Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Version Controlling- 1) Centralised

2) Distributed (GIT)

1. Developer pushes the code to local repository. (individual level)

Installing Git on Windows:

1. Open <https://git-scm.com/downloads>
2. Download git for windows –Install it

Installing Git on Linux:

1. Open terminal
2. Update apt repository

Sudo apt-get update

1. Install git

Sudo apt-get install -y git

1. git --version

Setting Username and Email:

-Setting username and email globally for all users on a specific system

1. Configuring global name and id

git config --global user.name "Abinash"

git config --global user.email [amohantyx@gmail.com](mailto:amohantyx@gmail.com)

1. List configured global user names:

git config --global –list

Git uses 3 sections when working on local machine

1. Working dir -> default folder where dev is creating the code
2. Staging Area -> intermediate buffer zone where the files are initially moved before moving into local rep
3. Local Repository -> The location where version controlling happens

Working Area -> Staging Area -> Local Repository

Untracked files staged files committed filed

git add git commit

$ pwd

/d/Dev Practice/git\_practice

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice

$ ls

hello.py.txt

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice

$ git status

fatal: Not a git repository (or any of the parent directories): .git

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice

$ pwd

/d/Dev Practice/git\_practice

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice

$ git init

Initialized empty Git repository in D:/Dev Practice/git\_practice/.git/

# git init crates a hidden git configuration folder in the particular folder and makes it a git working directory

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice (master)

$ ls -a

./ ../ .git/ hello.py.txt

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice (master)

$ git status

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

hello.py.txt

nothing added to commit but untracked files present (use "git add" to track)

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice (master)

$ git add hello.py.txt

Adds the untracked file from working area to staging area

$ git add \*

Adds all files in the current working dir to staging area

Abinash@LAPTOP-0CHV1OV5 MINGW64 /d/Dev Practice/git\_practice (master)

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: hello.py.txt

This shows the status of files in untracked and staging section

$ git reset touch\_file\_10

OR

$ git rm --cached touch\_file\_10

It removes the file from staging area to working area

To send files from staging area to Local Repository:

$ git commit -m “some message”

To see the list of commits done in Local repository

$ git log

commit 763066c4686a42a218217f9987aeaef528d79bcb (HEAD -> master)

Author: Abinash <amohantyx@gmail.com>

Date: Tue Dec 24 06:04:37 2019 +0530

first commit

commit ada16721e14220632de035f71f64b6e2be05a9e0

Author: Abinash <amohantyx@gmail.com>

Date: Tue Dec 24 06:02:56 2019 +0530

first commit

To see the list of commits done in oneline format

git log --oneline

$ git log --oneline

763066c (HEAD -> master) first commit

ada1672 first commit

Head shows the latest commit

$ cat > .gitignore

touch\_file\_11

git ignores the files mentioned in .gitignore file. It does not come in untracked or staged file

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

nothing added to commit but untracked files present (use "git add" to track)

**Git Branching**

-------------------------------

This is a feature of git which enables the developer to create code related to different functionalities on separate branches in an uncluttered way . Each indivisual functionality can be created on separate branch, it can be tested and deployed from that branch and later it can be merged with the main branch(master). The default branch of git is called master

1. To see the list of all branches:

Git branch

1. To see the list of all branches (Local and remote)

Git branch -a

1. To create a new branch

Git branch branch\_name

1. To move into a branch

Git checkout branch\_name

1. To create a branch and move into it

Git checkout -b branch\_name

1. To merge a branch with master

First move to master and then merge

Git checkout master

Git merge branch\_name

1. To delete a branch that is merged

Git branch -d branch\_name

This is also called soft delete

1. To delete a branch that is not merged

Git branch -D branch\_name

This is also called hard delete

Note: The commit history gets copied to the new branch from the master branch while creating a new branch.

Note: Irrespective of where a file is created or modified that file belongs only to the branch where it was commited

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git log --oneline

d0aedf1 (HEAD -> master) B

00d2bea A

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git branch dev

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git branch

dev

\* master

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git checkout dev

Switched to branch 'dev'

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (dev)

$ git branch

\* dev

master

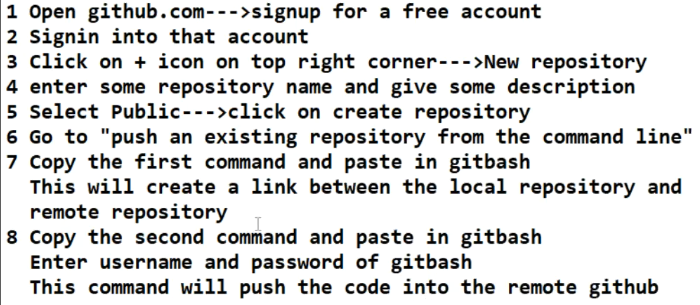
Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (dev)

$ git log --oneline

d0aedf1 (HEAD -> dev, master) B

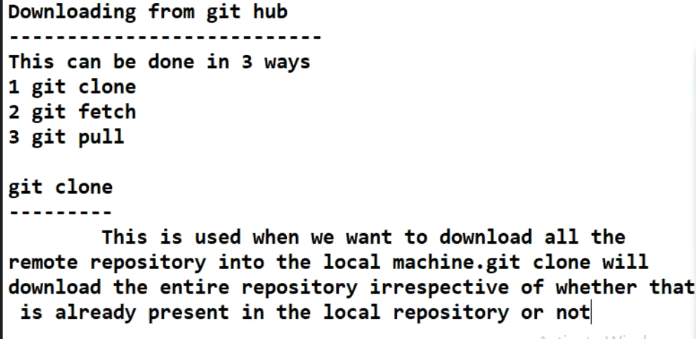
00d2bea A

**Git hub**

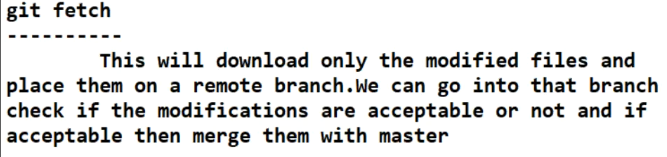


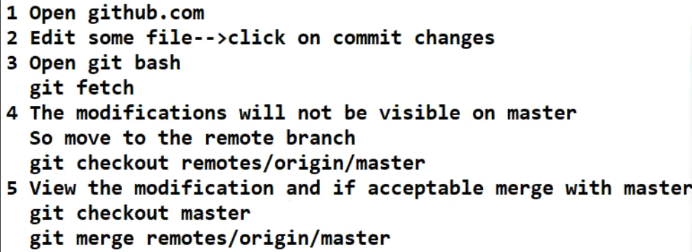
git remote add origin https://github.com/abinashm1/dev.git

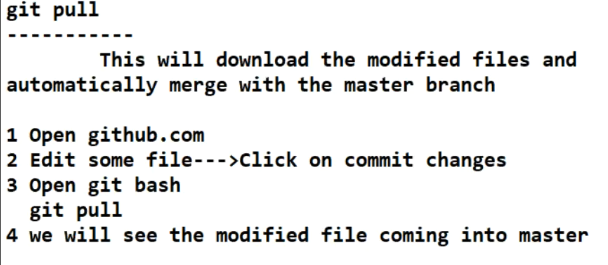
git push -u origin master



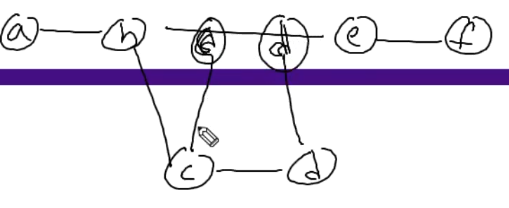
**Git clone remote\_github\_url**







**Git rebase:**



In case of merging, ‘c’ and ‘d’ commit will be merged between ‘b’ and ‘e’ commits in master

In rebasing, ‘c’ and ‘d’ commit will be the top most commit i.e ‘c’ and ‘d’ will be after ‘f’. This is also called fast forward merge

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git log --oneline

16fd6e8 (HEAD -> master) f

143d5b4 e

7d4b924 b

de34c91 a

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git checkout test

Switched to branch 'test'

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (test)

$ git log --oneline

a4acca1 (HEAD -> test) d

2983a7c c

7d4b924 b

de34c91 a

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (test)

$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: c

Applying: d

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (test)

$ git checkout master

Switched to branch 'master'

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git merge test

Updating 16fd6e8..051c0eb

Fast-forward

f3 | 0

f4 | 0

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 f3

create mode 100644 f4

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git log --oneline

051c0eb (HEAD -> master, test) d

74f40f5 c

16fd6e8 f

143d5b4 e

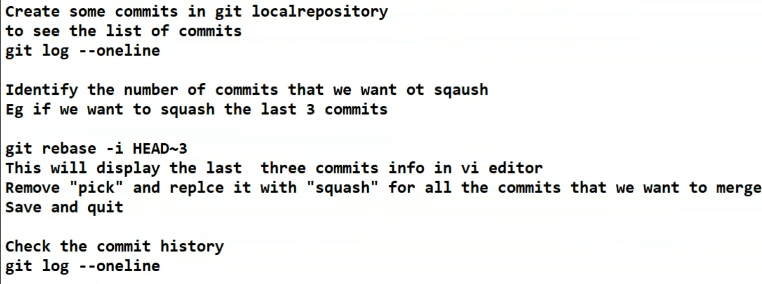
7d4b924 b

de34c91 a

**rebase for squash:**

Squashing – merging of multiple commits into one

Git rebase can also be used for performing squash.



Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git log --oneline

15bcdb2 (HEAD -> master) f4

5e018d7 f3

aa20539 f2

74f7ba7 f1

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git rebase -i HEAD~3

Successfully rebased and updated refs/heads/master.

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git log --oneline

15bcdb2 (HEAD -> master) f4

5e018d7 f3

aa20539 f2

74f7ba7 f1

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git rebase -i HEAD~3

[detached HEAD 979eaf6] f2

Date: Wed Jan 1 07:17:31 2020 +0530

3 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 f2

create mode 100644 f3

create mode 100644 f4

Successfully rebased and updated refs/heads/master.

$ git log --oneline

979eaf6 (HEAD -> master) f2

74f7ba7 f1

Abinash@LAPTOP-0CHV1OV5 MINGW64 /g/Devops\_Practice/git\_practice (master)

$ git log

commit 979eaf64c734008a4fe9b679847df65d79ce22f8 (HEAD -> master)

Author: Abinash <amohantyx@gmail.com>

Date: Wed Jan 1 07:17:31 2020 +0530

f2

f3

f4

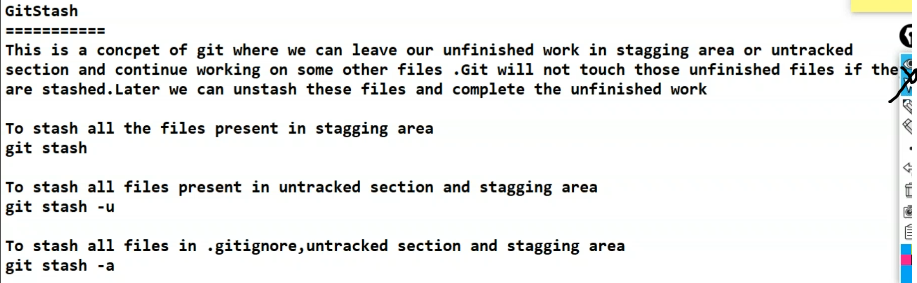
commit 74f7ba738c7ddf1d4457155f7c3bd05893f89a5b

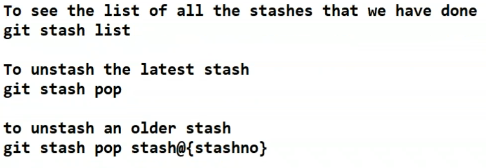
Author: Abinash <amohantyx@gmail.com>

Date: Wed Jan 1 07:17:22 2020 +0530

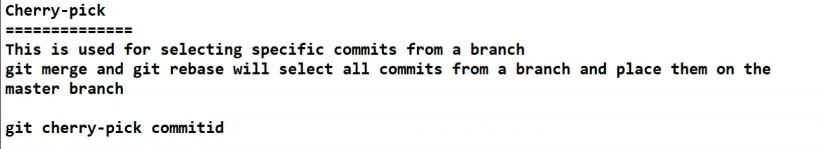
f1

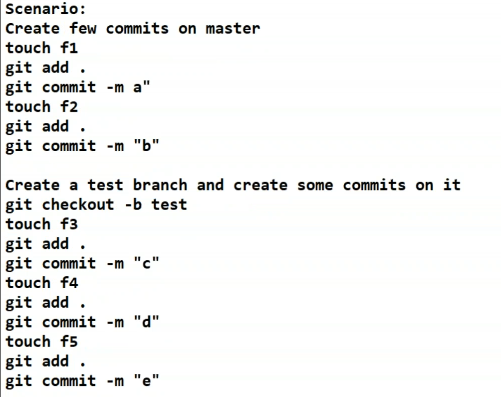
**git stash:**

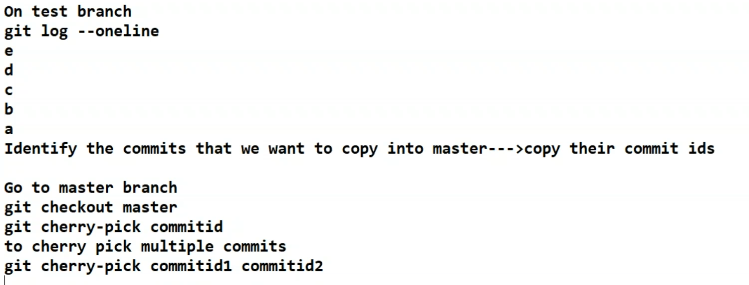




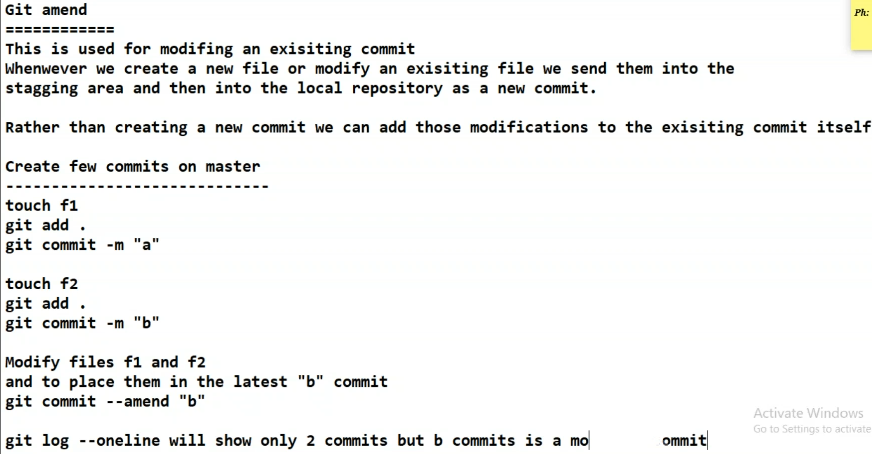
**Cherry-pick:**





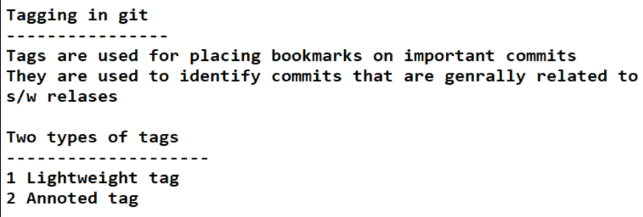


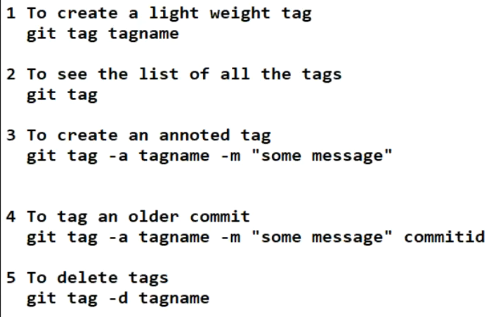
**Git Amend:**

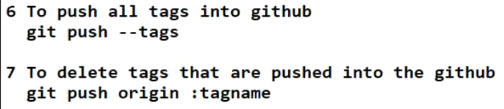


Modified\*

**Tagging in git:**







**Git Reset:**

This is used for going back to an older commit in the commit history. The head will be moved to the older commit and we will be able to see the data related to that commit.

