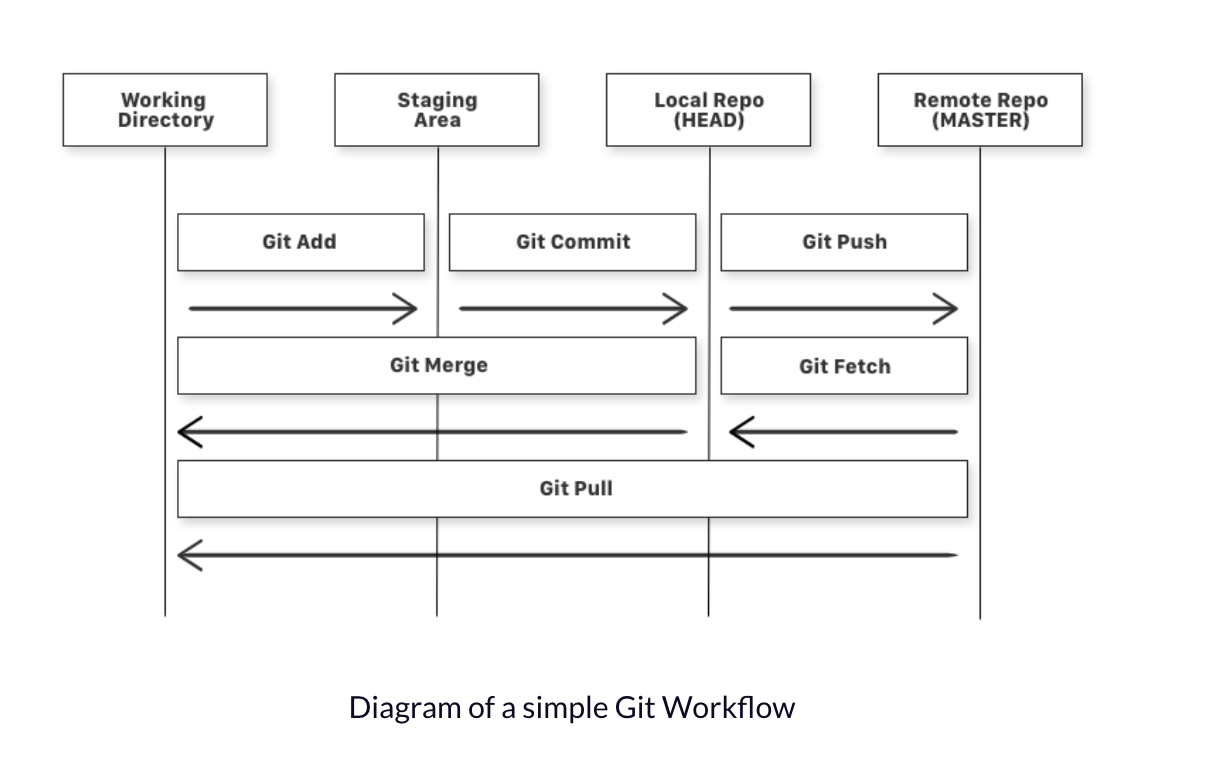
**HOMEWORK WEEK 4**

(handout for students)

**TASK 1 (Git and GitHub)**

**Question 1**

Complete definitions for key Git & GitHub terminology

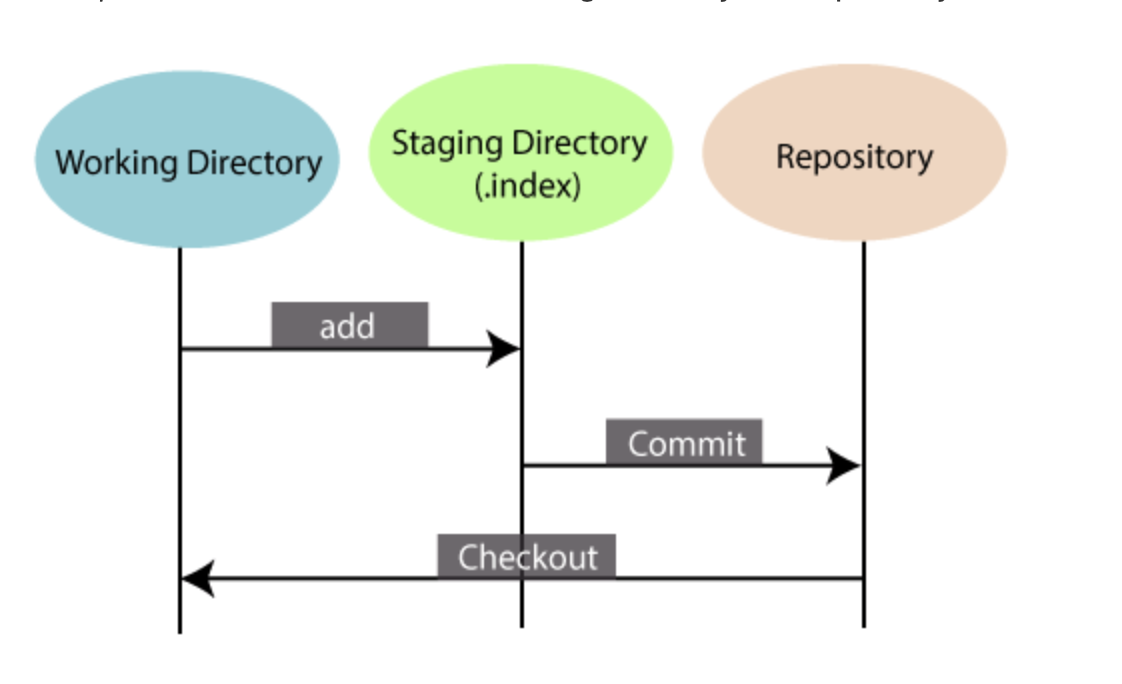


GIT WORKFLOW FUNDAMENTALS

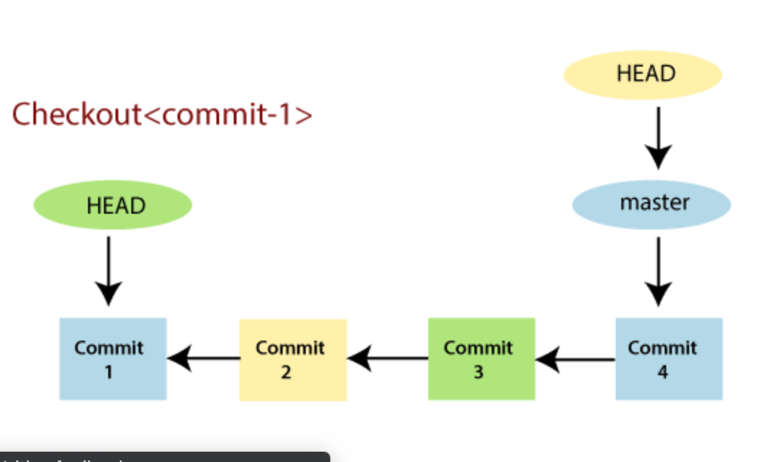
· **Working Directory –** *is the directory with the user’s source files under git control.*

*For instance, the working directory is like a workbench, it's where you work on your files (you edit them, you add new files, you delete files etc.).*

· **Staging Area -** *also known as the ‘intermediate area’, ‘git index’ or ‘cache’. This area is a staging area between the working directory and repository. It is used to build up a set of changes that you want to commit together.*

**

· **Local Repo (head) –** *Fully known as a local repository. A repository is a collection of source code. The head points out the last commit in the current checkout branch. It is like a pointer to any reference. The head can be understood as the "current branch." When you switch branches with 'checkout,' the head is transferred to the new branch.*

**

*How to make a local Git repo*

*Start a new git repository*

1. *Create a directory to contain the project.*
2. *Go into the new directory.*
3. *Type git init*
4. *Write some code.*
5. *Type git add to add the files (see the typical use page).*
6. *Type git commit*

· **Remote repo (master) –** Remote repository.

*Remote repositories are versions of your project that are hosted on the Internet or other network. The user can have several of them, each of which generally is either read-only or read/write.*

WORKING DIRECTORY STATES:

· **Staged -** *the files with the updated changes are marked to be committed to the local repository but not yet committed.*

· **Modified -** *the files with the updated changes are not yet stored in the local repository.*

· **Committed -** *the changes the user made to their file are safely stored in the local repository.*

GIT COMMANDS:

· **Git add -** *is a command used to add a file that is in the working directory to the staging area.*

· **Git commit -** *is a command used to add all files that are staged to the local repository.*

· **Git push -** *is a command used to add all committed files in the local repository to the remote repository. Hence, in the remote repository, all files and changes will be visible to anyone with access to the remote repository.*

· **Git fetch -** *is a command used to get files from the remote repository to the local repository but not into the working directory.*

· **Git merge -** *is a command used to get the files from the local repository into the working directory.*

· **Git pull -** *is command used to get files from the remote repository directly into the working directory. It is equivalent to a git fetch and a git merge.*

**TASK 2 (Exception Handling)**

**Question 1**

**Simple ATM program**

Using blocks such as try/ except / else / finally (NB: the more the better, but try to use at least two key words e.g. try/except) write a program that simulates an ATM machine to withdraw money.

**Tasks:**

1. Prompt user for a pin code

2. If the pin code is correct then proceed to the next step, otherwise ask a user to type in a password again. You can give a user a maximum of 3 attempts and then exit a program.

3. Set account balance to 100.

4. Now we need to simulate cash withdrawal

5. Accept the withdrawal amount

6. Subtract the amount from the account balance and display the remaining balance (NOTE! The balance cannot be negative!)

7. However, when a user asks to ‘withdraw’ more money than they have on their account, then you need to raise an error an exit the program.

opening\_balance = 100

current\_pin = 1234

attempts = 3

def display\_balance():

print('Your account balance is £{}'.format(opening\_balance))

def cash\_withdraw(withdrawal\_amount):

global opening\_balance

print('Your account balance is £{}'.format(opening\_balance))

if withdrawal\_amount > opening\_balance:

raise ValueError('Insufficient Balance')

opening\_balance = opening\_balance - withdrawal\_amount

print("Your updated account balance is £{}".format(opening\_balance))

while attempts != 0:

Enter\_pin = int(input("Enter your 4-digit passcode to log in: "))

if Enter\_pin != current\_pin:

attempts -= 1

print("Incorrect entry.You've {} attempts remaining".format(attempts))

if attempts == 0:

print("Card blocked! Please visit nearest branch to reactivate pin")

elif Enter\_pin == current\_pin:

print("Welcome to NatWest!")

break

while True:

print("""

1. Check A/C Balance

2. Cash Withdraw

3. Exit

""")

choice = int(input("Please Enter Choice: "))

if choice == 1:

display\_balance()

elif choice == 2:

amount = int(input("Enter Withdrawal Amount: "))

try:

cash\_withdraw(amount)

except ValueError as err:

print(err)

print("Enter another amount to withdraw")

elif choice == 3:

print("goodbye")

break

**TASK 3 (Testing)**

**Question 1**

Use the Simple ATM program to write unit tests for your functions.

You are allowed to re-factor your function to ‘untangle’ some logic into smaller blocks of code to make it easier to write tests.

Try to write at least 5 unit tests in total covering various cases.

from core import cash\_withdraw, verify\_pin

opening\_balance = 100

current\_pin = 1234

attempts = 3

while attempts != 0:

Enter\_pin = int(input("Enter your 4-digit passcode to log in: "))

if verify\_pin(current\_pin, Enter\_pin) is False:

attempts -= 1

print("Incorrect entry.You've {} attempts remaining".format(attempts))

if attempts == 0:

print("Card blocked! Please visit nearest branch to reactivate pin")

elif Enter\_pin == current\_pin:

print("Welcome to NatWest!")

break

while True:

print("""

1. Check A/C Balance

2. Cash Withdraw

3. Exit

""")

choice = int(input("Please Enter Choice: "))

if choice == 1:

display\_balance(opening\_balance)

elif choice == 2:

amount = int(input("Enter Withdrawal Amount: "))

try:

print('Your account balance is £{}'.format(opening\_balance))

opening\_balance = cash\_withdraw(opening\_balance, amount)

print("Your updated account balance is £{}".format(opening\_balance))

except ValueError as err:

print(err)

print("Enter another amount to withdraw")

elif choice == 3:

print("goodbye")

break

from core import cash\_withdraw, verify\_pin

assert cash\_withdraw(1000, 100) == 900, "Cash not properly withdrawn"

assert cash\_withdraw(456, 6) == 450, "Cash not properly withdrawn"

assert verify\_pin("4567", "4567") is True, "Pin Verifcation not accurate"

assert verify\_pin("4568", "4567") is False, "Pin Verifcation not accurate"

try:

cash\_withdraw(100, 1000)

assert False, "Should Throw Exception"

except ValueError as Err:

assert True

print("Everything passed")

def display\_balance(opening\_balance):

print('Your account balance is £{}'.format(opening\_balance))

def cash\_withdraw(opening\_balance, withdrawal\_amount):

if withdrawal\_amount > opening\_balance:

raise ValueError("Insufficient Balance")

return opening\_balance - withdrawal\_amount

def verify\_pin(saved\_pin, entered\_pin):

return saved\_pin == entered\_pin