있다.
3. 데이터의 보장성
   - git의 모든 파일은 암호화 시켜주며, commit시 체크썸이란 검사를 통해 데이터가 유효한지 검사. 또 commit을 할때마다 user name, 시간, 내용등을 기록하여 히스토리 관리가 용의하다.
4. Staging area
   - 작업한 내용을 바로 commit 시키지 않고 index영역에 올려둔 내용만 commit 할수 있다.
5. 브랜치(branch) 모델

Git의 구조

1. Work Space
   - 현재 프로젝트의 작업이 이뤄지는 장소
   - work tree/work directory라고도 한다.

2. index
   - work space에 있는 내용을 내부저장소에 저장하기전 올려두는 중간 단계 공간
   - index공간을 통해서 수정된 파일 중 원하는 파일만 저장소에 저장 시킬수 있다.

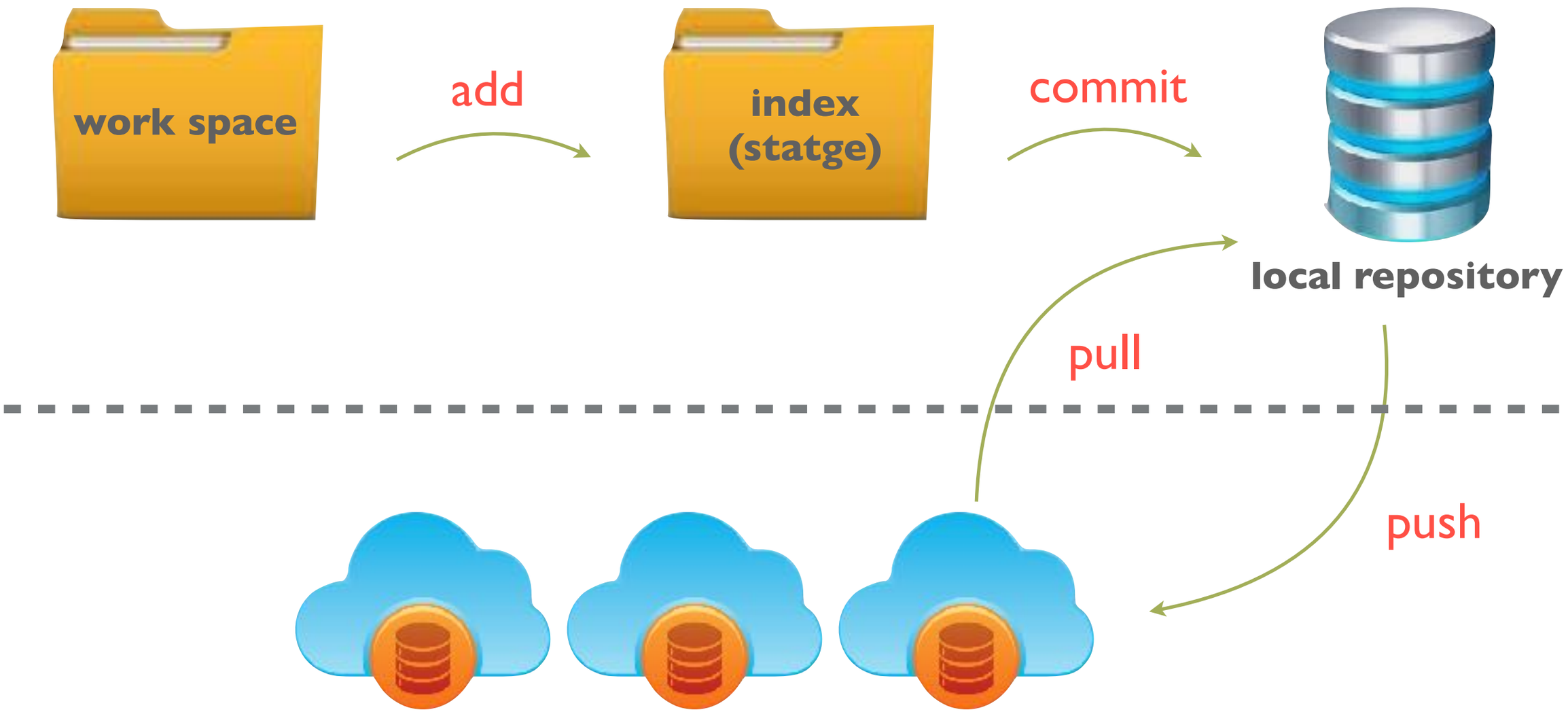
3. 내부 저장소
   - git이 설치된 내부 컴퓨터

4. 외부 저장소
   - 외부에 저장하는 저장소, github이란 서비스도 외부 저장소에 속한다.

Git 명령어

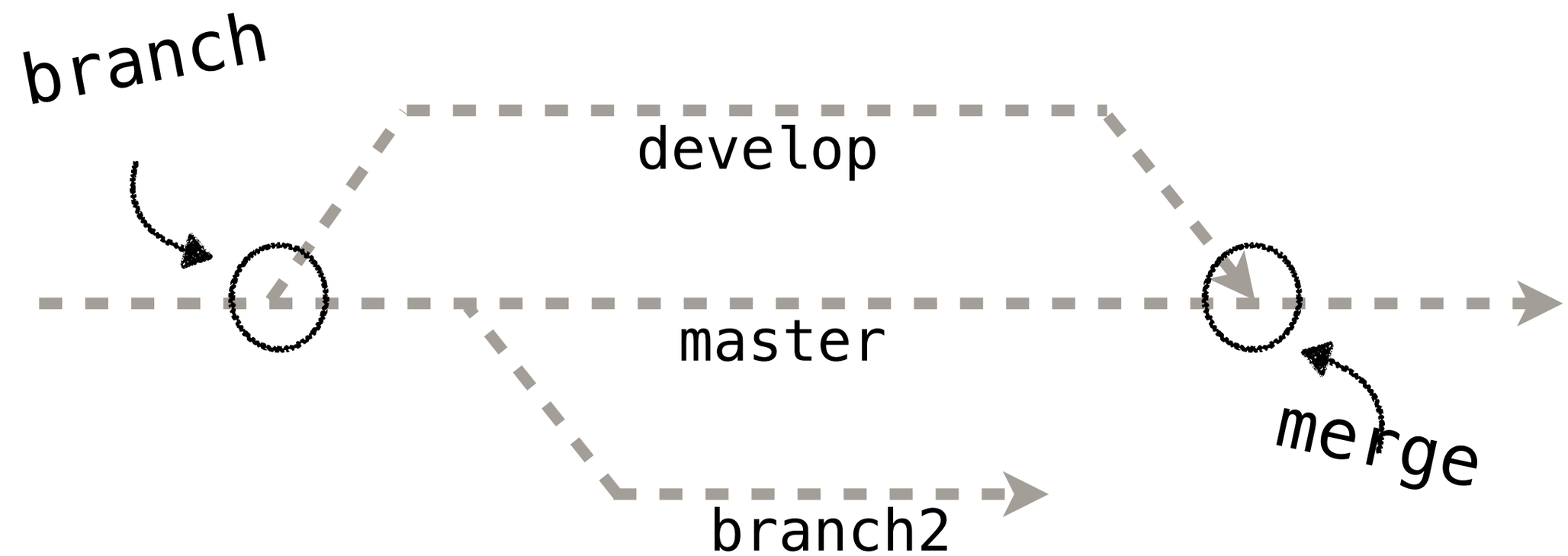
1. init : git 초기화
2. clone : 외부저장소에서 프로젝트 내려받기
3. add : 변경된 내용을 index 영역에 올려두는 명령어
4. commit : index 영역에 있는 내용 내부저장소에 저장
5. status : 현재 git 상태 확인
6. branch : 가지치기
7. checkout : 작업 환경 이동
8. push : 외부저장소에 데이터 저장
9. pull : 외부 저장소에서 최신 변경사항 받아오기

Git 요약



Fast campus

Branch Model



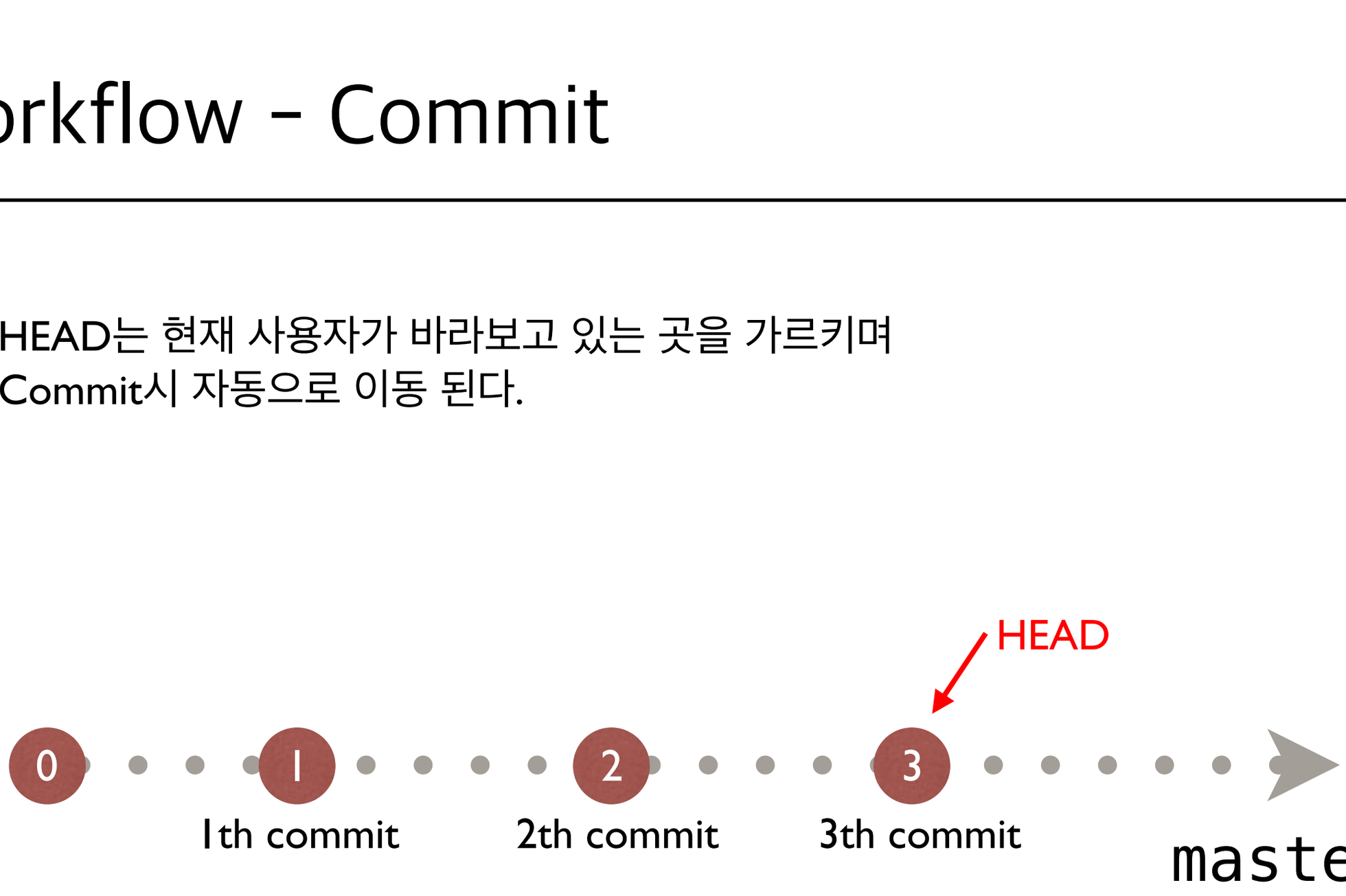
Workflow - Commit

-Commit을 하는 순간 프로젝트의 상태를 내부 저장소에 저장한다.
-Commit하는 순간의 상태를 중요시 여기며 그 상태를 snapshot으로 찍는다고 한다.

“현재 작업 branch는 master"

Workflow - Commit

HEAD는 현재 사용자가 바라보고 있는 곳을 가르키며
Commit시 자동으로 이동 된다.



“현재 작업 branch는 master"

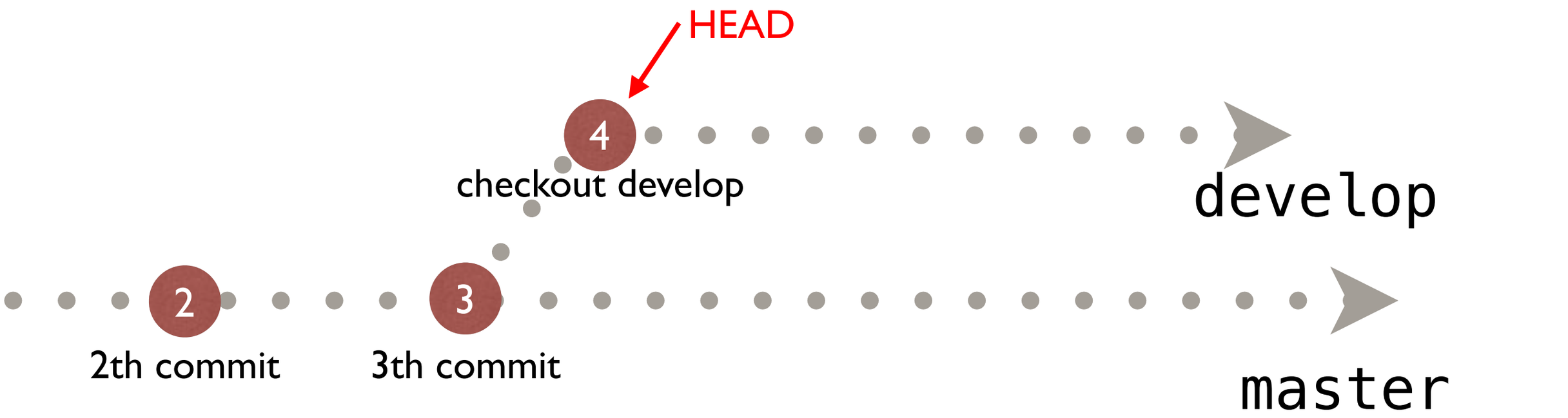
Workflow - Branch

새로운 branch 생성(develop)

“현재 작업 branch는 master"

Workflow - checkout

새로운 branch로 작업 환경 변경 : checkout

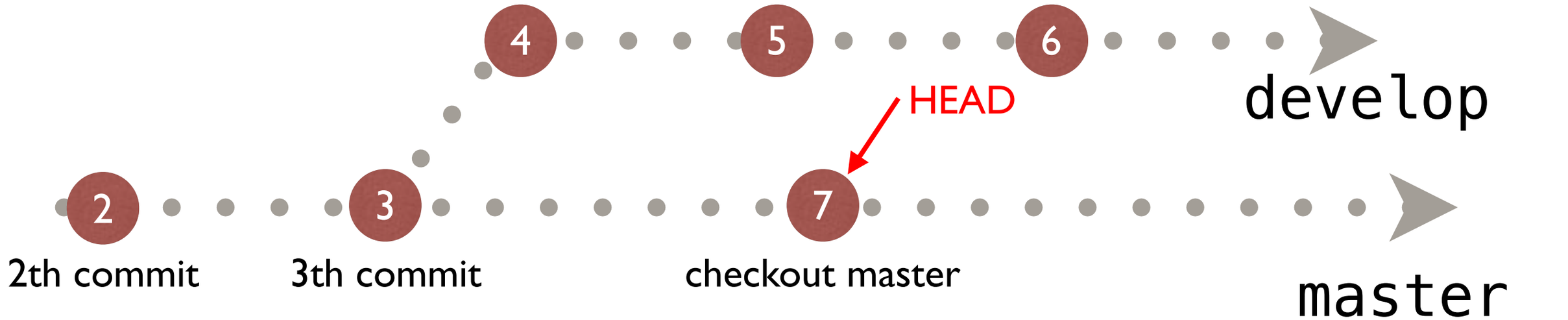


“현재 작업 branch는 develop"

Fast campus

Workflow - checkout

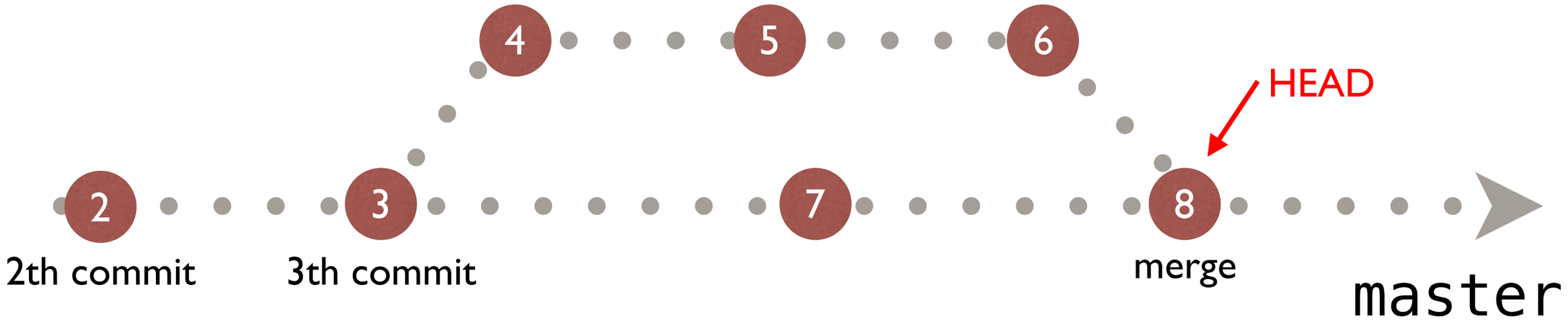
develop branch에서 master branch로 작업환경 변경 : checkout master



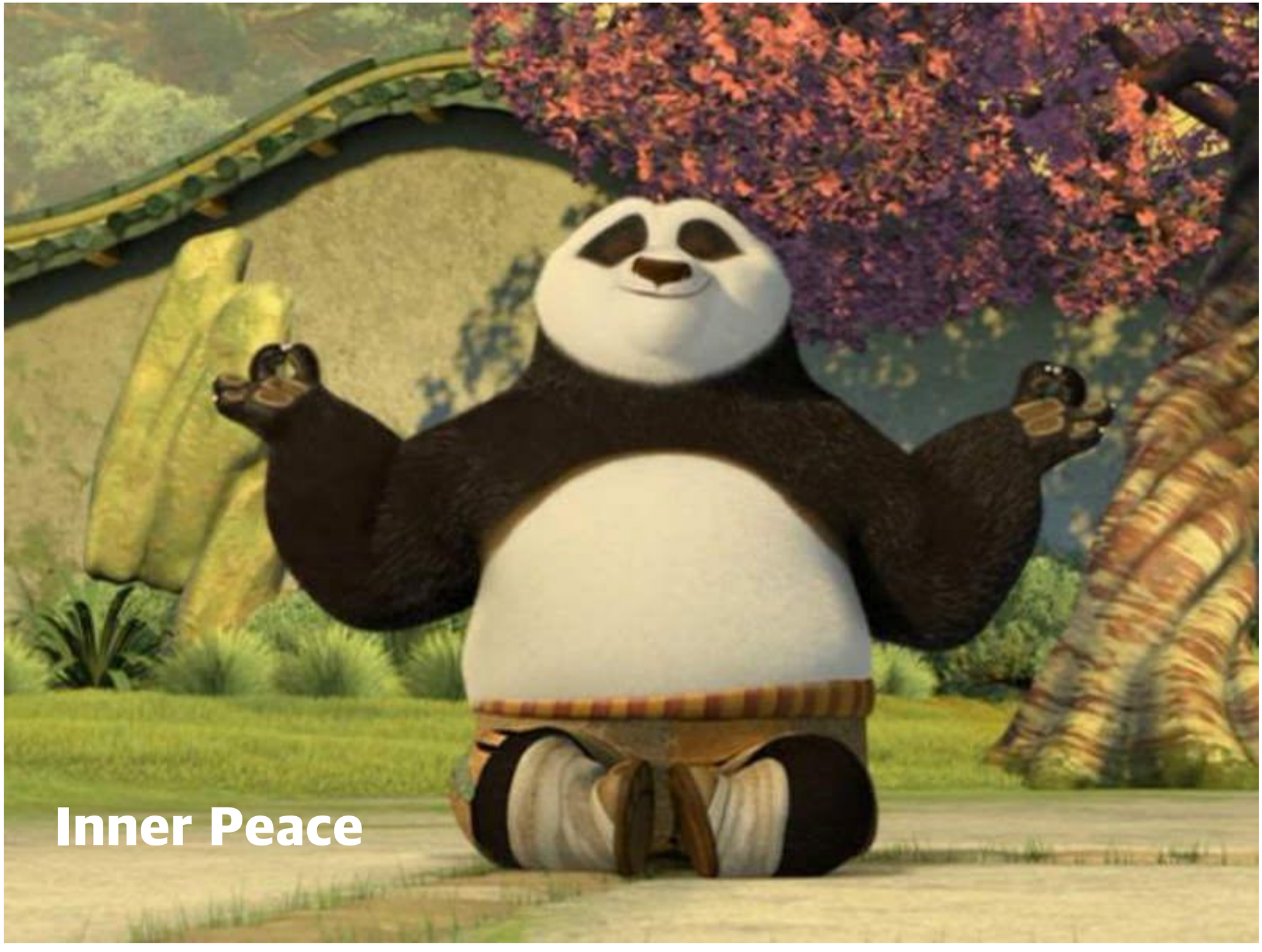
"현재 작업 branch는 master"

Workflow - Merge

develop branch에서 변경된 내용을 master branch와 합치기 : merge develop



"현재 작업 branch는 master"



Fast campus

Setting & init

<설정>

```
user$ git config --global user.name "사용자명"
      :사용자명을 등록합니다.

user$ git config --global user.email "이메일주소"
      :이메일 주소를 등록합니다.
```

<option>

```
user$ git config --global color.ui ”auto"
      :출력결과 색상 활성화하기
user$ git config --global color.ui ”off"
      :출력결과 색상 비활성화하기
```

<초기화>

```
user$ git init
      :내부 저장소 생성.
```

add, commit

<수정된 파일 index로 이동>

```
user$ git add <file>
       : 파일 1개씩 이동
```

or

```
user$ git add .
       :디렉토리 안의 수정된 모든 파일 이동
```

<index에 있는 내용 내부 저장소에 스냅샷으로 저장>

```
user$ git commit –m “this is commit message”
```

status

<현재 git 상태 확인>

user$ git status

```
youngmin-ui-MacBook-Air:startGit knightjym$ git status
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello world.html
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello world.html
#
```

<log확인>

user$ git reflog

```
youngmin-ui-MacBook-Air:startGit knightjym$ git reflog
854f1ce HEAD@{0}: reset: moving to HEAD~
e7a5ba6 HEAD@{1}: reset: moving to HEAD@{1}
854f1ce HEAD@{2}: reset: moving to HEAD~
e7a5ba6 HEAD@{3}: checkout: moving from master to e7a5ba6d55f9324d6e2910e13c109d9b919b6d00
e7a5ba6 HEAD@{4}: commit: first change
854f1ce HEAD@{5}: clone: from git@github.com:wingjoo/startGit.git
```

remove repository

Using Github

https://github.com/

remote setting

<원격 저장소에 설정>

```
user$ git remote add <원격 저장소> <저장소url>

ex) git remote add origin https://github.com/wingjoo
```

<원격 저장소 설정 확인>

```
user$ git remote -v
```

<원격 저장소에 저장(push)>

```
user$ git push <원격 저장소> <branch>

ex) git push origin master
```

clone

<원격 저장소에서 복사본 가져오기>

```
user$ git clone <url>
```

<원격 저장소에서 최신 변경사항 받아오기>

```
user$ git pull <원격 저장소> <branch>

ex) git pull origin master
```

* 원격 저장소의 내용과 내부저장소의 내용이 충돌시
충돌 내용 정리후 다시 **commit**해줘야 한다.

branch

<로컬 저장소의 branch목록 보기>
```
user$ git branch
```

<원격 저장소의 branch목록 보기>
```
user$ git branch -r
```

<새로운 branch 만들기>
```
user$ git branch branch_name
```

<작업트리 이동>
```
user$ git checkout branch_name
```

<합치기 - merge>
```
user$ git merge branch_name
```

<merge된 branch 삭제>
```
user$ git branch -d branch_name
                                        * merge와 상관없이 삭제시 -D 사용
```

checkout

```
<commit log보기>
user$ git reflog

user$ git checkout (commit_id)
      :commit_id 상태로 checkout하기
      :이렇게 checkout된 상태는 branch 위에 올라가 있지 않다.

user$ git checkout -b <name>
      :<name> branch 만들기 checkout한다.

<branch만들고 commit 상태 되돌리기>
user$ git branch <name> (commit_id)
      :name branch만들어서 commit_id내용 넣기

user$ git checkout <name>
```

reset

<commit log보기>
user$ git reflog

<HEAD상태로 이동>
user$ git reset --soft HEAD~
　　:바로 전 HEAD로 이동

user$ git reset HEAD@{2} <file>
　　:2번 HEAD로 이동

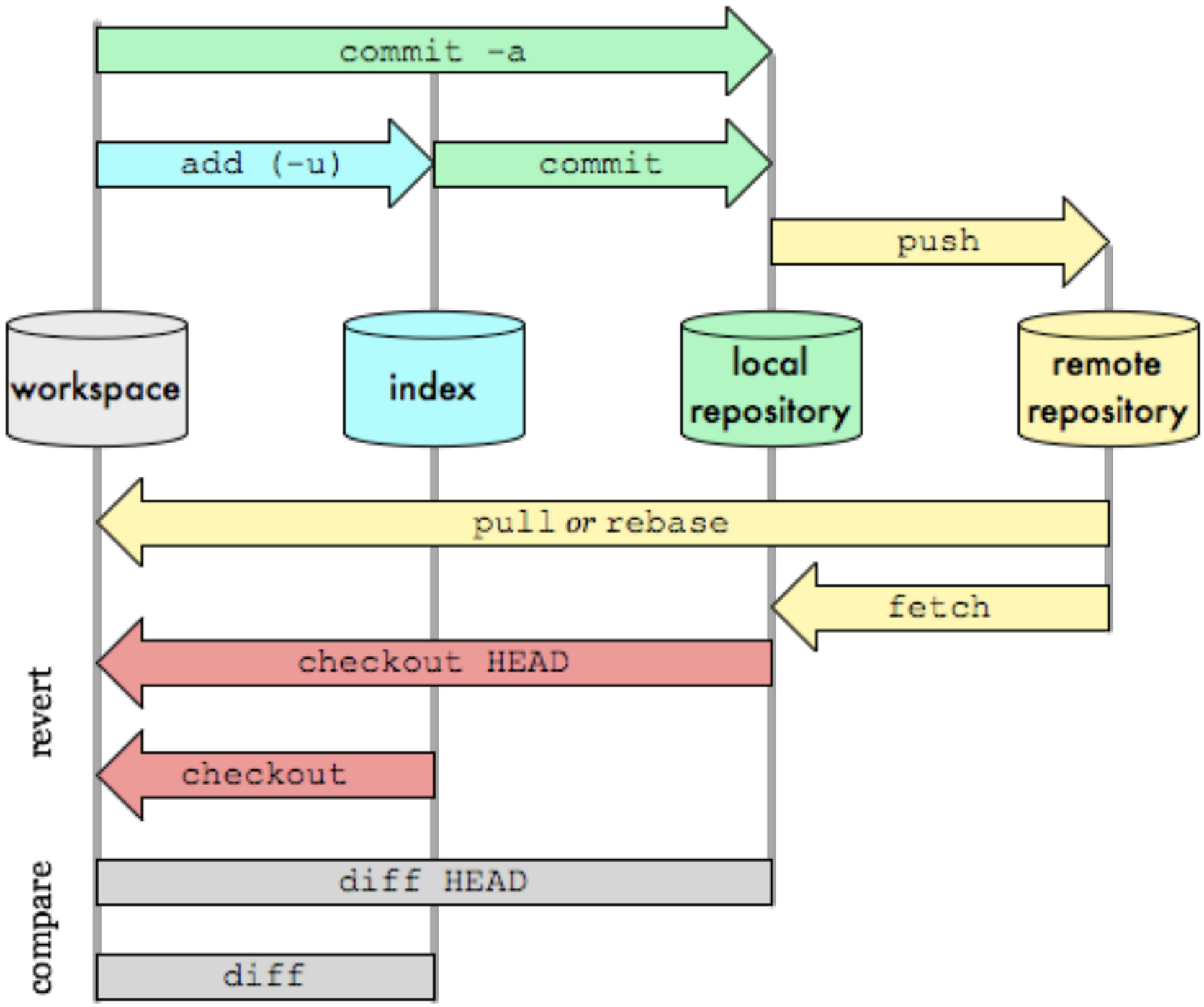
user$ git reset HEAD <file>
　　:스테이지로 이동된 변경사항 되돌리기

정리

1. git은 분산 버전 관리 시스템으로 프로젝트 변경 이력을 관리하는데 도움을 준다.
2. git의 작업이 이뤄지는 저장소의 시점을 work tree라고 하며 branch생성을 통해서 다양한 시점의 작업이 가능하다.
3. branch를 현재 작업tree로 변경하기 위해서는 “checkout”명령을 통해서 변경가능하다. ($git checkout branch_name)
4. index영역이 존재해서 work tree에서 작업한 내용을 곧바로 내부 저장소에 저장하지 않고 선별적으로 저장할 내용을 add 할 수 있다.
5. commit명령을 통해 index에 있는 변경 내용을 내부 저장소에 저장 할 수 있다.
6. 외부 저장소를 이용 중앙 집중 저장소로 사용할 수 있다.
7. 대표적인 외부 저장소로는 github이란 서비스가 있고 설정을 통해 외부 서버를 만들 수도 있다.
8. push & pull을 통해 외부 저장소에 데이터를 저장하고 불러올수 있다.
9. clone 명령어를 통해서 외부저장소를 내부 저장소로 복제 할수 있다.

Git Data Transport Commands

http://osteele.com



Fast campus

참고자료

<간편 안내서>
http://rogerdudler.github.io/git-guide/index.ko.html

<GUI로 branch만들어 보기>
http://learnbranch.urigit.com/

<Pro Git 번역>
http://git-scm.com/book/ko/v2