BEGINNER'S GUIDE TO VERSION CONTROL WITH GIT COMMANDS & GITHUB FOR COLLABORATIVE WORK

Author: Mudit Srivastava

March 2021

Contents

1. **Diff**

    a. $ **diff old_file.py new_file.py**
    b. '<' Removed from the file.
    c. '>' Added to the file
    d. 'c' Changed.
    e. 'a' Added
    f. $ **diff -u old_file.py new_file.py** (Unified format=> show diff with more context)
    g. $ **wdiff** (shows changes in words instead of working lines of the code)
    h. Graphical diff examples. **Meld, kDiff3, vimdiff**
    i. $ **diff -u old_file new_file > change.diff** ('>' Storing result from diff -u to change.diff)
    j. The results of diff could be stored in a diff file or patch file.
    k. $ **git diff** (shows modifications in all the files that **changes before staging**)
    l. $ **git diff -staged** (shows modifications in all the files that are **in Staging area**)

2. **Patch**

    a. $ **patch new_file.py < changes.diff** (If you want to reflect/apply the changes mentioned in change.diff to your original py file)

3. **Making Copies**

    a. $ **cp original.py original_unmodified.py**
    $ **cp original.py original _fixed.py**

4. **Check Git version**

    a. $ **git –version (current 2.27)**

5. **Basic Configurations** (Identify user)

    a. $ **git config --global user.email "you email address Here"**
    b. $ **git config --global user.name "Your Name Here"**
    c. $ **git config -l** (to show config)

6. **Making Directory**

    a. $ **mkdir test** (make directory)
    b. $ **cd test** (current directory)
    c. $ **git init** (git initialize)
    d. $ **ls -l** (no. of files)
    e. $ **ls -la** (show file starting a dot)

f. $ **ls -l .git/** (show more details of the directory)

7. Working tree

a. $ **cp ../file1.py**
b. $ **ls -l**

8. Adding to Staging Area & Commit

a. $ **git add file1.py**
b. $ **git status** (commits, to be committed, untracked)
c. $ **git commit** (commit everything on staging area)
d. ESC and then :wq (to exit vim)
e. $ **git commit -m 'message here'** (Or add commit message)
f. $ **git commit -m '1st msg' -m '2nd msg'** (For multiple lines)
g. $ **git add -p** (shows modified lines and 'y', 'n' to stage the changes, All tracked files)
h. $ **git add *** (Add all the files to staging area)

9. Modifying a file

a. $ **vim file1.py** (open a file in vim)

10. Log

a. $ **git log** (shows history, commit messages)
b. $ **git log -p** (shows changed lines, p from created patch)
c. $ **git log --stat** (shows no. of changed lines in all the commits)
d. $ **git log -2** (shows last 2 commits. Add any no. you like)
e. $ **git show** (view the log message and diff output the last commit if we don't know the commit ID)
f. $ **git show insert_commit_id_here** (shows changed lines for specific commit, you can also just use first 6-8 characters of commit id to work)

11. Skipping Staging Area

a. $ **git commit -a**
It is not an add then commit feature, instead it only directly commits the files those are tracked. Untracked files are not committed.

12. House keeping

a. $ **git rm file_name_here** (To delete files that are committed)
b. $ **git commit -m 'Deleted file'** (Commit this deletion)

c. $ **git mv old_file_name new_file_name** (To rename files)
d. $ **git commit -m 'Renamed old file name'** (Commit this renaming)

13. Creating a hidden file

a. $ **git > .gitignore** (Creates Gitignore type file, here dot means hidden)
b. $ **git > hello.txt** (Creates a text file)
c. $ **echo "hello" > hello.txt** (Copy text to hello.txt file)
d. $ **echo .datfiles > .gitignore** (Copy names of .datfiles to .gitignore)

14. Undoing Changes before Committing

a. $ **git checkout file_name_here** (Will restore to its latest commit, restore changes that **are not staged yet**)
b. $ **git checkout file_name_here -p** (Will restore asking each lines changed)
c. $ **git reset HEAD file_name_here** (Will restore changes **in the staging area** to it's HEAD)
d. $ **git reset -p** (Asks what changes need to be reset)

15. Amending Commits

a. $ **git commit --amend** (opens the last commit msg along with updated staging area before this and last commit. Overrides last commit)
b. $ **git revert HEAD** (Revert back the last commit and hence the changes made. It makes a new commit with a msg containing revert details and ID reverted, **Rollback**)
c. $ **git revert commit_id_here** (Revert back any commit and hence the changes made. It makes a new commit with a msg containing revert details and ID reverted)

16. Branches

a. $ **git branch** (list all the branches present, * shows current branch we're on)
b. $ **git branch new_branch_name_here** (Create new branch)
c. $ **git checkout new_branch_name_here** (To select this particular branch, * points to this branch now)
d. $ **git checkout -b new_branch_name_here** (To create and select this particular branch, Shortcut)
e. When switching back to the master branch, the actual files also change to the latest working position of the master branch. Hence the directory changes with the branch currently we are pointing to. Also, the commit history changes too.
f. $ **git branch -d branch_name** (Delete this branch. Gives error if unmerged changes with the master, you can force deletion just use **-D** instead of -d)
g. $ **git merge branch_name** (Merge this branch with master. Make sure you are currently pointing to master branch and then do this. Head points to now the Master and the merged branch_name)

17. Merge Conflicts

a. $ **git status** (Use this to locate where merge conflicts are present. Git also automatically add conflict details to the actual file for faster corrections)
b. $ **git add** (Add the rectification done and then use git status to check if all the conflicts are resolved)
c. $ **git commit** (automatic commit message as merge branch , you can add resolving merge conflict message too. Automatically merged now)
d. $ **git log --graph --oneline** (graphical way to represent the merge)
e. $ **git merge --abort** (Stop the merge if conflict is too complicated for now)

18. Remote repository (GitHub)

a. $ **git clone repo_url_here** (Making first local copy from Github repo)
b. $ **git push** (to push local repo to remote repo)
c. $ **git push -u origin branch_name** (to push local repo to remote repo if **1st time**)
d. $ **git config --global credential.helper cache** (To cache the username password for 15 minutes)
e. $ **git pull** (update local with new changes on remote repo)

19. Remote

a. $ **git remote -v** (When clone, repo default name **origin**. Fetch/ Push urls)
b. $ **git remote show origin** (to show origin remote details, where it's pointing, up to date?)
c. $ **git remote add remote_name remote_url** (Add remote)
d. $ **git branch -r** (display Remote branches, READ only)
e. $ **git status** (show additional details like origin (remote) up do date with local branch)

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote -v
origin  https://github.com/MuditSrivastava/Github-course.git (fetch)
origin  https://github.com/MuditSrivastava/Github-course.git (push)

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/MuditSrivastava/Github-course.git
  Push  URL: https://github.com/MuditSrivastava/Github-course.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git branch -r
  origin/HEAD -> origin/main
  origin/main

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   notes.docx

no changes added to commit (use "git add" and/or "git commit -a")
```

20. Fetching

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/MuditSrivastava/Github-course.git
  Push  URL: https://github.com/MuditSrivastava/Github-course.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (local out of date)
```

(When remote repo is modified without reflecting local remote branch => local out of date)

a. $ **git fetch** (to download remote repository to **current** remote **branch** (local copy) but **not** automatically **merge** to our local branch (main))

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 731 bytes | 4.00 KiB/s, done.
From https://github.com/MuditSrivastava/Github-course
   efaf102..419aa0c  main       -> origin/main

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/MuditSrivastava/Github-course.git
  Push  URL: https://github.com/MuditSrivastava/Github-course.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (local out of date)
```

b. $ **git log origin/main** (notice origin/main points to remote repo (extra) and main (local) points to latest local commit) i.e. local branch is behind origin/main by 1 commit

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git log origin/main
commit 419aa0c372979e84b9779d956114a5f281a07793 (origin/main, origin/HEAD)
Author: Mudit Srivastava <srivastava.mudit2908@gmail.com>
Date:   Tue Mar 9 23:54:47 2021 +0100

    Update README.md

    Contents added to README

commit efaf102ae04127e701634efd5d63c4f2fc94d400 (HEAD -> main)
Author: Mudit Srivastava <srivastava.mudit2908@gmail.com>
Date:   Tue Mar 9 05:24:24 2021 +0100

    starting remote repository
```

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

c. $ **git merge origin/main** (to merge remote origin/main to our local main)

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git merge origin/main
Updating efaf102..419aa0c
Fast-forward
 README.md | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
```

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git log origin/main
commit 419aa0c372979e84b9779d956114a5f281a07793 (HEAD -> main, origin/main, origin/HEAD)
Author: Mudit Srivastava <srivastava.mudit2908@gmail.com>
Date:   Tue Mar 9 23:54:47 2021 +0100

    Update README.md

    Contents added to README
```

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/MuditSrivastava/Github-course.git
  Push  URL: https://github.com/MuditSrivastava/Github-course.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```

NOTE: git **fetch** fetches remote updates but doesn't merge; git **pull** fetches remote updates and merges

21. Pulling

a. $ **git pull** (Automatic Fetch & Merge)

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/MuditSrivastava/Github-course.git
  Push  URL: https://github.com/MuditSrivastava/Github-course.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (local out of date)

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 689 bytes | 5.00 KiB/s, done.
From https://github.com/MuditSrivastava/Github-course
   419aa0c..05e5a22  main       -> origin/main
Updating 419aa0c..05e5a22
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/MuditSrivastava/Github-course.git
  Push  URL: https://github.com/MuditSrivastava/Github-course.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```

b. $ **git remote update** (fetch **all** remote **branches without automatic merge** with local branches. Use checkout and merge as needed)

22. Push conflicts

a. $ **git push**

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git push
Logon failed, use ctrl+c to cancel basic credential prompt.
git: 'credential-cache' is not a git command. See 'git --help'.
git: 'credential-cache' is not a git command. See 'git --help'.
To https://github.com/MuditSrivastava/Github-course.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/MuditSrivastava/Github-course.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

(Since remote repo is modified, first pull to update local repo)

b. $ **git pull**

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 673 bytes | 2.00 KiB/s, done.
From https://github.com/MuditSrivastava/Github-course
   01f8973..86646d1  main       -> origin/main
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

(Merge conflict here as local repo is modified at the same lines as of remote repo)

c. $ **git log –graph --oneline --all** (to visualize conflict, Three way merge required

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main|MERGING)
$ git log --graph --oneline --all
* 69b6f43 (HEAD -> main) Add date to README.md
| * 86646d1 (origin/main, origin/HEAD) Update README.md
|/
* 01f8973 Add remote, fetch & pull to notes
* 05e5a22 Add Cheat sheets to README.md
* 419aa0c Update README.md
* efaf102 starting remote repository
* ccf1544 Add Notes and Cheet sheets
* 1ab24b0 Initial commit
```

here)

d. $ git log -p origin/main

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main|MERGING)
$ git log -p origin/main
commit 86646d171d473f2429c14ce9f4a137188682c8f5 (origin/main, origin/HEAD)
Author: Mudit Srivastava <srivastava.mudit2908@gmail.com>
Date:   Wed Mar 10 21:09:26 2021 +0100

    Update README.md

diff --git a/README.md b/README.md
index 3777ef4..7a3c81f 100644
--- a/README.md
+++ b/README.md
@@ -1,5 +1,6 @@
 # Github-course
 Coursera offered by Google
+03, 2021
```

While our local repo has:

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main|MERGING)
$ git log -p main
commit 69b6f432884335ef85d7b7390e38cc4172534c27 (HEAD -> main)
Author: Mudit Srivastava <srivastava.mudit2908@gmail.com>
Date:   Wed Mar 10 21:12:13 2021 +0100

    Add date to README.md

diff --git a/README.md b/README.md
index 3777ef4..13a9484 100644
--- a/README.md
+++ b/README.md
@@ -1,5 +1,6 @@
 # Github-course
 Coursera offered by Google
+March, 2021
```

e. $ **vim README.md** (resolve conflict)

```
# Github-course
Coursera offered by Google
<<<<<<< HEAD
March, 2021
=======
03, 2021
>>>>>>> 86646d171d473f2429c14ce9f4a137188682c8f5

Contents included:
1. Notes
2. Code snippets
3. Cheat Sheets
~
```

(Choosing March, 2021, the local repo)

```
# Github-course
Coursera offered by Google
March, 2021

Contents included:
1. Notes
2. Code snippets
3. Cheat Sheets
~
~
```

f. $ **git add README.md** (Now add, commit and push)

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main|MERGING)
$ git add README.md

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main|MERGING)
$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
```

g. $ **git log --graph --oneline –all**

```
MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main|MERGING)
$ git commit
[main cb95678] Merge branch 'main' of https://github.com/MuditSrivastava/Github-course into main

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git push
Logon failed, use ctrl+c to cancel basic credential prompt.
git: 'credential-cache' is not a git command. See 'git --help'.
git: 'credential-cache' is not a git command. See 'git --help'.
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 478 bytes | 478.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To https://github.com/MuditSrivastava/Github-course.git
   86646d1..cb95678  main -> main

MUDIT SRIVASTAVA@r094168 MINGW64 /e/Germany Files/March/Git/Github-course (main)
$ git log --graph --oneline --all
*   cb95678 (HEAD -> main, origin/main, origin/HEAD) Merge branch 'main' of https://github.com/MuditSr
|\
| * 86646d1 Update README.md
* | 69b6f43 Add date to README.md
|/
* 01f8973 Add remote, fetch & pull to notes
* 05e5a22 Add Cheat sheets to README.md
```

```
Merge branch 'main' of https://github.com/MuditSrivastava/Github-course into main

# Conflicts:
#       README.md
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch and 'origin/main' have diverged,
# and have 1 and 1 different commits each, respectively.
#   (use "git pull" to merge the remote branch into yours)
#
# All conflicts fixed but you are still merging.
#
```

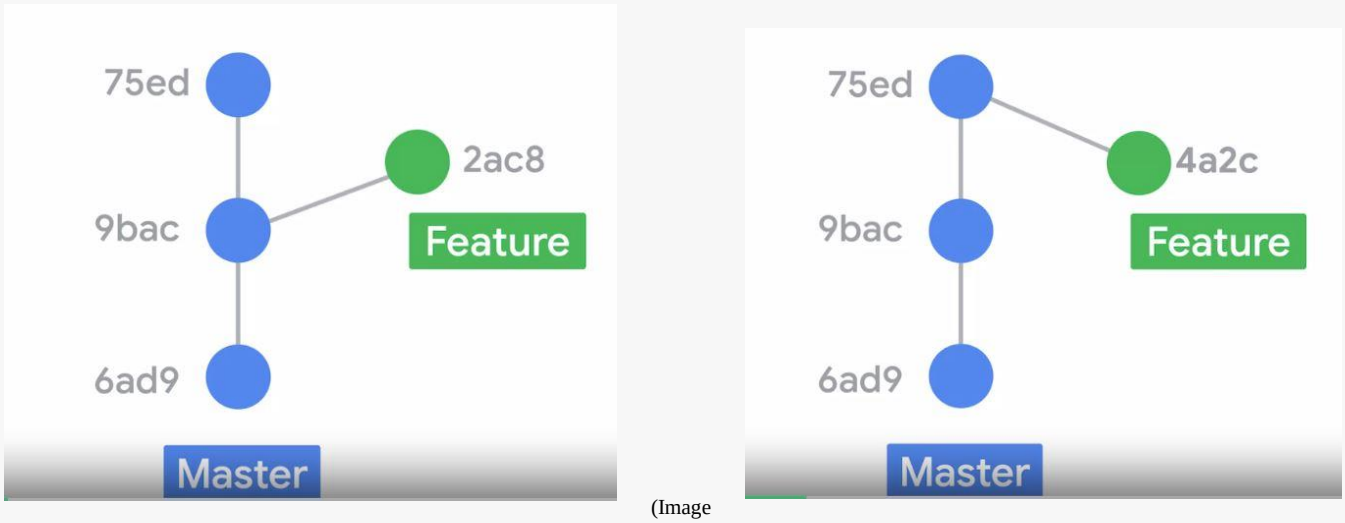
(Automatic commit message generated when committing after resolving push conflict)

23. Pushing remote branches

a. $ **git push -u origin branch_name_here** (Pushing local remote branch to remote repository)

24. **Merging remote branch**

    a. $ git checkout main
    b. $ git pull (all above steps to update main branch)
    c. $ git log --graph --oneline --all (check the common ancestor)
    d. $ **git checkout branch_name**
    e. $ **git rebase main** (changes the common ancestor to latest main commit, **still need to manually merge**)



(Image source: coursera.org/ google)

Rebasing – You can use Merge but then 3 way merge will happen. The problem with this type of merge is that it's difficult (because of the non-linear commit history) to find the moment any bug was introduced. But by using rebase, a **fast forward merge** will happen and thus it gets easier with finding that moment because of its **linear** nature.

    f. $ **git checkout master**
    g. $ **git merge branch_name** (Merge the new linear branch)
    h. $ **git push --delete origin branch_name** (delete remote branch)
    i. $ **git branch -d branch_name** (delete local branch)
    j. $ **git push** (push: since the branch is reflected on main, delete this branch both locally and remotely)

25. **Other use of Rebasing** : To rebase the changes (small) made on the main branch by two contributors at the same time to maintain a **linear** history.

    a. $ **git fetch** (Don't use git pull as it will perform a 3 way merge here. Instead use fetch so that the remote repo is copied to remote branch. But don't merge remote branch (origin/main) to our local branch (main)).
    b. $ **git checkout main**
    c. $ **git rebase origin/main** (attaching linearly the local main branch to origin/main (remote) branch. There might arise some conflicts here)

d. $ **vim conflict_file** (solve conflict in the file to continue rebase)
e. $ **add conflict_file** (after solving, stage the changes)
f. $ **git rebase --continue** (complete the rebase process)
g. $ **git push** (push the new linear structure so that origin/main also points to HEAD-> main)

26. Forking

a. A way of creating a copy of the given repository so that it belongs to our user.

27. Pull Request

a. A commit or series of commits that you send to the owner of the repository so that they incorporate it into their tree.
b. Since not everyone has a commit access to a repository, pull request is a way to suggest patches or other changes to the team with commit access.
c. **$ git push -u origin branch_name** (to push local repo to forked repo if **1st time**)
d. Before creating a pull request, it's always important to check that the code will merge successfully. GitHub tells us that our change can be automatically merged
e. Github automatically adds any further commit in the forked repo branch to the existing pull request. In order to stop auto update, make another branch in the forked repo and commit changes to it. Then create a new pull request.

28. Squashing Changes (combining all commits for pull request)

a. Create a single commit that includes both changes and a more detailed description than the one we submitted.
b. The rule to not push rebase to published commit is not valid for pull request as we will be working with forked repo.
c. **$ git rebase -i main** (interactive version of rebase)
d. An interactive rebase, a text editor opens with a list of all the selected commits from the oldest to the most recent. By changing the first word of each line, we can select what we want to do with the commits.
e. **Pick**: default action for all the commits here. Rebases the commit to the branch we selected. (Same action as $ git rebase main)
f. **Squash**: it combines both the commits so to form single commit for pull request. Also let us modify commit descriptions.
    - **pick commit1_id commit1**
    - **squash commit2_id commit2** (Use squash in the text editor of interactive rebase to combine)
    - A combine commit message will open in editor where you can modify the descriptions like combining commit messages into a single descriptive message.
    - $ **git push** (Git will give warning as merging rebase with published branch and no fast forward merge)

- **$ git push -f** (Since we don’t want to merge but replace old commits with new single commit, hence force this push. The diverging branches will be gone)

g. **Fixup**: same as squash but discard commit’s log message.

29. Code Reviews

a. Going through someone else's code, documentation or configuration and checking that it all makes sense and follows the expected patterns.
b. The goal of a code review is to improve the project by making sure that changes are high quality. It also helps us make sure that the contents are easy to understand. That the style is consistent with the overall project. And that we don't forget any important cases.
c. Code review tools let us comment on someone else's code

30. Managing Collaboration

a. If you're a project maintainer, it's important that you are reply promptly to pull requests and don't let them stagnate. The more time that passes until a pull request gets reviewed, the more likely it is that there's a new commit that causes a conflict when you try to merge in the change.
b. It's important that you understand any changes you accept.
c. When it comes to coordinating who does what and when, a common strategy for active software projects is to use an issue tracker.

31. Issue Tracker

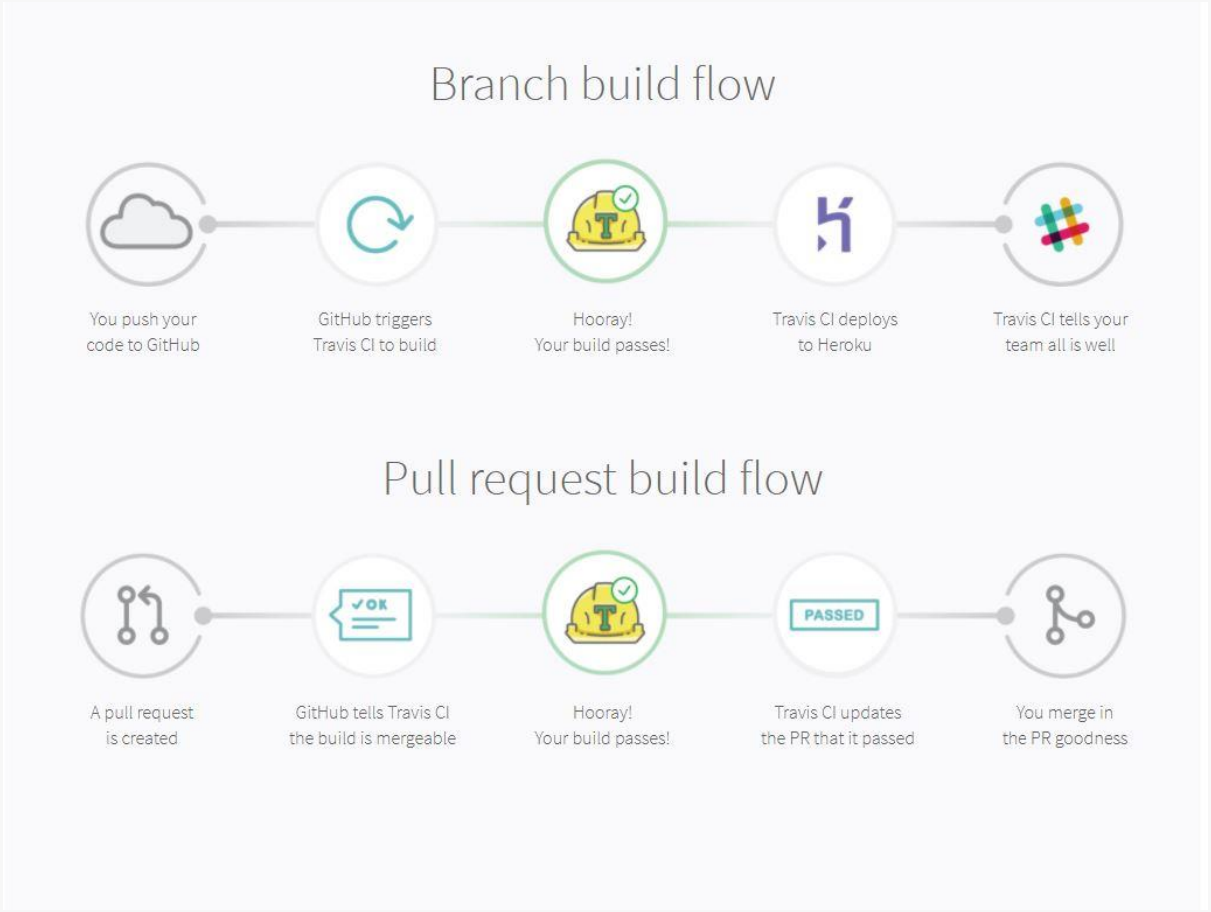
a. An issue tracker tells us the tasks that need to be done, the state they're in and who's working on them.
b. Also used to report bugs by common users.
c. You can add **Closes #issue_number_here** on your commit message while working on the issue. This message will then automatically close the issue identified by the issue no. in the commit message when pushed to the remote branch.

32. Continuous Integration (CI) system

a. Write automated tests to test the code for us and then use a continuous integration or CI system to run those tests automatically.
b. A continuous integration system will build and test our code every time there's a change. This means that it will run whenever there's a new commit in the main branch of our code. It will also run for any changes that come in through pull request.
c. Once we have our code automatically built and tested, the next automation step is continuous deployment which is sometimes called **Continuous Delivery (CD)** . Continuous deployment means the new code is deployed often.

33. Continuous Deployment

a. The goal is to avoid roll outs with a lot of changes between two versions of a project and instead do incremental updates with only a few changes at a time. This allows errors to be caught and fixed early.



(Image source: travis-ci.org)

Fig: Integration of Travis with Github Project

NOTES

1. Git-scm.com (SCM – Source Control Management, similar term for VCS)
2. Other VCSs (Ex: Subversion, Mercurial)
3. MinGW64 – Environment in Windows to let us use same command lines as in Linux.
4. IDE- Integrated Development Environments
5. GPL Version 2 License – free license to use and modify
6. .git – Configuration file open with text and .sh file open with bash
7. Git directory – Contains changes
8. Working tree – uncommitted files (current working files)
9. Git project have three areas:
    a. Git directory
    b. Staging Area
    c. Working Tree
10. A file in the directory could be:
    a. Untracked
    b. Tracked
        i. Modified
        ii. Staged
        iii. Committed
11. Commit without a message is aborted. No commit without message.
12. Good Commit Message :
    a. First line – short description (<50 characters)
    b. Second line – Empty
    c. Follows a paragraph (<72 characters) for detailed description
13. HEAD – points to current commit (could be in different branch, like bookmark)
14. $ clear – to clear screen
15. .gitignore files are used to tell the git tool to intentionally ignore some files in a given Git repository. For example, this can be useful for configuration files or metadata files that a user may not want to check into the master branch.
16. Avoid amending commits that are already public. Good for local commits though.
17. Commit ID : 40 char long string (Hash key generated by SHA1 algorithm)
18. Branch – A pointer to a particular commit (independent line of development)
19. Git Checkout – To check out the latest snapshot for both files and for branches
20. Branch Merge – two different algorithms for this: **fast forward** and **three-way merge**
21. Three-way merge activates when there some kind of divergence between two branches. From their most recent common ancestor, git combines the changes in both the branches. If the changes are in the same part of a file, **merge conflicts** arise.
22. Github, BitBucket, Gitlab are some examples of remote web based repository hosting.
23. .md – Markdown file
24. Remote work order – modify, stage, commit, fetch (automatic merge), push
25. $ git remote -v, Fetch url can be set with READ only using HTTP and Push url can be set with access control given by HTTP or SSH.
26. Remote branches (local copy) – store copy of the data stored in remote repositories.
27. If there's no Merge conflict, git pull perform **fast forward merge**.
28. git remote update will update all of your branches set to track remote ones, but not merge any changes in.
29. git fetch will update only the branch you're on, but not merge any changes in.

30. git pull will update and merge any remote changes of the current branch you're on.
31. Conflict Markers – highlight the line where conflict arises using >>>>> ======= <<<<<< symbols. Remove these symbols when resolving conflict.
32. CI/ CD Tools – Jenkins, Travis
33. Artifacts -name used to describe any files that are generated as part of the pipeline.
34. Make sure the authorized entities for the test servers are not the same entities authorized to deploy on the production servers.
35. Always have a plan to recover your access in case your pipeline gets compromised.

GOOD PRACTICES

1. Always synchronize your branches before starting any work on your own.
2. Avoid having very large changes that modify a lot of different things. Instead, try to make changes as small as possible as long as they're self-contained.
3. Push your changes often and pull before doing any work, you reduce the chances of getting conflict.
4. When working on a big change, it makes sense to have a separate feature branch. This lets you work on new changes while still enabling you to fix bugs in the other branch.
5. Regularly merge changes made on the master branch back onto the feature branch. Thus, you won't end up with a huge number of merge conflicts when the final merge time comes around.
6. If you need to maintain more than one version of a project at the same time, it's common practice to have the latest version of the project in the master branch and a stable version of the project on a separate branch.
7. Merge your changes into the separate branch whenever you declare a stable release. When using these two branches, some bug fixes for the stable version may be done directly on the stable branch if they aren't relevant to the latest version anymore.
8. Don't rebase changes that have been pushed to remote repos.
   Whenever we do a rebase, we're rewriting the history of our branch. The old commits get replaced with new commits, so they'll be based on different snapshots than the ones we had before, and they'll have completely different hash sums. This works fine for local changes but can cause a lot of trouble for changes that have been published and downloaded by other collaborators.
   Git server will automatically reject pushes that changes branch history.
   Don't push the rebase changes to the feature branch, only to the master branch that hadn't seen those changes before.
9. Having good commit messages are important.