change ⇄ commit go back to ⇒ checkout push ⇅ pull

Terminal

> files related

use tab to autocomplete

change directory: cd [folder]

go back one level: cd ..

go back to home base cd

List items : ls

clean Terminal: clear
or: command + K

> move files

create directory (folder): mkdir [folder]

create file: touch main.js

move files : mv main.js [folder]
(main.js ← file name)

move file back a level: mv main.js ../

move file back a level and to another folder: mv main.js ../[folder]

> copy files

copy file to directory cp [file] [dir]

> rename file

change the name : mv [file] [new filename]

> delete files

Remove a file : rm [file]

Remove multi: rm [file] [file]

remove entire directory: rm -R [dir] & (for directory with files inside)

remove empty directory: rmdir [dir] (only for empty directories)

> Git related

i ⇒ insert

vim [file] shift + : + X ⇒ exit vim

git init (root level of the project) # it red ⇒ not tracked

git add [file name]

git commit -m "first commit"
(# message)

git checkout [#]
(# first seven letters)

git checkout -b myBranch
(⇒ create a new branch called myBranch)

git add -A ⇐ # will add all the files

sit hub ssh key

git clone [url]
(⇐ set up ssh first)

Add ⇒ commit ⇒ pull ⇒ push
(pull before push)

git pull origin master

git push origin master

git mergetool ⇒ command + S to save

// enum

```
enum NameOfEnum{
    case caseOne
    case caseTwo
    case caseThree
}
```

```
enum NameOfEnum{
    case caseOne, caseTwo, caseThree
}
```

```
let enumeration: NameOfEnum = .caseTwo
```

// associated values

```
enum Barcode{
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}
var productBarcode = Barcode.upc(8, 85909, 51226, 3)

switch productBarcode{
    case let .upc(numberSystem, manufacture, product, check):
        print("UPC: \(numberSystem), \(manufacture) .....")
    case let .qrCode(productCode)
        print("QR CODE: \(productCode)")
}

productBarcode = .qrCode("... --- --")
```

// rawValues

```
enum JediMaster: String{
    case yoda = "Yoda"
    case maceWindu = "Mace Windu"
}

print(JediMaster.yoda.rawValue
```

//

```
enum SwitchStatus{
    case on
    case off
}

var switchStatus: SwitchStatus = .off

func flipSwitch(status: SwitchStatus) -> SwitchStatus{
    if status == .off{
        return .on
    }else{
        return .off
    }
}
```

//Extensions

```
import UIKit
class MyClass{}

extension MyClass{

}

extension String{
    func reverse() -> String{
        var characterArray = [Character]()
        for character in self.characters{
            characterArray.insert(character, at:0)
        }
        return String(characterArray)
    }
}
var name = "nameA"
name.reverse()

extension Int {
    func square() -> Int{
        return self * self
    }
}

var value = 9
value.square()
```

//Mutating function

```
extension Double{
    mutating func calculateArea(){
        let pi = 3.1415
        self = pi * (self * self)
    }
}

class Circle{
    var radius : Double

    init(radius : Double){
        self.radius = radius
    }
}
var circle = Circle(radius: 3.3)
print(circle.radius) //3.3
circle.radius.calculateArea() 34.2109
print(circle.radius) // 34.2109
```

// UIButtonExt

// Avenir Next

```
import UIKit

extension UIButton{
    func wiggle(){
        let wiggleAnim = CABasicAnimation(keyPath "position")
        wiggleAnim.duration = 0.05
        wiggleAnim.repeatCount = 5
        wiggleAnim.autoreverses = true
        wiggleAnim.fromValue = CGPoint(x: self.center.x - 4.0 , y self.center.y)
        wiggleAnim.toValue = CGPoint(x: self.center.x + 4.0 , y self.center.y)
        layer.add(wiggleAnim, forKey: "position")
    }

    func dim(){
        UIView.animate( withDuration : 0.15, animations : {
            self.alpha = 0.75
        }){finished) in
            UIView.animate(with Duration : 0.15, animations:{
                self.alpha = 1.0
            })
        }
    }

    func colorize(){
        let randomNumberArray = generateRandomNumbers(quantity:3)
        let random Color = UIColor(red: randomNumberArray[0]/255, green: randomNumberArray[1]/255,
            blue : randomNumberArray[2]/255, alpha : 1.0)
        UIView.animate(withDuration:0.3){
            self.backgroundColor = randomColor
        }
    }
}
```

// Helpers.swift

```
import UIKit

func generateRandomNumbers(quantity:Int) -> [CGFloat]{
    var randomNumberArray = [CGFloat]()
    for _ in 1.. quantity{
        let randomNumber = CGFloat(arc4random_uniform(255))
        randomNumberArray.append(randomNumber)
    }
    return randomNumberArray
}
```

// Protocols and Delegates

```
import UIKit

protocol Number{
    var floatValue : Float { get }   -> (set)
}

extension Float : Number{
    var floatValue : Float{
        return self
    }
}
extension Double : Number{
    var floatValue : Float{
        return Float(self)
    }
}
var three : Double = 3
var four : Float = 4
three.floatValue + four
        -> //override the / operator -> // two parameters conforms to Number protocol
func /(valueA: Number, valueB: Number) -> Float{
    return valueA.floatValue / valueB.floatValue
}
let x:Double = 1.2345
let y: Int = 5
let q = x / y
```

//

```
class Question{
    var type: QuestionType
    var query: String
    var answer : String

    init(type : QuestionType, query : String, answer String){
        self.type = type
        self.query = query
        self.answer = answer
    }
}

enum QuestionType : String{
    case trueFalse = "...."
    case multipleChoice = "...."
    case shortAnswer = "...."
    case essay = "...."

    static let types = [trueFalse, multipleChoice, shortAnswer, essay]
}
```

```
enum AnswerType : String{
    case trueFalse = "._."
    case multipleChoice = "..."
    case shortAnswer = "..."
    case essay = "._."
    static let types = [trueFalse, multipleChoice, shortAnswer, essay]
}
protocol QuestionGenerator{
    func generateRandomQuestion() -> Question
}                                  -> // a protocol can inherit another protocol
protocol AnswerGenerator : QuestionGenerator{
    func generateRandomAnswer() -> Answer
}

class Quiz : QuestionGenerator{
    func generateRandomQuestion() -> Question{
        let randomNumeral = Int(arc4random_uniform(4))
        let randomType = QuestionType.types[randomNumeral]
        let randomQuery = randomType.rawValue
        let randomAnswer = AnswerType.types[randomNumeral].rawValue
        let randomQuestion = Question(type: randomType, query: randomQuery, answer: randomAnswer)
        return randomQuestion
    }
}

let quiz = Quiz()
let question = quiz.generateRandomQuestion()
                                   next line "\n"
question.type
question.query
question.answer
```

1130-7