```
unstage   > git rm --cached 1.txt
          > git status
            1.txt ← red colour
```

Git is a version control s/m for tracking changes in computer files & co-ordinating work on those files among multiple people. Used for src-code mgmt in s/w dev. and to track changes in any set of files

↳ create Repository (init ↳ Synching Repositories
↳ make changes [Track status]
↳ Parallel Development

```
> mkdir devops
> cd devops
> gedit 1.txt
> gedit 2.txt
> ls
   1.txt   2.txt
> git init
  Initialized empty Git repository /devops/.git/
```

```
> git status
    1.txt  } red
    2.txt  } untracked
             ↳ stage files to track
> git add .
       space → all files
> git status
    new file :  1.txt
    new file :  2.txt
```

// commit to repository

```
> git commit -m 'first comit adding 1.txt and
  2.txt'

> git status
    nothing to commit

> gedit 3.txt ——→ .
> git status
      3.txt ← red
    (nothing add to commit but untracked
     files present.

> ~~nano~~ .gedit 2.txt : ——→ make changes
> git .status.
      modified : 2.txt
      untracked  3.txt
```

// sync repository

git add origin.

```
/devops> git remote add origin. " https:.g. .git"
    > git push origin master
username :
password :
    done.
 > Refresh github.

> git status
>git add 2.txt
 > git status. modified : 2.txt ↙ green
                          3.txt ← red
```

↳ github
↓
new repository
↳ repo name
(public)
(creates)

3.txt <- red

```
> git commit -m "modified 2.txt"
> git push origin master
```

-> to production

```
> git add .
> git commit -m "adding 3.txt"
> git push origin master
  username :
  password :
```

2.txt with commit modify

3.txt added to github.

```
/devops > git status
```

[git clone to download remote repository to local system.

goto github

Clone or down

clone

copy

```
/devops > cd
```

~~> cd test~~

```
> mkdir test
> cd test
> git clone ____________
~/test> ls
        devops-course.
```

```
                cd devopscoure
tes/devopscourse  gedit 4.txt
         > git add .
         > git commit -m "adding clone file"
         > git push orgin master.

         > cd.
         > mkdir hello
         > cd hello
   hello > git init
         > gedit helloworld.txt
         > git add .
         > git commit -m "adding file"
         > git push orgin master

       > cd
       > cd test
  /test > git pull origin master
          not git reps

       > cd
       > ls
   devops .
 /devops > git pull orrgin master
       1 file chang, 1 insertion
```

```
>ls.
1.txt   2.txt   3.tx   4.txt
/devops > cd
>ls
devops   hello   test
> cd test.
test > ls.
devops-course.
> cd devops-course
```

test

```
>ls
1  2  3  4
>git pull origin master
>ls
1  2  3  4  5
```

folder devops

```
>gedit 5.txt
>git add.
>git commt -m
"test"
>git push origin
master
>ls
1 2 3.4 5
```

github
5.

git pull
[used to pull latest changes from repository.
git clone :- only work inside an initialized git repository]

> all ready working on clone repository & want to pull latest changes that other might have pushed to remote repository)

Parallel Development

Use branches to handle workspace of multiple developers

```
git branch <name-of-new-branch>
```

to delete `git branch -D <branch.name>`

to switch to new branch `git checkout <branch-name>`

```
> cd devops
/devops > git branch feature1
> git branch.
feature 1
* master
> git branch -D feature1
> git branch feature1
> ls
1  2  3  45
> git checkout feature 1
switched to branch feature1
> ls
1  2  3  4  5
> gedit 6.txt
> git add .
> git commit -m" added to feature
```

branch"

> ls

1 2 3 4 5 6

6 in feature branch

>git branch

* feature1

master

> git checkout master

switch to mas

> ls

1 2 3 4 5 → only 5

> git status

on branch master

> git checkout feature1

> git status

on branch feature 1

> git push origin feature 1

>git log.

Branch master
↓
Branch feature 1
→ 6.txt.

[log for every commit in repository]

git stash

: save work without commiting code.

(Helpfull when wer want to switch branches but do not want to save work to git repository

[To stash your staged files w/o commiting type "git stash"

stash untracked files git stash -u

once back and want to retreive working type git stash pop.

```
> git branch
    feature 1
  * master
> git checkout feature 1
> gedit 7.txt            ( stash file check)
> ls                -> not commited        staged
1-6, 7.txt                                  -> means
                                            commit
> git checkout master
> ls
1, 2 . 6, (7.txt) -> not tracked
                     and not part
                     of any branches as
                     it is saved
```

it is saved

```
> git checkout feature1
> git add.                [Now staged but not
                           commited]
> ls
1 . . .  7 txt
> git checkout master
> ls
> 1 . . .  7.txt   can still see 7 as its supposed
                                     to be in
> git checkout feature1               feature1
> gedit 5.txt       ( change content     only
> git add.           [stage]
> git checkout master
    M   5.txt
    A   7.txt

> ls
1 . . . 5, 7.
> git checkout   feature 1
  M.  5.txt
        A   7.txt
        > git stash
           ( saved working
                directory
        > git checkout master
            7.txt not visible anymore.
```

```
> cat 5.txt
test file
```

git back to work in feature

```
> git checkout feature1
> ls
1 - - . 6.
> cat 5.txt
test file
> git stash pop
```

[restore to working state

```
> ls
1 - - 6. 7.txt
> cat 5.txt
test file 2
> git status
new file 7.txt
modified : 5.txt
> git add .
> git commit -m "committed stashed changes"
> clear.
```

```
> ls
1 - - . 7 txt

> git status
```