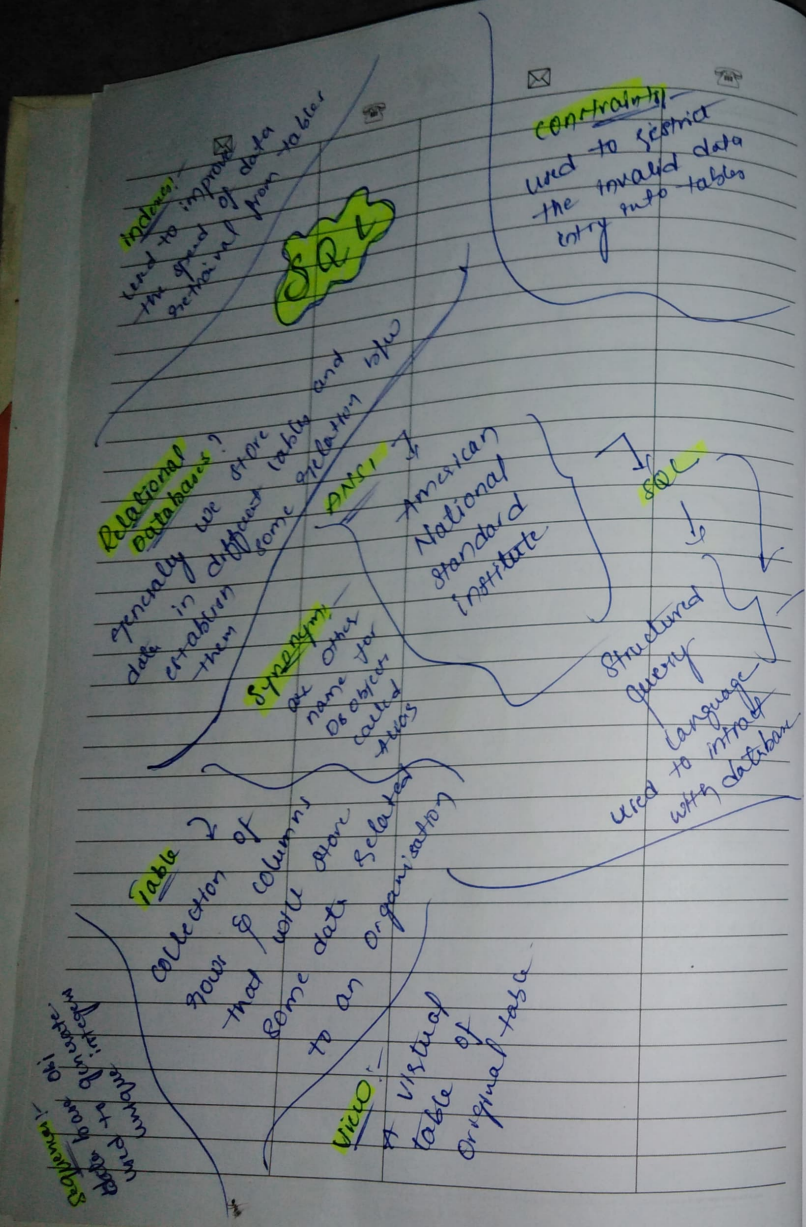
SQL course — New course

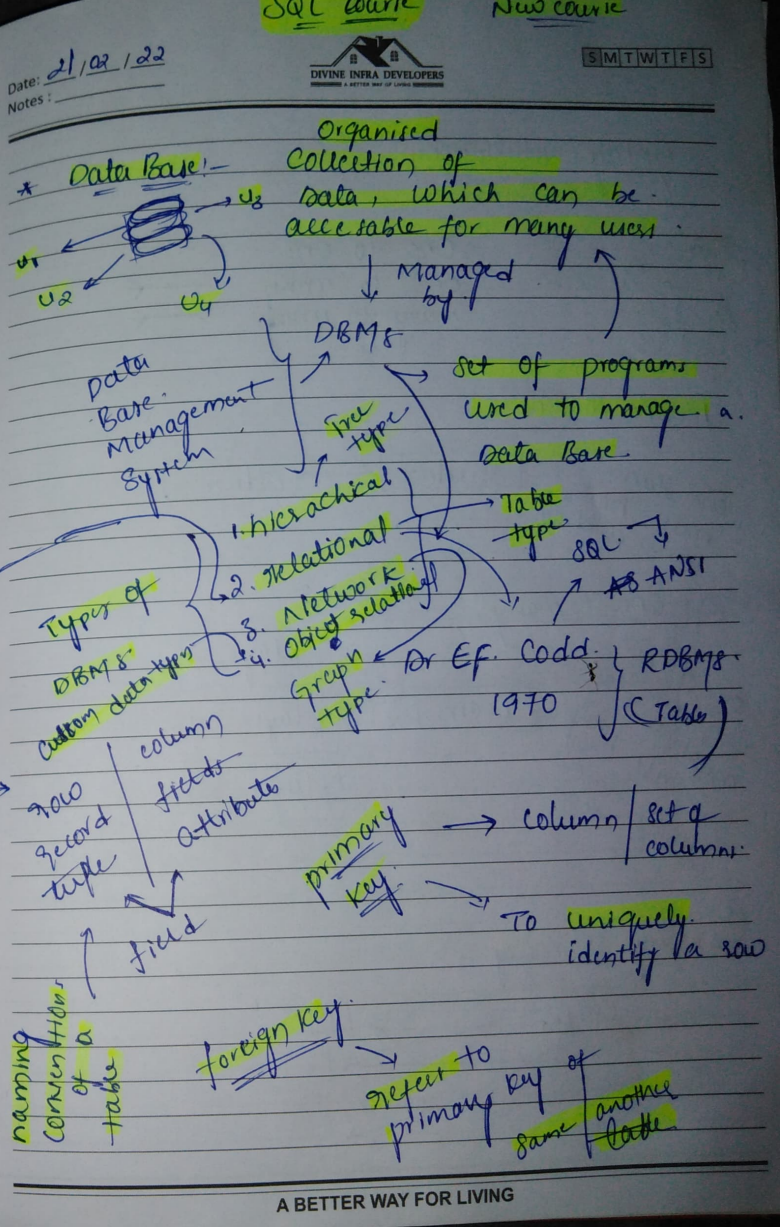
Date: 21/02/22

DIVINE INFRA DEVELOPERS — A BETTER WAY OF LIVING

S M T W T F S

Notes :





Entity relationship:-

Model design of DB.

One to one
one to many
many to many

① one to one:-

one student must have one address.

Student → Address

② One to many:-

one team can have multiple players but all players belong to one team.

Team → player 1, player 2, player 3

③ many to many:-

Emp1, Emp2, Emp3 → project 1, P2, P3

Constraints

remove inconsistent data

① allow required data only.

prevents the deletion of table if there is any dependencies.

Types of constraints :

1. Column level → used for single column.
2. Table level → used for one or more columns.

Employee name

1. Not null :- i. does not allow null values
   2. column level

Date

2. Default :- * It will insert default value if there is no specified value from u

Email

3. Unique :- It prevents duplicate values in a column. (column / table level)
   * If unique key contain 2 or more columns it is called composite unique key.
   * allow null values.

max of 32 colum

4. Primary key:- * It makes a column to have unique values

There only can be one primary key in a table

* It does not allow null values.

5. Foreign key:- 1. Column level / Table level.
2.
* It refers to primary key of same or different table.

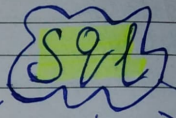
6. Check * It define a condition that each row must satisfy.

Ex!-
1. Check gender IN ('M', 'F', 'NA')

2. Check DOJ <= systemdate.

Note:
The constraint name must be unique within the schema / database

* Data dictionary:- → Collection of tables that contains info about the database.
It is created & maintained by Oracle server.



1. efficient
2. easy to learn
3. universal

DDL
Data Definition
used to define database structure
1. Create
2. Alter
3. Drop
4. Truncate

DML
Data Manipulation
used to manage data within the table
1. Insert
2. Delete
3. Update
4. Select
5. Merge

DCL
Data control
used to give / remove access on ob to users
1. Grant
2. Revoke

TCL
Transaction control
used to manage changes by DML
1. Commit
2. Rollback
3. Save

DDL

Date: __/__/__
Notes :

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

S M T W T F S

Create

1. Creating new table

```
Create Table tablename(
    column1 datatype,
    column2 datatype,
    ----
);
```

2. Create table from exitting table

```
Create Table new_tablename As
    Select column1, column2,----
    From exitting_tablename
    Where ----;
```

Note:
Making where clause to false to get table structure not the data

While creating a table like this only not null constraints are copied

Date: __/__/__
Notes :

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

S M T W T F S

Alter

1. Adding column:-

```
Alter table tablename
Add columnname datatype;
```

2. Drop column:-

```
Alter Table tablename
Drop Column columnname;
```

3. Alter / modify table:-

* Alter Table Tname rename
  ~~Alter~~ column dateofbirth to year;

To retrive a dropped table (not possible after the time)

Drop

Syntax:-

```
DROP Table tname;
```

* Can not rollback

Flash back table tablename
To before drop;

Truncate

* Truncate Table tablename;

Can not roll back

Viewing comments

Alter Table
old Table name

Rename & comment

* Rename ~~tablename~~ to New-table name;
* comment on column tablename.columnname is 'comment';

DML

① insert:- used to add a new row.

* Insert into tablename (C1, C2, C3 ...) Values (V1, V2, V3, ...);

* Insert into table name Values (V1, V2, ...);

② update:- used to modify existing row of values in a table

Syntax:
update TB name
set column = value
[where condition];

Single column
* update employee set department id = 50 where employee id = 5; → for single row.

Single column
* update employee set department = 40;
if where condition is ommited all the row will be updated.

Multiple columns
* update employee set department = 60, manager id = 1 where employee id = 6;

Delete } used to delete row from table

* delete from TB name where department id = 10; } single delete

* delete from TB name; } All row delete

Update with subquery
* update employee set (salary, bonus) = (select max(salary), max(bonus)) where job id = 'IT';

Select - clause.

→ used to retrive data from data base.

Syntax: Select $c_1, c_2, \ldots c_n$ from $t_1, t_2, \ldots t_n$;

- $c_1, c_2, \ldots c_n$ → which column to display
- $t_1, t_2, \ldots t_n$ → which table to select

① Select * From department;

Selects all columns data.

② Select id, name from department;

Selects 2 columns

③ Select job_id from employees;

Display all jobid's including Null & duplicate values.

④ Select Distinct job_id from employees;

Only unique values are displayed.

Arithematic Operators

*	1
/	1
+	2
-	2

↓ precidence.

Arithematic operator + Null value → returns → Null value.

* Select id, firstname, salary+100 from employees;

incrementation by 100 from initial value.

Aliasing:

Used to change the o/p name of a column from SQL query to stop displaying original name.

* Select firstname "First Name" from employee
* Select first_name Name from employees;
* Select firstname As username;

firstname, first_name, firstname → Original names

"First Name", Name, username → Alias names

Concatination Operator

* Concatinates columns or characters string to other columns

* Representation → ||

Ex:-

```
Select firstname||' with job type:'||jobid
"Employee details" from employees;
```

↳ result

Employee details
John with job type : it_prof
Ken with job type : sales head

Where → Used to filter no of rows while returning a query.

It requires a condition

Ex:-

```
1. select firstname, salary from employees
   where empid = 5;

2. select firstname, salary from employees
   where department = 'EEE';
```

Comparision Operators:- =, <, >, <=, >=, <> ↳ not equal

```
1. where salary >= 6000;
2. where salary < 2000;
3. where name = 'Sivaji';
```

↑ Normal comparision operators

Other comparision operators:-

1. **between**:- * for range of values.
   * both limits are inclusive.

Ex:- where salary between 2k and 5k;
range of salaries.

2. **IN**:- for a list of values

Ex:- where department IN ('EEE', 'CSE', 'IT');

3. **LIKE**:- for comparing string patterns

* where firstname like 'S%';
  ↳ all 1st names start with 'S'
* where firstname like '_a%';
  ↳ all 1st names having second letter as with 'a'

* firstname like '%S';
  ↳ all 1st names ends with S
* firstname like '%a%';
  ↳ all values that contains 'a'
* first name like 'a%o';
  ↳ starts with 'a' & ends with 'o'.

4. **Is Null**:- to test a value is null or not

* Select name, mobile from employee where address is Null;

Date: ___/___/___
Notes : _____

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

M T W T F S

Logical conditions

AND: Both true

OR: Anyone true

NOT: If condition is false.

① where Salary >= 4000 and Last name like 'D%';

② where Salary < 5000 OR name like '%S%';

③ i, where job NOT IN ('EEE', 'COE');

ii, where name NOT LIKE '%S%';

iii, where Salary not between 4000 and 5000;

5.

Date: ___/___/___
Notes : _____

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

S M

Order by

We can sort either by ascending / descending order

Null values are in bottom of the sorted list

1. Used to sort the rows from a query.
2. Order by is the last clause in SQL statement

* Select name, Salary from employee order by salary; → Default in ascending order.

(Descending order)

* Select name, Salary from employee order by salary DESC;

* Selct name, jobid, salary from employee order by jobid, Salary desc;

name,	jobid	, salary
—	IT	7K
—	IT	3K
—	Sales	11K
—	Sales	5K
—	Sales	3K

Functions → SQL functions

Single row / Multiple row / Group functions.

① Single row functions:-

* Character functions:-

Work on a single row at a time

Case manipulation

lower()
upper()
initcap()

Character manipulation.

Concat | Substr | Length | Instr | trim | replace | LPAD | RPAD

* Numerical functions:-

1. Round (column, n); 45.928,2 → 45.93
   45.928,0 → 46
2. Trunc (column, n); 45.928,2 → 45.92
   45.928,0 → 45
3. MOD(m,n); 51.2 → 1
4. Ceil(n); 2.27 → 3
5. Floor(n); 2.62 → 2.

Date: __/__/__
Notes:

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

* Working with Dates

Default format → DD-MON-RR

sysdate() → returns current database server date & time.

① Arithematic Dates:-

- Date + number
- Date - number

adding or subtracting.

finding no of days b/w two dates → Date - Date

Date + number → adding no. of days in terms of hrs

Amount of Day = Hrs / 24

1. MONTHS_BTW (d1, d2);
2. ADD_MONTHS (date, n);
3. Next_Day (date, char);
4. Last_Day (date);

* Nested fun(fun(fun(x)))

* General functions

1. NVL(exp1, exp2) → converts null value to actual value.

2. NVL2(e1, e2, e3)

if e1 is not null it returns e2
else it returns e3.

3. NULLIF(e1, e2) → null if e1 = e2
e1 if e1 ≠ e2.

4. coalesce(e1, e2, ... en) →

return the 1st not null value in the list

* Conversion functions

Implicite

Varchar2/char → Num/Date

Num → Varchar2

Date → Varchar2

Explicite

TO_char()
TO_number()
TO_date()
(char, format);

Conditional Expressions

Case expression
or
Decode expression

if, then, else logic

Syntax:-

CASE expr → can not return null value

WHEN comp1 THEN return exp1
WHEN comp2 THEN return exp2
ELSE elsexp;
END;

expr & comp exp same must be datatype

Syntax:-

DECODE (col/expression,
search1, result1,
search2, result2,
salary);

we can return null.

if search does not match with any of the search values

② Aggregate funs:

Grouped functions | Multi row functions → Operate on set of rows at a time

Example:-

1. Avg( )
2. Max( )
3. min( )
4. Sum( )
5. Count( )
6. Std dive( )
7. Variance( )

→ All group functions ignores null values

count(*); → includes null values

* Select avg(salary), Sum(salary);
* Select min(salary), max(salary);

* We can not use group fun and normal column with Group by clause.

* Listagg:- Select Listagg(firstnam, ',') within group (order by firstname) from employees;

→ to list our employee names with ; seperated

Grouped by Clause

used to divide the rows in the table into groups

Select column1, group function column2, from table [where condition] (grouped by exp) [order by column];

without Having

* Select id, Sum(salary) from employee grouped by department;

with Having

* Select id, max(salary) from employee grouped by department Having max(salary) > 45k;

All clauses

Select jobid, Sum(salary) from employee where jobid Not like '%prof%' grouped by jobid Having Sum(salary) >= 50000 Order by sum(salary);

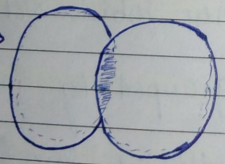
Combining data → Set Operations
→ Joins

① **Set operations:-** Set operators combine result of two or more queries into one result.

Set → union → all distinct rows from either query
→ unionall → All rows including duplicates from either query
→ intersect → All distinct rows selected by both the query's
→ Minus → All distinct rows selected by 1st statement

* They will evaluated from left to right.

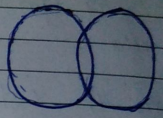
i) **Union operator:-**



* It will return the result from both the queries after eliminating the duplicates
* output is sorted default in ascending order of 1st column in select statement

* The no of columns and data types of columns being selected must be identical in all the select statements in the query, names of columns need not be identical.

ii) **Union all operator:-**

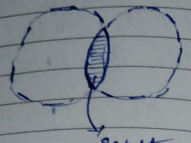


* Duplicates are not eliminated
* O/p is not sorted by default
* distinct keyword can not be used.

Ex:-

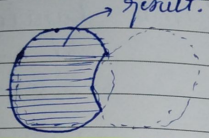
```
Select empid, jobid from employee
→ Union
  Select employeeid, jobid from jobhistory;

  Select empid, jobid from employee
  Union all
    Select empid, jobid from jobhistory;
```

iii) Intersect operator:-



* Intersect operator will return all the common rows b/w both the queries.
* Reversing the order of intersected tables does not alter the result.

iv) Minus Operator:-



* Minus operator will return the rows of 1st query which are not present in the 2nd query.
* All the columns in the where clause must be in the select statement clause for the minus operator to work

Ex:-
```
* Select jobid, empid from employee
  intersect
  select jobid, empid from jobhistory;

* Select empid from employee
  minus
  select empid from history;
```


Basics ② Joining tables

* When data from one or more than one table in database is required, a join condition is used.

Syntax:-
```
Select table1.column, table2.column
from table1, table2
where table1.column = table2.column2;
```

Table alias:- column alias will give you another name for a column.
* Whereas table alias will give you another name for table.
* It keeps SQL code smaller & saving memory.

Syntax:- From employee emp, department dept

Difference b/w Set operations and join operations:-

* Both are used to combine data from multiple tables.
* Set operations combine rows from separate ta
* Eg: Join combines columns from separat

20

Inner join:-

* To display employee department details we need to add department id from employee table & department name from department table.

Ex:-
```
Select employee.employeeid, employee.lastname,
employee.departmentid, employee.department.departmentname
from employee, department
where employee.deptid = department.deptid;
```

Table alias

```
Select E.employeeid, E.lastname, E.departmentid,
D.departmentname from employee E,
department D where E.departmentid = D.deptid;
```

Cartesian Product

Cartesian product is formed when

- → join condition is omitted.
- → join condition is invalid.

In that case all the rows in 1st table join with all rows in 2nd table

Ex:
```
Select E.lastname, D.departmentname
from E, D;
```


1

Natural join

→ Oracle 9i
→ SQL 1999 join syntax.

* It is posible to perform join automatically based on columns in 2 tables which have matching datatype and name, using a keyword called Natural join.

Syntax:
```
Select departmentid, departmentname, city
from department Natural join location;
```

2

Self join

joining a table to itself

* See pdf for better understanding.

```
Select e.employeeid, e.firstname As employeename,
m.employeeid As managerid, m.firstname as
managername from employee e, employee m
where managerid = e.employeeid;
```

Both refer to single table called employee

Date: ___/___/___
Notes: _____

DIVINE INFRA DEVELOPERS

S M T W T F S

(4) Outer join

* If a particular row does not satisfy joining condition, it will not appear in result query. To include those row in result we can use outer join.
* outer join symbol is (+): pulse sing in parantheses

Syntax: Select table1.column1, table2.column2 from table1, table2 where table1.column1 = table2.column2(+);

* (+) symbol can be placed on any side but not on both sides.

Ex:

① left — from employee E left outer join department D on E.employeeId = D.departmentid;

② Right — from employee E Right outer join department D on E.employee id = D.department id;

③ fulljoin — from employee E FULL outer join department d;

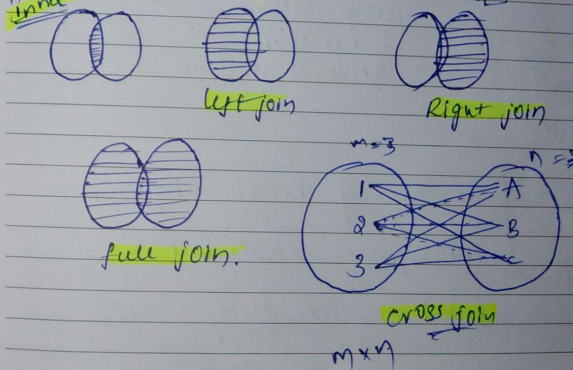
A BETTER WAY FOR LIVING

Date: ___/___/___
Notes: _____

DIVINE INFRA DEVELOPERS

S M T W

1. Inner join:- only matched rows b/w two tables.
2. Left Outer join:- returns matched pairs b/w two tables & unmatched rows of left table.
3. Right Outter join:- returns matched rows b/w two tables & unmatched rows of right table.
4. full Outer join:- returns matched rows b/w two table along with unmatched rows of both left & right table.
5. Cross join:- returns all rows from one table with another



A BETTER WAY FOR LIVING

Subqueries

- it is a select statement that is embeded in a clause of another SQL statement

Subquery's are of 2 types

- Nested Subquery
  - Single row — When inner query return a single row at a time
  - Multiple row — When inner query return multiple rows at at time
- Correlated Subquery

evaluation process:
- in case of nested subqueries
- inner query executed 1st &
- based on inner query result
- outer query evaluated.

Ex: Find all such employee whose salary is greater than robin's salary

1st query: To find robin's salary.

```
Select Salary from employee
where 1st name = 'robin';   → 40,000
```

2nd query: To find all employee having salary greater than robin's salary (40k).

```
Select firstname, employeeid, salary
where Salary > 40,000
from employee
```

↓ Subquery.

```
Select firstname, employee id, salary from
employee where salary > ( select salary
from employee where firstname = 'Robin' );
```

main query

subquery / inner query

- Subquery result is used to evaluate main query result

→ Single row Sub query.

- It returns one row from inner select statement
- It uses =, >, <, >=, <=, <> symbols.

* Display all the employees whose job id's are same as ~~emp~~ jobid of employee whose employee id is 4.

query:-

Select employeeid, lastname, jobid from employee where jobid = (select jobid from employee where employee id = 4);

query with 2 inner select queries

Select employeeid, lastname, jobid from employee where jobid = (select jobid from employee where employee id = 4) and salary > (select salary from employee where employee id = 3);

Multiple row subquery.

- Queries that return more than one row from inner select statement
- IN, ANY, ALL comparision operators are used.

Ex:- Find the employees who earn same salary as minimum salary in department.

query:-

Select lastname, salary, departmentid from employee where salary In (Select min (salary) from employee where departmentid is not null grouped by department id);

In :- Check whether a value matches any values in a subquery.

Any :- Check whether a value matches ~~any~~ at least one value in a subquery

All :- Check whether a value matches all values in a subquery

Date: __/__/__
Notes :

Correlated Subquery.

* Sql Server performs correlated subquery when the subquery references a column from the table referred in outer (parent) statement.
* in, Any, All - operators are used.
* parent statements are Select, Update or Delete.

Ex:- correlated subquery execution :-

1. Outer query executes 1st.
2. Inner query executes for each row returned by outer query.
3. Use the values resulting from the inner query to display.
4. Repeat until no candidate row remains.

Ex:- find all the employees who earn more than the average salary of their department.

query :-

```
Select lastname, salary, department id
from employee outer where salary >
(Select avg(salary) from employee
where department id = outer.department id)
```


Date: __/__/__
Notes :

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

S M T W T F S

Exists

* Exists operator is basically used with correlated subquery to test whether a value returned by the outer query exists in the result set of values returned by inner query.

* In case of Exists, if the subquery returns at least one matching row
  - → The search does not continue in inner query.
  - → The condition is flagged to true

* In case of exist, if the subquery returns row which does not match
  - → The search does not continue in inner query.
  - → The condition is flagged to false.

Date: __/__/__
Notes :

Subquery with insert, update, delete

1. Subquery with insert statement is used to copy rows from one table to another

Select insert into emp-history (employeeid, firstname, lastname, phno)
. (Select employeeid, firstname, lastname, phno from employee)

2) Subquery with update statement is used to update rows in a table based on another table

Update employee set department = (select department id from employee where employeeid = 1), jobid = (select jobid from employee where employee id = 2) where employee id = 3;

3) Subquery with delete statement is used to remove rows from a table

Ex:

Delete from employee where department id = (select department id from employee department where lower(department) like '% sales %');

Date: __/__/__
Notes :

DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

SM

TCL and DCL

TCL :- Transaction control language.

→ Commit, Rollback, Savepoint.

- Commit: End the current transaction by making all the pending changes permanent.
- Savepoint: End the current transaction by discarding all the pending changes

DCL :- Data control language → Specify the control on DB for a user

- Grant: to grant a privilege to a user
- Revoke: to revoke / remove a privilege from a user.

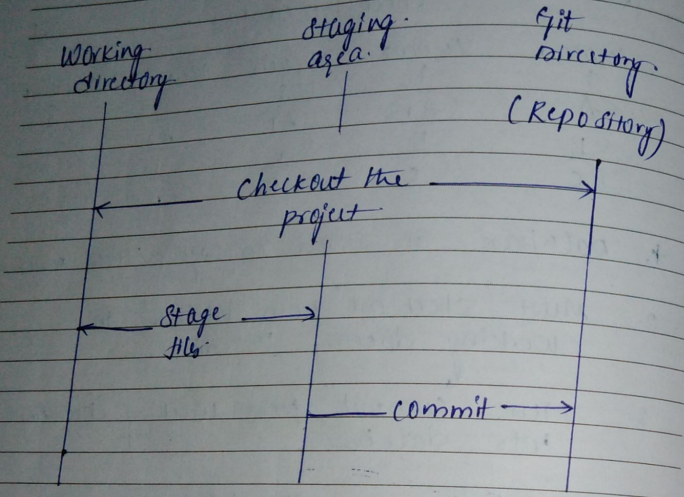
Order of Execution of SQL Statements

F → J → W → G → H → S → D → O → L

from, join, where, Group, having, Select, Distinct, Order by, Limit

FJ WHn GHS DOL

Local operations - Stage files



* Continue with udemy course.

Git & Github Bootcamp

Configuring Git

* To set username:-

```
git config --global user.name "name"
```

* To set email:-

```
git config --global user.email abc@gmail.com
```

Repositories:- A place used to store something.

↳ * different repositories will have different git histories

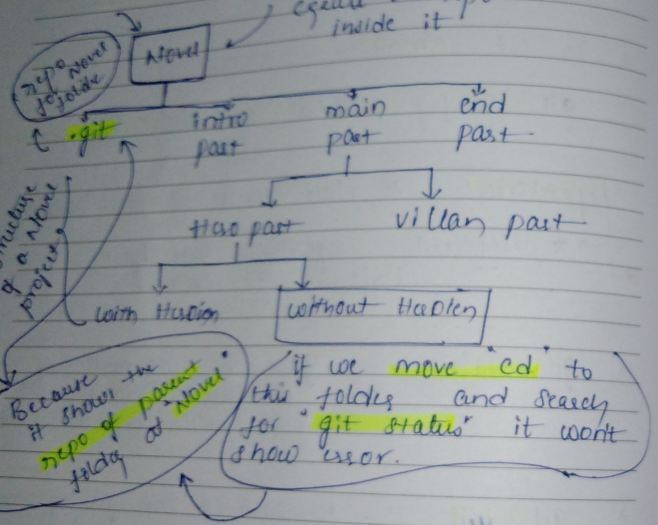
* git status :- gives the information of on the current status of a git repository & its contents.

* git init :- To create a new git repository.

Git repo created
- ls → list out the files in the current directory.
- mkdir MyFirstNovel → New folder is created
- git init → to initiate git

* rm -rf .git] :-

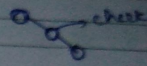
to delete current git repo (.git folder stored in the current directory.



* do not init / create a repo inside of a repo

* so before making git init make sure to check git status to know repo status.

Committing workflow:-

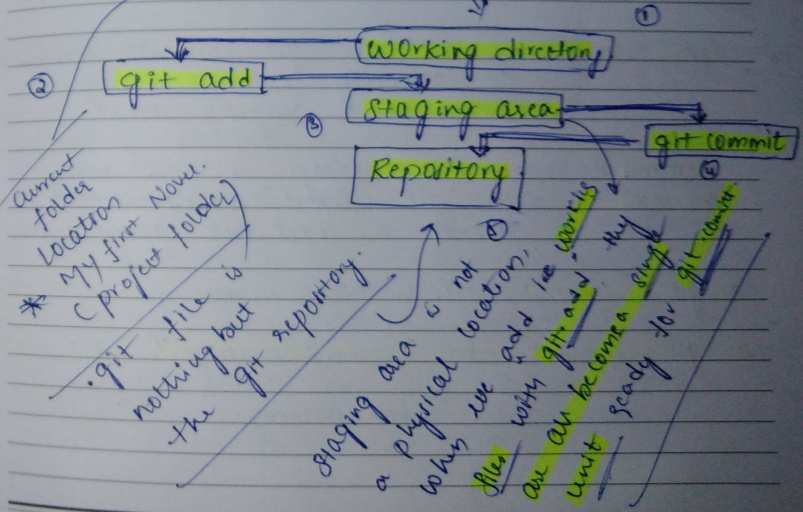


* each checkpoint are called a commit

work on stuff → Make new files, edit files, delete files etc.,

Add changes → Group specific changes together, in preparation of committing.

commit → commit everything that was previously added.



* **Adding:-** used to add working files to staging area to make them a single unit so that they are ready for git commit.

1. git add file1 → for single file add
2. git add file1 file2. → for 2 files
3. git add file1 file2 . . . filen → for n files adding

* All file names are separated by spaces.

* **Git commit:-** We use git commit command to actually commit changes from the staging area.

Commit message:- commit massage is used along with "git commit" statement used to describe a short note about commit

1. git commit → without message
2. git commit -m "Commit message". → With message.

Atomic commits:- Try to keep each commit focused on a single thing.

* i.e it's not about one file, it's about one feature or one new item. So that in feature we can undo/rollback those futures if they are not necessary.

* if you commit two features at a time and later on you are supposed to remove one feature out of those two features if you rollback the commit then both the features are rollbacked. But you are not intensioned to not to remove 2 features but it's happend. That's why we have do commit one feature or one block at a time.

Adding commit message outside git commit:-

With vim

1. git commit
2. press 'i' key. (-- insert --)
3. Type commit message.
4. press 'esc' key
5. press :wq

With Vs code!-

1. When you type git commit I,

- it will open Vs code if you are previously configured it.

* The Vs code contains an new file with some text which is committed. commented.

* Do write your commit message as much as you want and as like as you and then close the file.

* The commit message is automatically added, to check you use git log.

git log --oneline!-

To show all git messages only instead of showing a lot more stuff.

Amending commits!- If you made a commit and then you realized you forgot to include a file! or may be a typo in the commit message, it will be corrected by Amending.

Simply for redo the previous commit

Syntax: git commit --amend

Ignoring files!- we can tell git to ignore some files, directories in a given repository.

* We can use .gitignore file. to do this

* Then those files never been commited/tracked.

Uses of ignoring files!-

* To avoid seeing secret info to the world

* API keys info i.e privite info.

① **Git diff**:- We can use git diff command to view the changes between commits, branches, file and our working directory and more.

* 'git diff' list all the changes in our working directory that are not staged for next commit.

Syntax:- git diff ↵

(unstaged changes)

* A diff doesn't show entire content of a file, but instead only shows the portion or "chunk" that we are modified.

* A chunck also include some unchanged lines before and after a change to provide some context.

Terminology:-

--git a/ → old file
--git b/ → new file.

a file → - removed/modified
+ added → b file

- Hello! → present in a & removed in b
+ good morning → present in b, not in a

* git commit -am " added some message " ↵
↳ To make adding & commit in a single time.

file a:- The file which is known by staging area (Old Version)
file b:- The file which is known by directory. (New Version).

② **git diff HEAD**:- git diff head list all the changes in the working tree since your last commit.

(staged changes & unstaged)

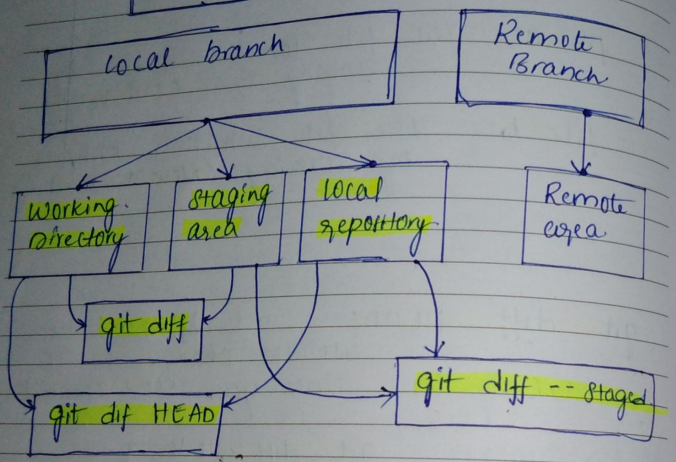
Syntax:- git diff HEAD ↵

③ git diff --staged or git diff --cached will list the changes b/w staged area and our last commit.

Syntax:- git diff --staged ↵
git diff --cached ↵

Git Basic workflow life cycle

Git diff



④ git diff for Specific files:-

To see only changes of a particular file instead of all bunch of files

Syntax: git diff HEAD path ↵

Ex: git diff HEAD Styles/main.css ↵

⑤ Comparing commits:- To compare two commits, provide git diff with the commit hashes of the commits in question.

Syntax:- git diff commit1.. commit2 ↵

Note:-

git switch master ↵

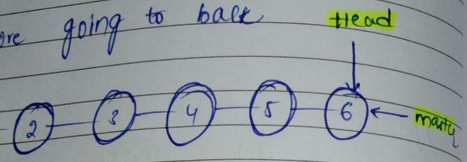
To goto original master commit in the repo.

Checkout:- We can use git checkout to previous commit.

Syntax:- git checkout commit Hashcode ↵

When we know Hashcode or branch name

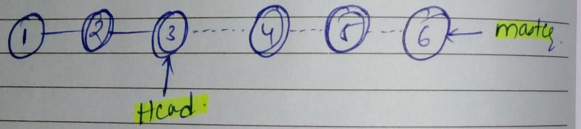
Stage 1:- Before going to back



master & head are pointing at 6 i.e latest commit.

Stage 2:- Going back to previous commit

↳ git checkout 3 ↵

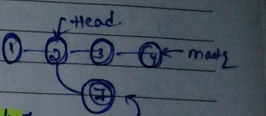


* Head points to current commit
* master remains at latest commit

Switch & checkout are same

Note:- Imagine head & master as temp & head pointers in a 'que'

To create new branch :-

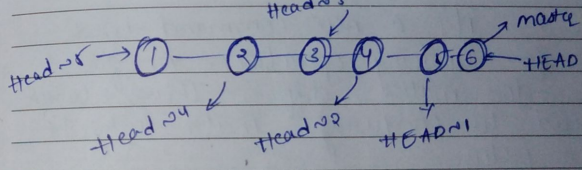


Syntax: git switch -c name of branch ↵

Ex:- git switch -c chapter1 - error ↵

* Shifting back, the head position without hashcode:-

Syntax: git checkout HEAD~1 ↵



Checkout with HEAD is used when we know upto how many commits we want to go back.

Switch is used only when we know branch name.

* git switch -: ↵

used to toggle b/w last two branches we have been working so far.

Syntax:- git switch - ↵

git switch master ↵

Undoing file upto last commit

Discarding changes with git checkout:-

Suppose you have made some changes to a file but don't want to keep them. to revert the file like whatever it looks like when you last commited, you can use

Syntax: 1. `git checkout HEAD <filename>`

2. `git checkout -- file1 file2 file3`

* Both syntax are same so we can use any one as you like

Git Restore:- it's a new command used for undoing operations. As git checkout does million different Things which may lead to confusing. That's why its comfy to picture.

Syntax:- `git restore filename`

It will restore upto last commit

undoing commit upto previous commit

* If you want to restore changes to previous commits along with last commit.

Syntax: `git restore --source HEAD~1 filename`

* it will restore your file for last commit along with 1 previous commit

unstaging files

Unstaging using git restore:-

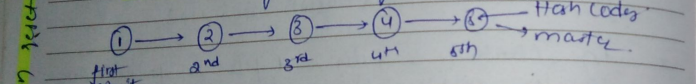
* If you add (created 3 new files and added all those at once by using git add . to staging are.
* In those 3 files, 1 file is private one.
* so to make it private we don't need to commit, we have to ignore those private file
* But these 3 are added, if you restore back then all the info & 3 files are deleted.
* So how can you push out that private file.

Syntax:- `git restore --staged filename`

Undoing commits with git reset :-

Git Reset :- Suppose you just made a couple of commits on the master branch, but you actually meant to make them on a seperate branch instead. To undo those commits you can use git reset.

* It won't delete the files, its just go back to commit i.e it will remove the commits from "log" not the files, hence our files now are in working directory.



Syntax :- git reset 4

git reset Hashcode

* If 4th & 5th commits are not req, you want it on seperate branch & remove it from master branch then follow below steps:

1. git reset 4 — files 4,5 added to working directory

Plain reset

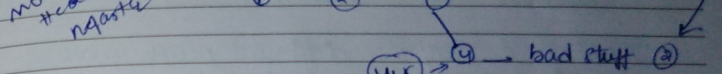
to create new branch

Creating new branch
2. git switch -c badstuff

adding files to staging area
3. git add .

committing files
4. git commit -m "add bad stuff"

moving HEAD to master
5. git switch master

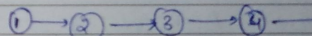


Plain reset

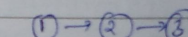
② If you want to undo both commits and actual changes file use --hard option

Syntax :- git reset --hard hashcode

* files will be deleted along with commits



git reset --hard 4



4,5 files are not added to working directory they are deleted

Hard reset

Date: __/__/__ Notes:

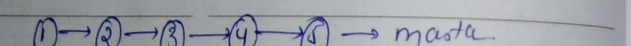
DIVINE INFRA DEVELOPERS

S M T W T F S

Reverting commits

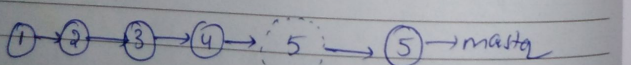
* To undo commits we have reset but why revert is introduced?
* Both do same functionality
* By using revert we can either make a new branch or completely delete the commit, so it will show impact on commits
* During a project different people working on different commits, so they may require those commits.
* so how can you undo those commits when they are sharing with others
* so we have to revert it.

Reverting



Syntax:- git revert hashcode

Ex:- git revert 5



This new commit reverse the changes from 5

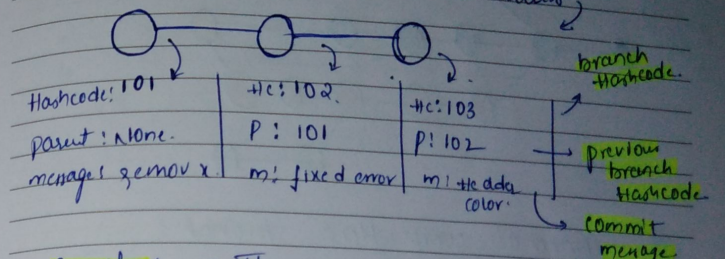
Date: __/__/__ Notes:

DIVINE INFRA DEVELOPERS

S M T W T F S

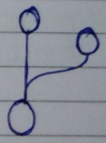
Introducing branches:-

each branch have 3 values

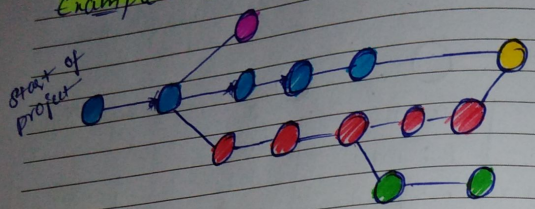


Branches:-

* They are essential part of git
* Branches are alternative timelines for a project
* They allow us to create separate contexts where we can try new things or even work on multiple ideas in parallel
* If we make changes on one branch, they do not impact on the other branches (unless we merge the changes)



Example Structure :-



- → normal work flow branch
- → New color design branch
- → bugfixing branch
- → Experimental ge design branch
- → Merge point! for

Note :- * After a bug is fixed, then we must merge the bugfixing branch to main branch so as to eliminate the bug from main branch.

* Master is the default branch name created by the git.

Viewing branches :- We git branch to view your existing branches.

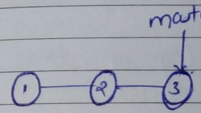
Syntax :- git branch ↵

* indicates the current branch on which we are working.

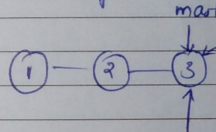
Creating branches :- We git branch <branch name> to make a new branch based on the current head position.

Syntax :- git branch branch-name ↵

Ex :-



if you do → git branch test ↵



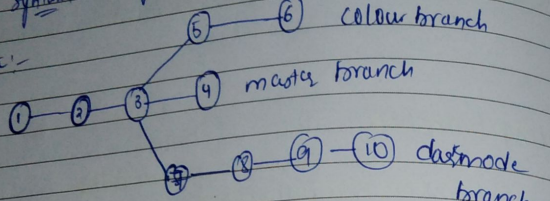
→ New branch at current position.

Note :- Head is not shifted to test branch. It's still on master branch. But it's just created a new branch at the position current.

Switching branches:- To switch between branches you have created we use it. git switch with branch name.

Syntax:- `git switch branch-name` ↵

Note:-



1. Suppose if you are on colour branch
   - if you type `git log --oneline` ↵
   it will only shows the log of that branch only.

O/p:- 1, 2, 5, 6 } commit logs

2. Similarly if you are on dark mode branch it shows

1, 2, 3, 7, 8, 9, 10 } commit logs

Creating & switching:- Instead of creating a branch, and switching to it we can use some different syntax so that we can create & switch to new branch simultaneously

Syntax:- Both are same
- `git switch -c branch-name` ↵
- `git checkout -b branchname` ↵

Notes related to switching:- two changes

1. If you modified a file in current branch and now you are trying to switch to another branch without committing the changes in current branch, then it is not possible. Because without switching without committing leads to lost data we are modified. so commit it 1st & switch.

2. If you add a new file in current directory and add some stuff in that file and now you are trying to switch to another branch without committing the file, then it is absolutely possible. As the new file does not modified, its just added & it does not shared with anyone of other branches.

Deleting and renaming Branches:-

1. Deleting:-

rule:- 1. We can not delete a branch on which we are currently staying right at that moment.
i.e if we are in master branch we can not delete the master branch.

2. We have to shift our branch from the branch we are going to delete, to another branch, then you we delete command to delete the branch.

Syntax:- 1. git branch -d branch-name ↵
(lower case)

2. git branch -D branch-name ↵
(upper case)

* -d:- used to delete the branches which are merged

* -D:- used to delete the branches which are not merged.

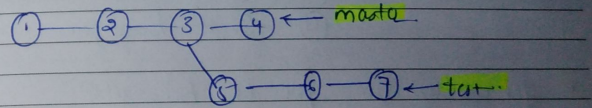
2. Renaming:-

rule:- To rename a branch we must be on the branch we are going to rename it.

i.e rename will be applied on the current branch

Syntax:- git branch -m new branch-name ↵

Note:- Branches always refers to the last or recent commit on the respective branches.



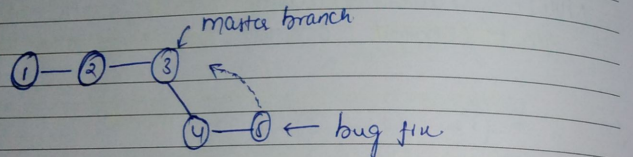
Merging Branches:-

Branches makes it super easy to work within self contained contexts, but often we want to incorporate changes from one branch to another!

This can be done by using 'git merge'.

Note :- 1. We merge branches not commits.

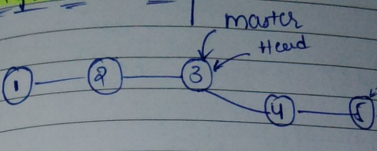
2. We always merge to current head branch.



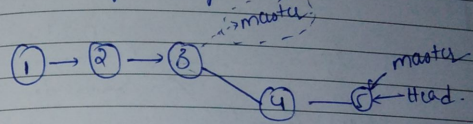
* generally we ~~fix~~ the bug 1st.
* So we want to add bugfix branch to master branch
* For that 1st we have to switch to master branch, then add bugfix branch to master branch.
* merge will add the branch specified in merge command to the current branch on which we are working.

Fast forward merge:-

1.

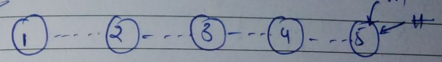


2. git merge bugfix



Master & Head are moved from 3 to 5

simple visual matter!



Since there is no other branches or commits at '3' the master branch is automatically shifted to '5' by fast forwarding its position

* This is called fast forward merge

Quit merging:- To quit current merging use quit

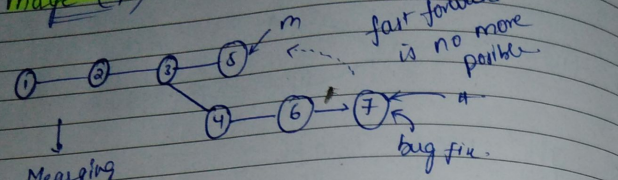
Syntax:- ~~git Quit~~ git merge --quit

Date: __/__/__
Notes: _____

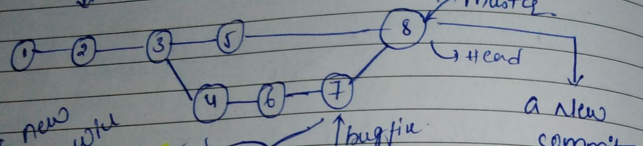
DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

S M T W T F S

Non- fast forwarding merge (Y):-



Merging



* This new commit will contain both parent hashcodes of commits i.e 5, 7'

completly New file

Date: __/__/__
Notes: _____

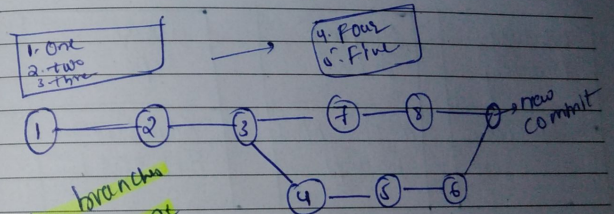
DIVINE INFRA DEVELOPERS
A BETTER WAY OF LIVING

S M T W T F S

Merging with conflicts

Rules to resolve the conflicts:-

1. Openup the files with merged conflicts.
2. edit the file(s) to remove the conflicts. Decide which branch content you want to keep in each conflict. or keep the content from both.
3. Remove the conflict "markers" in the documment.
4. Add your changes & make a commit.



So both branches deals with same file, we have to choose which branch changes appes at last

Same file is modified

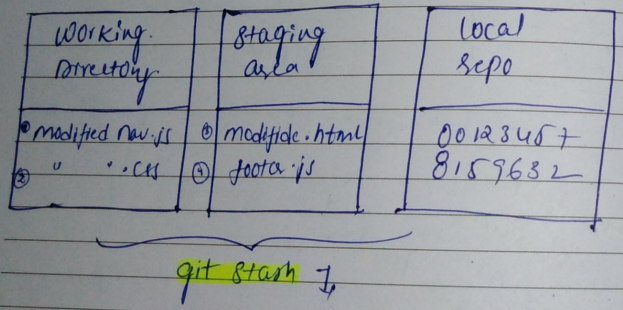
1. ONE
2. Two
3. Three

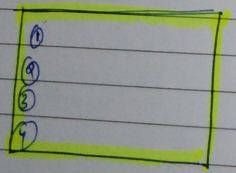
Stashing!- Git provides an easy way of stashing the uncommitted changes so that we can return to them later without having to make unnecessary commits.

Syntax!- `git stash` ↵

Removing stashing!- use "git stash pop" to remove the most recently stashed changes in your stash and re-apply them to your working copy.

Syntax!- `git stash pop` ↵





} stash area.

* **stashing area!-** a temporary area to store files.

1. we have some files in working directory & in committed staging area.
2. you want to switch the branch for an emagency purpose.
3. we have to stash those files.
4. Then the all staged & unstaged files move to stashing area.
5. so Now both wd & staging area are empty so we can switch to any branch and do some stuff.
6. Later on if you want your stashed files we have to pop them from stashing area.
7. so all the files related to working directory and staging area will go to respective areas as normal as they are in past.

Note!- we can add multiple stashes & pops since it works like a 'stack'

Stash Apply:- 'git stash apply' is works similar to 'stash pop'.

* The difference is by using pop we get all our files from stash area to working area and changes are applied.

* But when we use 'apply' instead of 'pop' the files are not moved from stashing area they will keep remains at stashing area but the changes we are making with stashed files may temporarily applied to working branch.

Note!- while working with multiple stash's

① git stash list ↵
- used to see stash list of all stashes.

② git stash apply stash@{2} ↵
- it will apply the second most recent stash in the stashing area.

Dropping stashes!- To delete a particular stash we can use drop

Syntax!- git stash drop stash@{2} ↵
- stash@{2} → second most recent stash

Clearing stash!- To clear everything from stash.

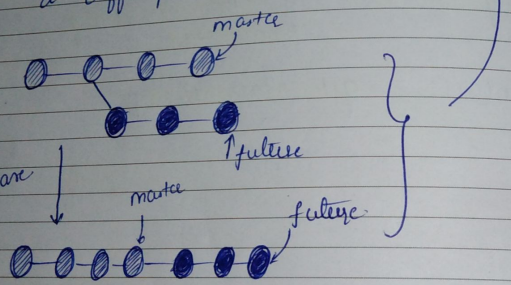
Syntax!- git stash clear ↵

Rebasing :- alternative merge → (1)
Cleanup tool → (2)

def based on

(1) * Alternative merge :-

Rebasing is a process of changing base of your branch from one commit to another commit making it appear as if you would created your branch from a different commit.



Syntax :-

1. git switch future ↓
2. git rebase master ↓

* Simply rebasing is shifting our current branch to the tip of the another branch

(2) Rewriting history :- Some times we want to rewrite, delete, rename or even reword commits (before sharing them) we can we this using "git rebase"

Syntax :- git rebase -i HEAD~1 ↓
no. of steps to go back from current head position

1. reword → To rename commits.
2. fixup → To merge the fixup commit with the commit which is just above to it.
3. drop :- To remove the commit.

* we also have a lot more to do See git description

Git Tags:- Tags are pointers that refer to particular points in git history.

* Simply a tag is just a lable for a commit
* They are used to make markings on git history for an important commit.

Tags
- light weight tags — They are just like a lable/name
- annotated tags — It stores extra meta data including author nam, mail, date and tagging message (like commit message)

All tag names are unique

Version code:-

Ex: 2.4.1

2.4.1 → major release . minor release . patch release

Syntax:- git tag ↵

we can use tags by above command

Checkout using tags:- To view the stage of a repo at a particular tag we, can use "git checkout tagname". This puts in detached head.

Syntax:- git checkout <tag> ↵

1) To create a light weight tag

Syntax:- git tag tag-name ↵

Note:- Tag will be applied at current head position in git history.

2) To create annotated tags

Syntax:- git tag -a tag-name ↵

3) To show annotated tag details

Syntax:- git show tag-name ↵

All these tag applicable only after committing

All these tag syntax 1, 2 shown above will applied to current commit.

4) Tagging previous commits:-

Syntax:- To tag a commit which is previously a loong back time and it is not a current tag.

Syntax:- git tag tagname commit hash ↵

5) Repositioning a tag:- As we know tags are unique so if you want to assign the tag to a different commit we have to reposition the tag.

Syntax:- git tag -f tagname ↵

or

git tag tagname hashcode -f ↵

6) Deleting tags:- To delete a tag

Syntax:- git tag -d tagname ↵

Git hub

What is a Github?

Git hub is a hosting platform for git repositories. You can put your own git repo's on Github and access them from anywhere and share them with people around the world.

Clonning:- Clonning means simply downloading a repo under a 'Uri'

Syntax:- git clone url ↵

Note:- while running a clone command make sure you are not inside a repo.

* git remote ↵ :- To see any remote branches are there in repo.

* git remote -v ↵ :- To know more details about remote repo.

* **Adding new remote!-** To add a new remote we require a name & url.

Syntax!- git remote add name url ↵

* **Renamming remote name!-**

Syntax!- git remote rename old name new name ↵

* **Deleting remote!-**

Syntax!- git remote remove name ↵

* **To Git push!-** To push our work to git hub

Syntax!- git push remote branch. ↵

Ex!

git push origin master ↵

name to given url while adding to git remote

push in detail:-

* Every time while you are pushing it will push a local branch up to a remote branch of same name.

* But if to push our local branch upto a remote branch of different name the syntax will quite change.

Syntax!- git push origin local branch : remote branch.

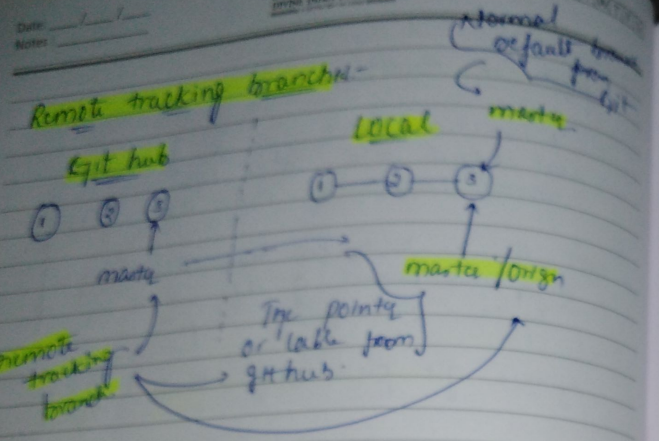
The -U option!- The -U option allows us to set the upspectrum of the branch we are pushing. You can think of this as a link connecting our local branch to the branch on Github.

Syntax!- git push -u remote branch ↵

↓

git push

Remote tracking branches:-



* We know how to access normal git branches
* To access the remote tracking branch we have to use diffrent syntax

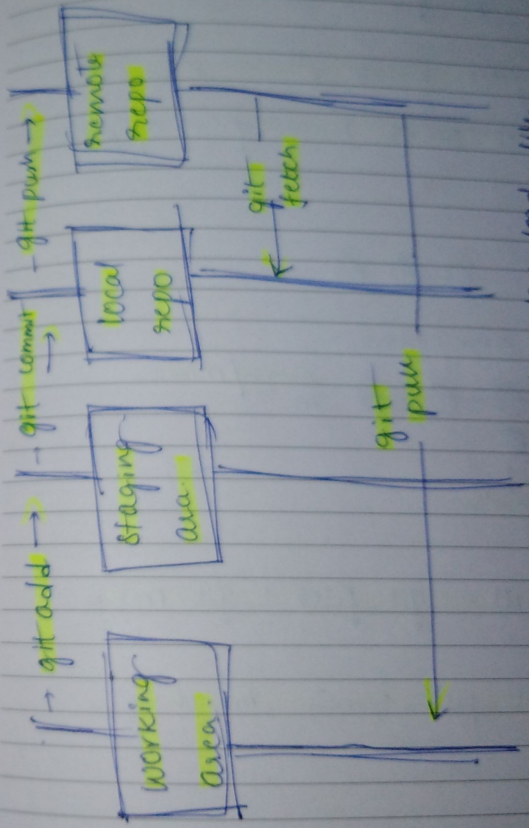
Syntax:- git switch origin/master.

Ex:- git switch remote/branch

* git branch -r

To view remote tracking branch.

fetch and pull



fetch :- it will download file from remote repo to local repo

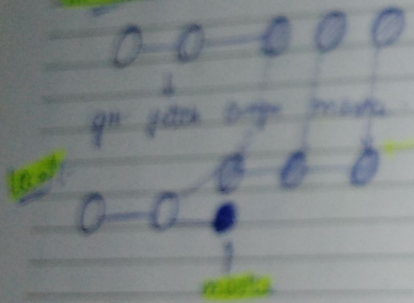
pull :- it will download file from remote repo to working area

* git fetch origin

Syntax to fetch & pull:-

* git fetch remote_name → remote_fetch
* git fetch name branch_name → only a single branch

git_fetch:



* git pull = git fetch + git merge

Syntax: git pull remote branch

Note: As pull merge the downloaded file to local repo. but to which branch it will merge?

* It will merge to current branch in local repository.

Date: / /
Notes:

Git commands:

1. `ls` :- To list out the contents of your current directory.
2. `cd foldername` : To go into the folder.
3. `cd ..` : To come out from the folder.
4. `start .` : To open current files in file explorer.
5. `pwd` : To print out the current working directory.
6. `touch filename.typeoffile` : To create files
7. `mkdir foldername` : To create a new folder
8. `rm filename.filetype` : To remove file(s)
9. `rm -rf foldername` : To delete a folder