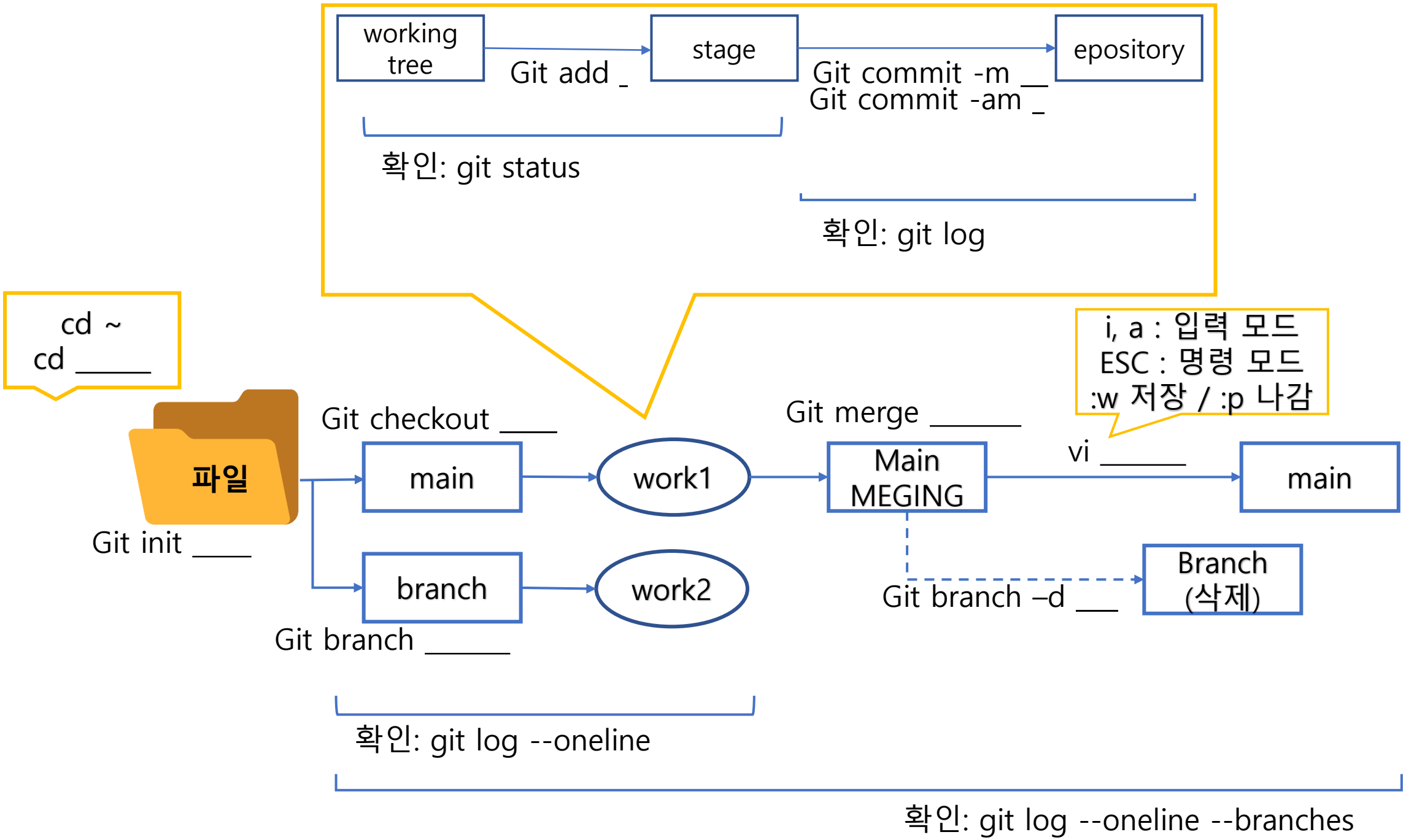
Git Hub

1~2일차 실용요약본


working tree
Git add _
stage
Git commit -m __
Git commit -am _
epository
확인: git status
확인: git log
cd ~
cd _____
i, a : 입력 모드
ESC : 명령 모드
:w 저장 / :p 나감
Git checkout ____
Git merge _____
vi _____
파일
main
work1
Main
MEGING
main
Git init ____
branch
work2
Branch
(삭제)
Git branch –d ___
Git branch _____
확인: git log --oneline
확인: git log --oneline --branches

HTTPS

Cd ~ → git init _ → cd _____ → 문서 작업 → git remote add 서버명 HTTPS →
git push (-u) 서버명 브런치명

SSH

Cd ~ → ssh–keygen(엔터 연타) → cd .ssh → ls –la → cat id_rsa.pub(복사) →
github에서 셋팅, 'ssh and GPG keys' 클릭, 공개키 추가 → 서버의 ssh 복사 →
Git remote add (서버명) ssh → git push (서버명) 브런치명 → yes or no → git pull (서버명) 브런치명

목록 확인 git remote –v
삭제 git remote reove ____

Clone 첫트만 이용, 모두 다운(git remote add + git pull)

cd~ → git clone ssh (파일명) → cd (파일명) → git checkout (브런치명)

Pull 덮어쓰기
Fetch 미리보기(?)

저장된 파일 확인 ls
저장소 확인 git log
(log로 브런치 보고 넣기)

dtype
dtype('O') = 오브젝트
dtype('int64') = 숫자타입

info()
데이터프레임의 기본정보 확인

range()
Ex) range(0, 101, 5)
└→0~100까지 5의 함수 출력

astype()
괄호 안에 int 등을 사용하여 타입변경

Import 사용할 모듈 **as** 약어

from 모듈 **import** 기능
모듈의 일부만 가져오는 방법

자료 참고 사이트:
강사님이 주신 자료: https://www.utc.fr/~jlaforet/Suppl/python-cheatsheets.pdf
자료 검색: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
판다스 : https://ml-ko.kr/homl2/tools_pandas.html
넘파이 : https://ml-ko.kr/homl2/tools_numpy.html

모듈 pandas

-Series / sr : (인덱스 값으로 부여.)
균일한 유형의 배열로 표시된 1차원 데이터
sr = pd.Series({'a':1, 'b':2, 'c':3})
sr= pd.Series(data, index = ['1', '2', '3', '4'])

-DataFrame / df : (시리즈 다발 묶음.)
가변적인 테이블 형식의 2차원 데이터
딕셔너리(키)-> 데이터프레임(컬럼) 전환
df= pd.DataFrame(data)
└→괄호 안에 _____=['1', '2'] 식으로 인덱스와 컬럼 지정 가능

from pandas **import** Series, DataFrame

-행을 선택 :
object.loc[인덱스명] <-인덱스로 선택
object.iloc[정수 인덱스명]<-정수로 선택
인덱싱과 슬라이싱으로도 선택가능.
(다수의 인덱스 경우 [['a']])

-열을 선택 :
object[컬럼명] 또는 object.컬럼명

-원소 선택 :
object.loc[행인덱스,컬럼인덱스]
object.iloc[정수인덱스, 정수인덱스]
Ex) df.loc['서준':'우현', '수학':'음악'] <-슬라이싱
Ex) df.loc['인아',['수학','체육']] <- 인아의 수학,체육만 출력

-열 추가 : object['새로운 컬럼명'] = 값
-행 추가: object.loc[새로운 행명] = 값 또는 배열
-수정 : df.loc['a', 'b']=c / df.loc['d', ['a', 'b']]=c
-행과 열의 위치 변경 : object.transpose(), object.T
Ex) df_t = df.T

모듈 pandas # inplace = True 괄호안에 넣으면 영향 줌.

-카피를 원할 경우:
df_copy=df.copy()

-인덱스 초기화 :
object.reset_index()

-행의 자료 삭제:
객체.drop(행인덱스 또는 배열, axis= 0)

-인덱스를 재배열 :
object.reindex(인덱스리스트)
ex) 리스트 작성 후 사용.

-인덱스 기준으로 정렬:
object.sort_index()

-칼럼 값을 기준으로 정렬 :
object.sort_values(by=컬럼, ascendig=False)
# ascending=False
내림 차순/ 아무것도 안적으면 오름차순
by=['a','b'] a 점수가 같을 시 b 점수로 결정.

-특정 컬럼을 인덱스로 설정 :
object.set_index(컬럼명)

-컬럼명 지정:
df.columns= [컬럼명, ...]

-인덱스명과 컬럼명을 일부만 수정하고자 할 경우 :
object.rename(index={old:new, ...})
object.rename(columns={old:new, ...})

-컬럼의 정보만 추출: with.슬라이싱
df = titanic[['age', 'fare']]
df = titanic.loc[100:200, ['age','fare']]
# df.head() : 처음 5개만 보여줌
# df.head(10) : 10개 보여줌
# df.tail() : 뒤에서 5개만 보여줌
-그 중 정보만 추출: df.loc[df['a'] == 'b', :]

-시리즈 연산이 가능해짐(개꿀)
Ex) print(student1+100)
Ex) div= student1 / student2
연산식에 Nan이 존재하면 연산의 결과는 Nan

모듈 pandas

-딕셔너리를 데이터프레임으로 전환
df1= pd.DataFrame(data)

-외부 파일 읽어오기
File_path = '위치'
Df= pd.read_파일형태(File_path)
df = pd.read_파일형태(file_path)
└→ 괄호에 식을 넣어 인덱스/컬럼 수정:
# 지정 없을 시 첫 행이 열이름.
Index=False #인덱스열 없이 모두 데이터 처리.
Header=None # 컬럼열 없이 모두 데이터 처리.
index_col= 'a' # 인덱스를 a 컬럼 값으로 지정.
Skiprows=1 #1행부터 컬럼 지정
Names= ['a','b'] #이름 지정

-파일 저장
file = pd.ExcelWriter('경로')
df1.to_excel(file, sheet_name= 'sheet1') #이름 지정
file.save() #저장

-데이터프레임의 모든 원소에 함수를 적용
.applymap()
-시리즈의 개별원소에 함수 매핑
시리즈.apply(매핑함수)
apply 한 컬럼 / applymap 데이터프레임(각 요소 별로) 적용
-데이터프레임 각 원소에 함수 매핑
데이터프레임.apply(매핑함수, axis=0)
-데이터프레임 객체에 함수매핑
데이터프레임.pipe(매핑함수)

-lambda 함수 : 두 수를 더 하는 함수
한줄로 쓰는 if 문 (X가 참일 때 적용 받음)

-열 분리 : 하나의 열에 다수의 정보를 가지고 있는 경우 각 정보를 분리 시키는 작업
Ex) dates=df['연월일'].str.split('-')
2018-07-02로 된 자료를 '-'를 넣어 제거 분리

-데이터를 골라내어 함수로 정제
시리즈.isin(리스트)
#필터링 : bool 리스트, isin(리스트)

모듈 pandas

-시리즈의 문자열 인덱싱
 시리즈.str.get(인덱스)

-시리즈 만들기
 sr1 = pd.Series(['e0', 'e1', 'e2', 'e3'], name='e')

-데이터프레임 연결 : pd.concat(데이터프레임 리스트)
 #join='inner'인덱스가 같은 행만 결합

-두개의 데이터프레임을 결합
데이터프레임1.join(데이터프레임2, how= ' ')
 # 인덱스 설정되어 있어야함. 인덱스 기준으로 정렬.
pd.merge(df1, df2) #중복되는 데이터만 보여줌
 #how='outer' : 모두 보여줌
 #how='inner' : 처음 컬럼을 기준으로
 #how='left' : df1 다 +df2 겹치는 것만 가져옴.
 #how='right' : df2 다 +df1 겹치는 것만 가져옴.
 ex) pd.merge(df1.loc[df1['price']< 50000, :], df2)

-사용자 정의 함수를 그룹 객체에 적용 :
 그룹객체.agg(매핑함수)
 그룹객체.agg([함수1,함수2,..]) #여러 객체

-그룹 연산 데이터 변환
 group객체.transform(매핑함수)

-그룹함수로 각 데이터에 연산이 가능
 grouped['age'].transform()

-그룹 객체 필터링
 그룹객체.filter(조건 함수)

그룹 객체에 함수 매핑
 그룹객체.apply(매핑함수)

-피벗 테이블
pd.pivot_table(데이터프레임, 열, 컬럼, 집게 함수, ...)

모듈 pandas

-피벗 테이블

pd.pivot_table(데이터프레임, 열, 컬럼, 집게 함수, ...)

```
Ex)pdf1= pd.pivot_table(df, #피벗에 사용될 데이터 프레임
          index=['class','sex'],  #그룹에 사용될 컬럼
         columns='survived',  #열 위치에 들어갈 컬럼명
         values= ['age','fare'],  #데이터로 사용될 컬럼
         aggfunc=['min','max']) #어떤 값으로 이걸 쓸거냐
```

모듈 numpy # 배열의 처리

-생성
np.arange(시작점, 끝점) #range함수와 같음.
np.empty(n, n) #공간을 할당
np.zeros(n) # 0을 n개 만큼 배열
np.ones(n) #1을 n개만큼 배열

-난수 생성
rand: 0부터 1사이의 균일분포(정규분포)로 난수 생성
randn: 기댓값이 0이고 표준편차가 1인 가우시안 표준
정규 분호를 다르는 난수를 생성
randint: 균일 분포의 정수 난수
ex) np.random.randint(10, size= 2)
└→ 10까지 (size=) 2개 뽑음
random.seed() : 동일한 난수 발생
shuffle() : 자료 섞음.

-샘플링 함수 / choice 함수
#np.random.choice(data, size=None, replace=True, p=None)
#np.random.choice(5, 5, replace=True)
True = 같은값도 출력 / False = 겹치지 않게 출력
replace=True 반복허용 / p는 확률, 1을 맞게 나눠줘야함.

-확인
shape() #행, 열 수
size() # 전체 데이터 수
reshape() #평탄화?
└→ reshape(4,5) #2차원으로 바꾼 배열(4행, 5열)

-차원 변경 :
array, reshape(행, 열)
np.array(리스트) # 리스트를 array로 변경

-계산 #모두 : describe
평균값: mean, 최대값: max, 최소값: min,
중간값: median, 표준편차: std, 분산: var,
상관계수: df[컬럼리스트].corr()
ex) df.median()
ex) df['컬럼명'].median()

-데이터에 고유값 종류 목록
.unique

모듈 Matplotlib import matplotlib.pyplot as plt

-여러 자료를 이용하여 그릴 시
```
fig =plt.figure(figsize=(n, n))        # 전체 가로/세로
ax1 = fig.add_subplot(행, 열, 순서)
```

-출력
```
plt.show()
```

-그리기
```
Plot()                                 #그래프
plot(kind='bar')                       #막대 그래프
plot(kind='hist')                      #히스토그램
plot(x='a', y='b', kind='scatter')     #산점도
plot(kind='box')                       #박스 플롯
```

-Seaborn을 이용하여 그리기
```
sns.kdeplot(x='a', data= D)            #선 그래프
sns.histplot(x='a', data= D)           #히스토그램
sns.distplot(titanic['a'])             #위 둘다
```

-Seaborn을 이용하여 점선도 및 선형 회귀선 표시
```
sns.regplot(x='a',   #x축 변수
            y='d',  #y축 변수
            data= D) #데이터
#fit_reg=False 선 표시안함.
```

-Seaborn을 이용하여 heatmap 그리기
```
sns.heatmap(table,          #데이터
            annot=True,     #각 셀의 값 표기 유무
            fmt='d',        #표기된 데이터를 정수 타입
            cmap='Blues_r', #원하는 컬러 맵
            linewidths=0.5, #컬러맵의 구분선 사이즈
            cbar=True)      #컬러바 유무
```

- Seaborn을 이용하여 범주형 산점도
```
sns.stripplot(x='a', y='b', data= D)              #범주형
sns.swarmplot(x='a', y='b', data= D, size=5) #분산형
```

- Seaborn을 이용하여 막대 그래프 그리기
```
barplot(x= ' a', y='b ' , data= D, dodge=False)
```
└→ dodge 스택으로 쌓는 막대 그래프

모듈 Matplotlib

-조인트 그래프
sns.jointplot(x='a', y='b', data= D)
 ㄴ→ 기본(선점도+히스토그램)
sns.jointplot(x='a', y='b', data= D, kind='reg')
 ㄴ→ 선점도 + 회귀선
sns.jointplot(x='a', y='b', data= D, kind='hex')
 ㄴ→ 선점도 + 육각형 히트맵
sns.jointplot(x='a', y='b', data= D, kind='kde')
 ㄴ→ 선점도 + 커널밀집 그래프(등고선 비슷한 형태)

-seaborn / sns :
seaborn에서 제공하는 데이터셋을 로드
sns.get_dataset_names() #sns 명령어 리스트
ex) df_mpg = sns.load_dataset('mpg')