git

Git…?

git

git

Git...?

git is currently the most popular implementation of a distributed version control system.

우리가 가지고 있는 문서의 버전을 (분산) 관리하는 시스템

Project1.pptx
Project2.pptx
Project3.pptx
Project4.pptx
Project5.pptx
Project6.pptx
Project final.pptx
Project final-1.pptx
Project final-2.pptx
Project final final.pptx
Project final final final.pptx
Project 진짜 final.pptx
… …

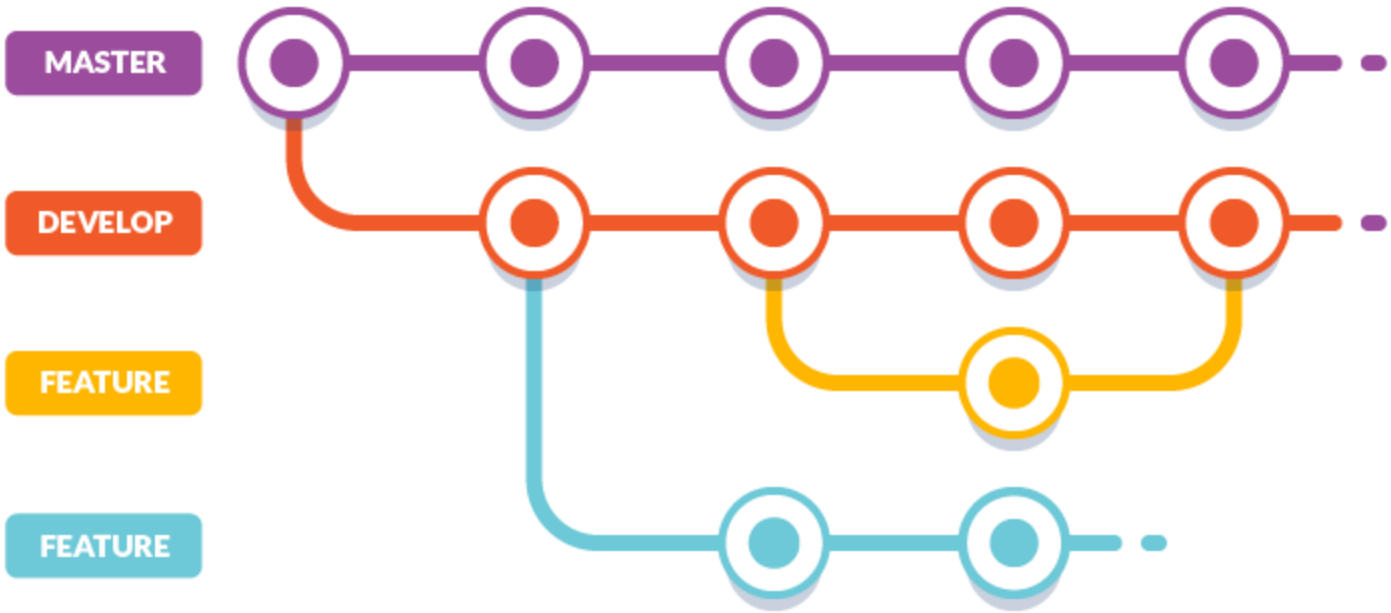
같은 이름으로 관리한다면...?

Project.pptx 2017.06.01 17:00:00
Project.pptx ……
Project.pptx ……
Project.pptx
Project.pptx
Project.pptx
Project.pptx
Project.pptx
Project.pptx
Project.pptx
Project.pptx
……

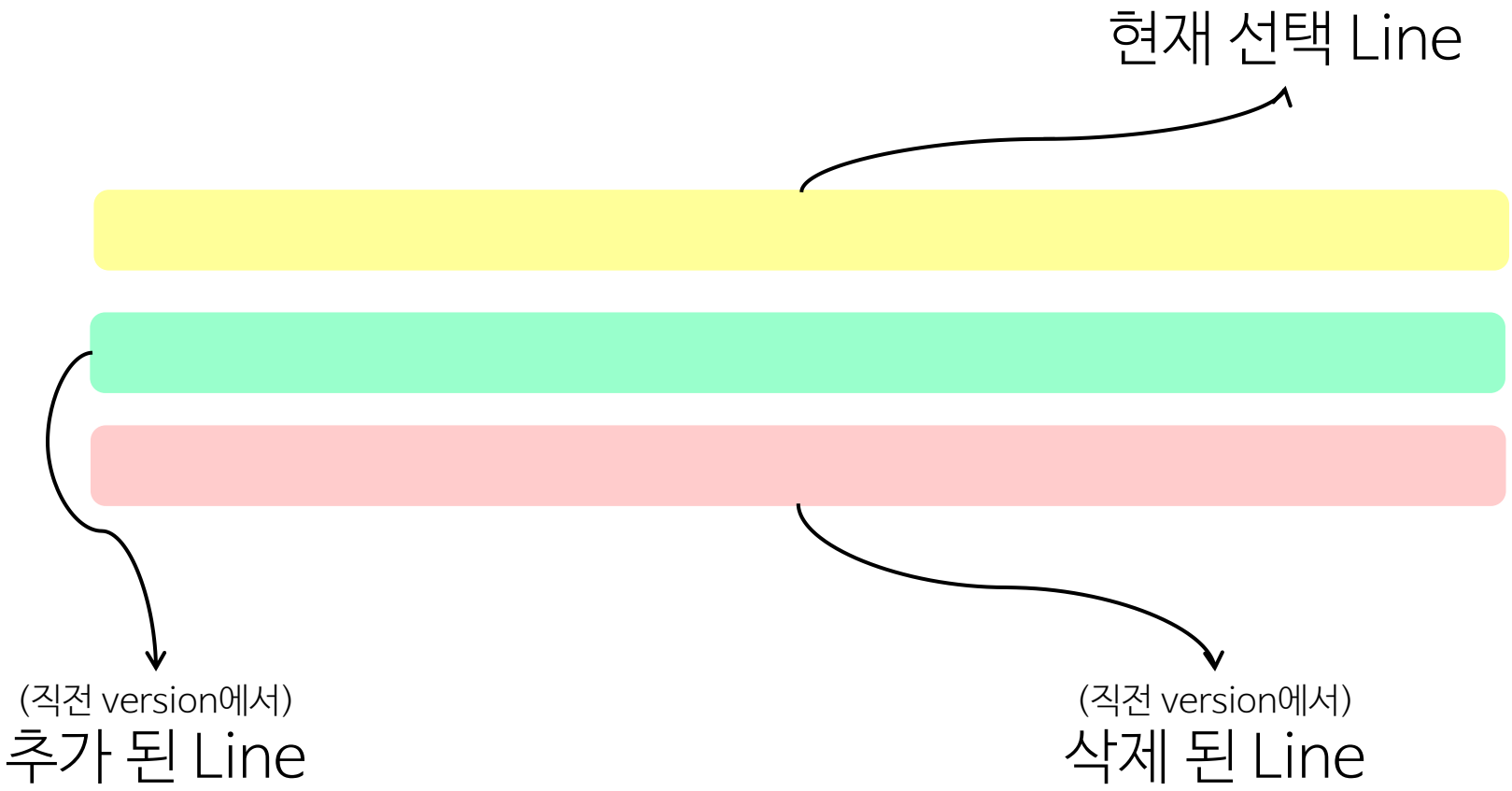
시간의 흐름에 따라 표현


MASTER


MASTER
FEATURE
FEATURE


MASTER
DEVELOP
FEATURE
FEATURE

어떻게 내용이 관리 될까?


현재 선택 Line
(직전 version에서)
추가 된 Line
(직전 version에서)
삭제 된 Line

```
+- (RACSignal *)enqueueRequest:(NSURLRequest *)request fetchAllPages:(BOOL)fetchAllPages;
+
 // Enqueues a request to fetch information about the current user by accessing
 // a path relative to the user object.
 //
@@ -241,11 +255,13 @@ - (id)initWithServer:(OCTServer *)server {
   NSString *userAgent = self.class.userAgent;
   if (userAgent != nil) [self setDefaultHeader:@"User-Agent" value:userAgent];

-  self.parameterEncoding = AFJSONParameterEncoding;
-  [self setDefaultHeader:@"Accept" value:@"application/vnd.github.beta+json"];
-
   [AFHTTPRequestOperation addAcceptableStatusCodes:[NSIndexSet indexSetWithIndex:OCTClientNotModifiedStatusCode]]
-  [AFJSONRequestOperation addAcceptableContentTypes:[NSSet setWithObject:@"application/vnd.github.beta+json"]];
+
+  NSString *contentType = [NSString stringWithFormat:@"application/vnd.github.%@+json", OCTClientAPIVersion];
+  [self setDefaultHeader:@"Accept" value:contentType];
+  [AFJSONRequestOperation addAcceptableContentTypes:[NSSet setWithObject:contentType]];
+
+  self.parameterEncoding = AFJSONParameterEncoding;
   [self registerHTTPOperationClass:AFJSONRequestOperation.class];

   return self;
```

이걸 꼭 해야 하는가?

Project
난이도

Git System
난이도

Project
난이도

Git System
난이도

어떻게 사용 되는가?

Command List

Git 주요 개념

-저장소(Repository)

사용자가 변경한 모든 내용을 추적하는 공간.

-브랜치(Branch)

독립적으로 어떤 작업을 진행하기 위해 분기시키는 것

-작업 트리

저장소를 바라보는 자신의 현재 시점.
소스 코드, 빌드 파일, 단위테스트 등 모든 파일

-합치기(Merger)

분기된 브랜치를 합치기 위한 작업.

-커밋(commit)

내 로컬 저장소에 변경 내역을 저장하는 것

-패치(fetch)

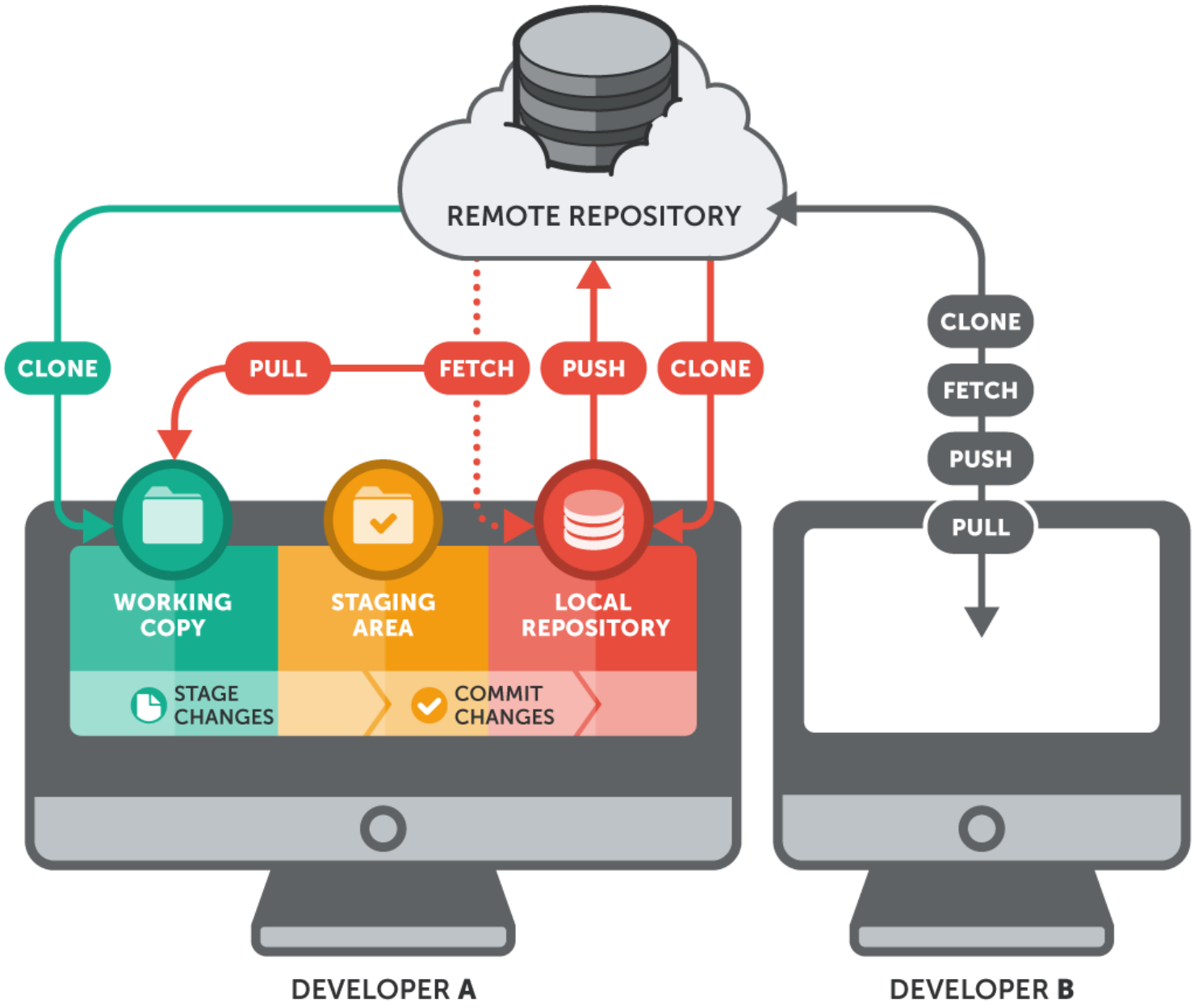
내 로컬 저장소에 서버 저장소 변경 이력을 다운 받는 것

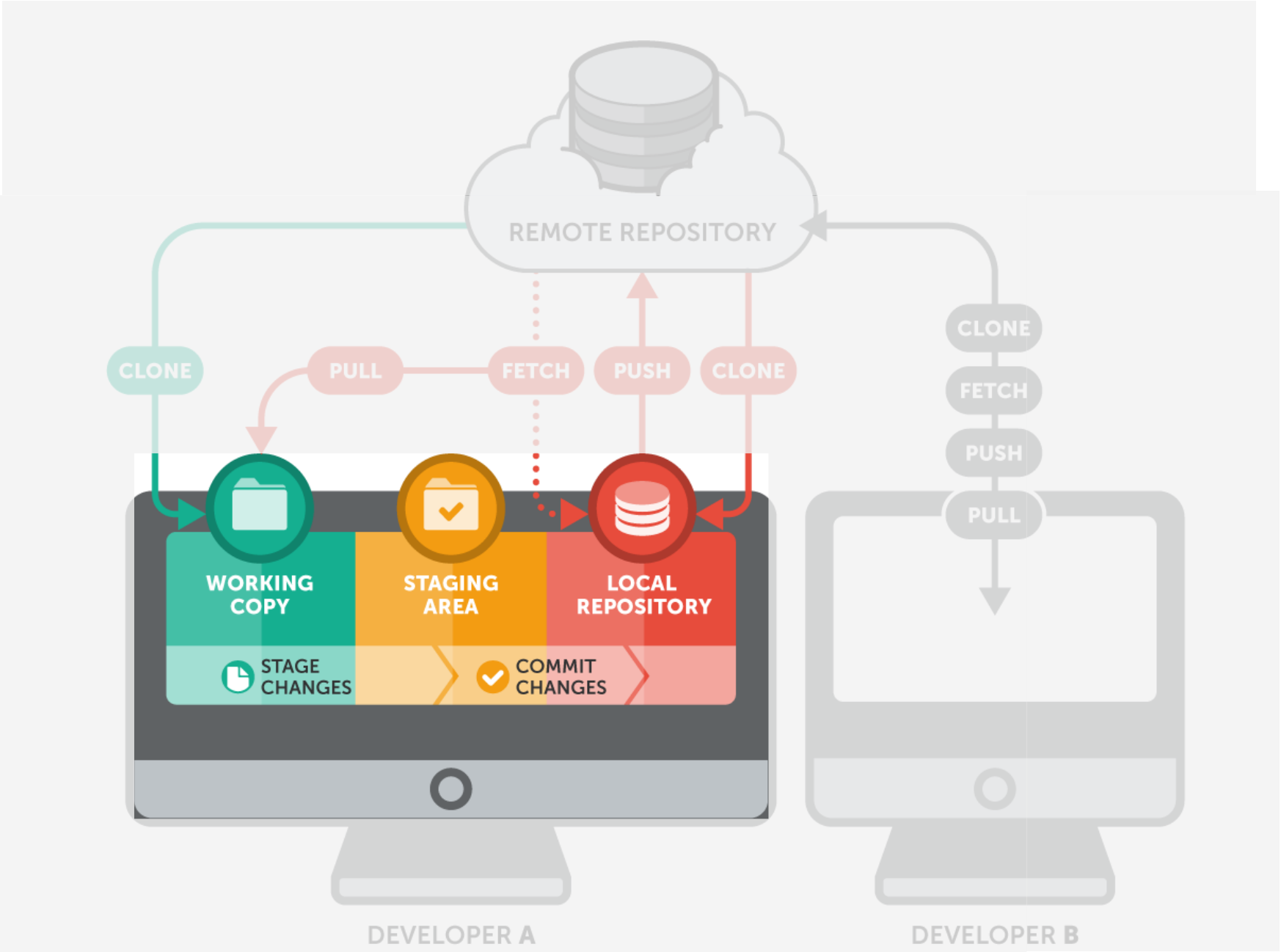
-푸시(push)

로컬 저장소의 내용을 서버 저장소에 전송하는 것

-풀(pull)

내 소스코드에 로컬 저장소 기록을 변경하는 것


REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
CLONE
FETCH
PUSH
PULL
DEVELOPER B


REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
CLONE
FETCH
PUSH
PULL
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
DEVELOPER B

Repository flow

Work Space

↓

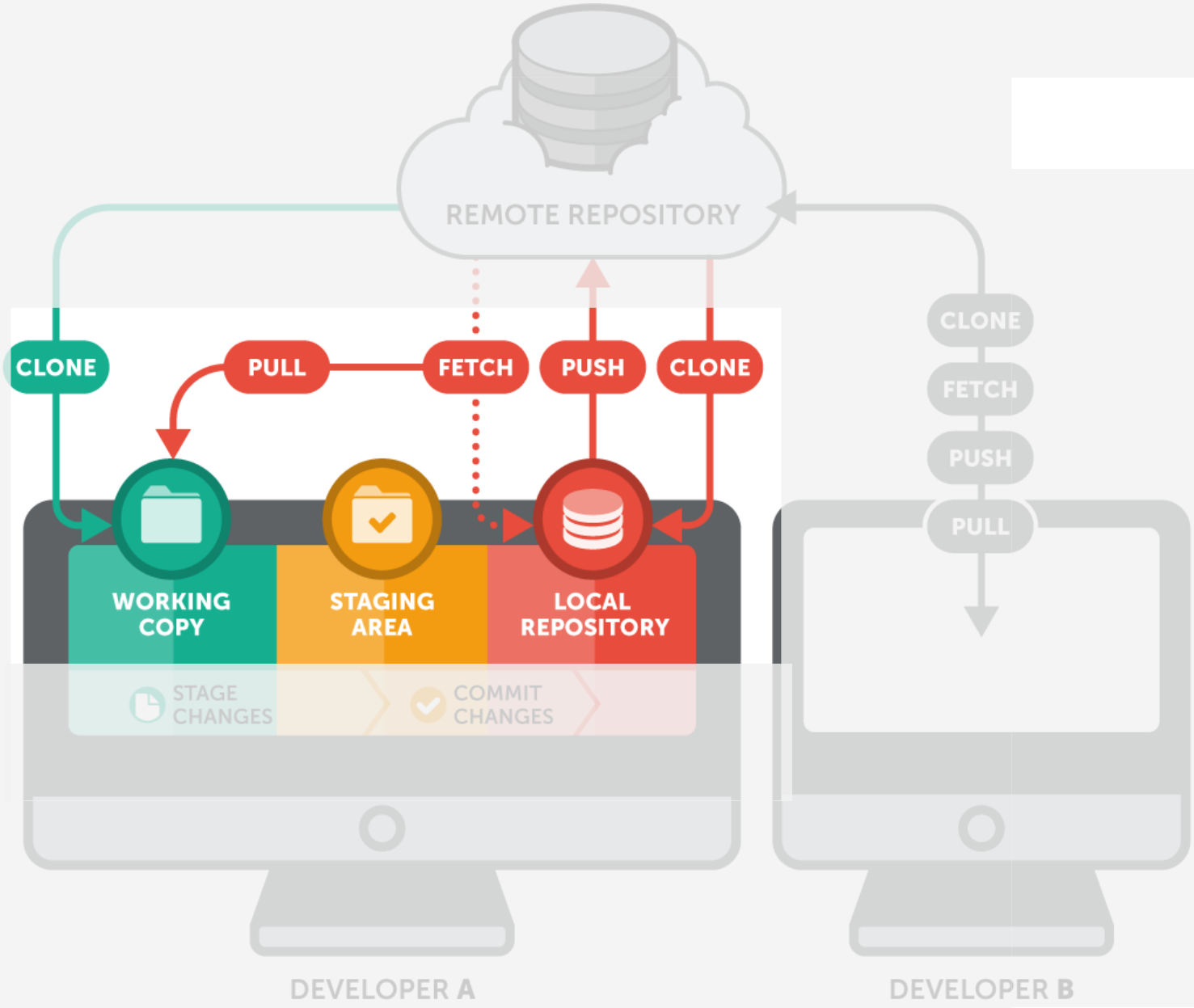
Stage Area

↓

Local Repository

↓

Remote Repository


REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
CLONE
FETCH
PUSH
PULL
DEVELOPER B

Command flow
(local → remote)

Git init

↓

Git add

↓

Git commit

↓

Git push

Command flow
(remote → local)

Git clone	Git pull
복제	현재 상태를 가져오기

어떻게 사용하는가?

CLI (Command Line Interface)
&
GUI (Graphic User Interface)

CLI(Command Line Interface)

```
student@M5018 MINGW64 /c/dev/workspace (master)
$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Git Settings

Git init

↓

Git remote add ⟨alias⟩
https://github.com/⟨id⟩/⟨remote_name⟩.git

↓

Git config --global user.name “name”
Git config --global user.email “email address”

↓

Git add

create & clone

create new repository	*git init*
clone local repository	*git clone /path/to/repository*
clone remote repository	*git clone username@host:/path/to/repository*

add & remove

add changes to INDEX	*git add <filename>*
add all changes to INDEX	*git add **
remove/delete	*git rm <filename>*

commit & synchronize

commit changes	*git commit -m "Commit message"*
push changes to remote repository	*git push origin master*
connect local repository to remote repository	*git remote add origin <server>*
update local repository with remote changes	*git pull*

branches

create new branch	*git checkout -b <branch>* *e.g. git checkout -b feature_x*
switch to master branch	*git checkout master*
delete branch	*git branch -d <branch>*
push branch to remote repository	*git push origin <branch>*

merge

merge changes from another branch	*git merge <branch>*
view changes between two branches	*git diff <source_branch> <target_branch>* *e.g. git diff feature_x feature_y*

tagging

create tag	*git tag <tag> <commit ID>* *e.g. git tag 1.0.0 1b2e1d63ff*
get commit IDs	*git log*

restore

Tip

Want a simple
git-client for

GUI (Graphic User Interface) 방식으로 하고 싶다면

SourceTree

Reference

- 누구나 쉽게 이해 할 수 있는 Git 입문 (원숭이도 한다는..)
  https://backlogtool.com/git-guide/kr/

- 생활코딩 Git (GUI)
  https://opentutorials.org/course/1492

- 생활코딩 지옥에서 온 Git
  https://opentutorials.org/course/2708

- Git Practice
  http://learnbranch.urigit.com/

- Github Flow
  https://ujuc.github.io/2015/12/16/git-flow-github-flow-gitlab-flow/

- Why the 'Git' name?
  https://m.blog.naver.com/PostView.nhn?blogId=onlysilence&logNo=114191637&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F

Reference

- Git Workflow
  https://buddy.works/blog/5-types-of-git-workflows

- Git Setting
  http://insanehong.kr/post/create-repository/

- Git Workflow Diagram
  https://www.git-tower.com/learn/git/ebook/en/command-line/remote-repositories/introduction