w5_git2

Working with others in Git

`git pull` = `git fetch` + `git merge`

Practice the push workflow

no valid ssh key:

```
$ git clone git@github.com:saquantum/testgit.git
Cloning into 'testgit'...
The authenticity of host 'github.com (20.26.156.215)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

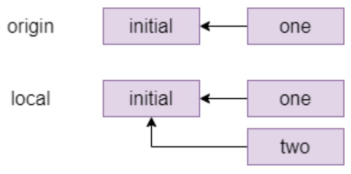
```
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 925 bytes | 132.00 KiB/s, done.
From github.com:Cailian-Liu/GitStudy
   ce4b714..49ecf0e  main       -> origin/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)

nothing to commit, working tree clean

$ git pull
Updating ce4b714..49ecf0e
Fast-forward
 GitStudy1.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

$ git add .
$ git commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 GitStudy2.txt

$ git fetch
From github.com:Cailian-Liu/GitStudy
   3b7071f..48675d0  main       -> origin/main

$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
```

```
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 279 bytes | 279.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cailian-Liu/GitStudy.git
   48675d0..b58135f  main -> main
```

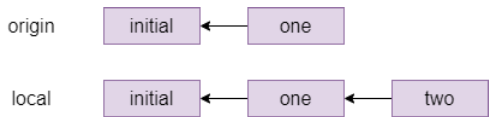
Resolve a fake conflict - rebase

One member adds or changes one file, then commits this change and pushes it by running the whole push workflow. At the same time, the second member adds or changes a different file, then commits this change. The second member starts the push workflow with `git fetch`, then `git status`. Notice you have `diverged`. (If you were to try to `git push`, with or without fetching this would produce an error.)

The commit graph of member two looks something like this:



One way to resolve this conflict is a *rebase*, which is pretending that member two had actually fetched the `one` commit before starting their own work. The command for this which member two types is `git rebase origin/main` which means *pretend that everything in origin/main happened before I started my local changes* and gives the following graph:



If member two does a `git status` after the rebase, they will see `ahead of origin/main by 1 commit` and they can now `git push` to send their local changes to the remote repository.

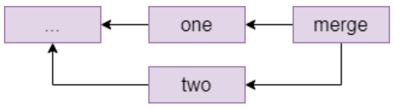
```
$ git fetch
From github.com:Cailian-Liu/GitStudy
   3b7071f..48675d0  main       -> origin/main

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

$ git rebase origin/main
Successfully rebased and updated refs/heads/main.

$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

$ git log
commit b58135facfbe440c84ac6f360f22af4bdbf2b049 (HEAD -> main)
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:23:34 2024 -0500

    GitStudy2

commit 48675d01e7ae946a602700bee199e7b7423456b1 (origin/main, origin/HEAD)
```

```
Author: soyo <KatieIove@outlook.com>
Date:   Sun Dec 1 16:22:36 2024 +0000

    abc

$ git push
To github.com:Cailian-Liu/GitStudy.git
   48675d0..b58135f  main -> main
```

Fake conflicts - merge

Like before, team member one edits one file, commits it and does the whole push workflow. The second team member at the same time, without another fetch, edits a different file and commits. The second team member starts the push workflow: fetch, status - notice you've `diverged`.

The second team member types `git pull`. Since this is a fake conflict (different files), this gets you into your editor, and you can see that on the first line is a suggested commit message starting with `Merge branch main`, which it is conventional to accept without changes - exit your editor again. Git replies `Merge made by the recursive strategy.`.



```
$ git fetch
From github.com:Cailian-Liu/GitStudy
   b58135f..7c9d977  main       -> origin/main

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

$ git pull
Merge made by the 'ort' strategy.
 GitStudy1.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git push
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cailian-Liu/GitStudy.git
   7c9d977..29de273  main -> main
```

Resolving a real conflict

Team member one creates a file called `README.md` or edits it if it already exists, and adds a line like `Created by NAME` with their own name. Then they commit this change and run the push workflow: `git fetch`, `git status`, check they're `ahead`, `git push` to the remote. Team member two, without fetching the latest commit, creates the same `README.md` file and adds a line `Created by NAME2` and commits this to their local repository. Team member two starts the push workflow: `git fetch`, `git status` and notice that you have `diverged` again.

Run `git pull` as member two. You should see the following message:

```
CONFLICT (add/add): Merge conflict in README.md
Auto-merging README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Open the file and notice that git has annotated it:

```
<<<<<<< HEAD
Created by NAME2.
=======
Created by NAME1.
>>>>>>> b955a75c7ca584ccf0c0bddccbcde46f445a7b30
```

The lines between `<<<<<<< HEAD` and `=======` are the local changes (team member two) and the ones from `======` to `>>>>>> ...` are the ones in the commit fetched from the remote, for which the commit id is shown. Member two now has to resolve the conflict by editing the file to produce the version they would like to commit. For example, you could remove all the offending lines and replace them with `Created by NAME1 and NAME2.`

Member two can now do the following:

- `git add README.md` (or whatever other files were affected).
- `git commit`. You could give a message directly, but a commit without a `m` drops you into your editor and you'll see that git is suggesting `Merge branch main ...` as a default message here. It is conventional to leave this message as it is, just exit your editor without any changes.
- Run another push workflow: `git fetch`, `git status` and notice you are now `ahead by 2 commits`: the first one was the work you did, the second is the merge commit. You're ahead, so finish the workflow with `git push`.

```
$ git fetch
From github.com:Cailian-Liu/GitStudy
   29de273..86fca56  main       -> origin/main

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

$ git pull
Auto-merging GitStudy2.txt
CONFLICT (content): Merge conflict in GitStudy2.txt
Automatic merge failed; fix conflicts and then commit the result.

$ git fetch

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   GitStudy2.txt

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

$ git log
commit 9ab7e924dd68008e9e41d70cd14218578d9e9376 (HEAD -> main)
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:38:13 2024 -0500

    change by lcl

commit 29de2735995e4a798d1f015d25c3fd605761baa2
Merge: 5b876b4 7c9d977
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:33:59 2024 -0500

    Merge branch 'main' of github.com:Cailian-Liu/GitStudy
```

```
commit 5b876b4fb5ad833d715e7a14a4e6d1be3eced6ee
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:32:49 2024 -0500

    third change

$ git add .

$ git commit
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/lcl/GitStudy/.g
it/COMMIT_EDITMSG to DOS format...
dos2unix: converting file C:/Users/lcl/GitStudy/.git/COMMIT_EDITMSG to Unix format...
[main ac64caa] Merge branch 'main' of github.com:Cailian-Liu/GitStudy

$ git push
To github.com:Cailian-Liu/GitStudy.git
   86fca56..ac64caa  main -> main
```

Working with others

Set-up

One member makes a Git repository on one of the online providers, adds the other team members and shares the cloning URL. Everyone clones the repository.

The repository must have at least one commit for the following to work. This condition is satisfied if you chose your provider's option to create a readme file; if not then make a file now, commit it and push.

The develop branch

By default, your repository has one branch named `main`. But you don't want to do your work on this branch directly. Instead, one team member creates a `develop` branch with the command

```
git checkout -b develop
```

The team member who created the develop branch should now make a commit on it.

This branch currently exists only in their local repository, and if they try and push they would get a warning about this. What they need to do is

```
git push --set-upstream origin develop
```

This adds an "upstream" entry on the local develop branch to say that it is linked to the copy of your repository called `origin`, which is the default name for the one you cloned the repository from.

You can check this with `git remote show origin`, which should display among other things:

```
Remote branches:
  develop tracked
  main  tracked
Local branches configured for 'git pull':
  develop merges with remote develop
  main  merges with remote main
Local refs configured for 'git push':
  develop pushes to develop (up to date)
  main  pushes to main  (up to date)
```

Everyone else can now `git pull` and see the branch with `git branch -a`, the `-a` (all) option means include branches that only exist on the remote. They can switch to the develop branch with `git checkout develop`, which should show:

```
Branch 'develop' set up to track remote branch 'develop' from 'origin'.
Switched to a new branch 'develop'
```

Feature branches

Since everyone is working on a different branch, you will never get a conflict this way. Anyone who is a project member can visit the github page can see all the feature branches there, but a normal `git branch` will not show other people's branches that you've never checked out yourself. Instead, you want to do `git branch -a` again that will show you all the branches, with names like `remotes/origin/NAME` for branches that so far only exist on the origin repository. You can check these out like any other branch to see their contents in your working copy.

```
$ git checkout -b lclbranch
Switched to a new branch 'lclbranch'

$ git add .
$ git commit
$ git fetch
From github.com:Cailian-Liu/GitStudy
 * [new branch]      yzhbranch  -> origin/yzhbranch

$ git status
On branch lclbranch
nothing to commit, working tree clean

$ git push
fatal: The current branch lclbranch has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin lclbranch

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

$ git push --set-upstream origin lclbranch
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'lclbranch' on GitHub by visiting:
remote:      https://github.com/Cailian-Liu/GitStudy/pull/new/lclbranch
remote:
To github.com:Cailian-Liu/GitStudy.git
 * [new branch]      lclbranch -> lclbranch
branch 'lclbranch' set up to track 'origin/lclbranch'.

$ git remote show origin
* remote origin
  Fetch URL: git@github.com:Cailian-Liu/GitStudy.git
  Push  URL: git@github.com:Cailian-Liu/GitStudy.git
  HEAD branch: main
  Remote branches:
    lclbranch tracked
    main      tracked
    yzhbranch tracked
  Local branches configured for 'git pull':
    lclbranch merges with remote lclbranch
    main      merges with remote main
  Local refs configured for 'git push':
    lclbranch pushes to lclbranch (up to date)
    main      pushes to main      (up to date)

$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/lclbranch
  remotes/origin/main
  remotes/origin/yzhbranch
```

Merging

When a feature is done, you want to merge it into develop. Everyone should try this, the procedure for this is

1. Commit all your changes and push.
2. Fetch the latest changes from origin (a simple `git fetch` does this).
3. `git checkout develop`, which switches you to the develop branch (the changes for your latest feature will disappear in the working copy, but they're still in the repository). You always merge into the currently active branch, so you need to be on `develop` to merge into it.
4. `git status` to see if someone else has updated develop since you started your feature. If so, then `git pull` (you will be *behind* rather than *diverged* because you have not changed develop yourself yet).
5. `git merge NAME` with the name of your feature branch.
6. Resolve conflicts, if necessary (see below).
7. `git push` to share your new feature with the rest of the team.

```
$ git checkout yzhbranch
branch 'yzhbranch' set up to track 'origin/yzhbranch'.
Switched to a new branch 'yzhbranch'

$ git checkout lclbranch
Switched to branch 'lclbranch'
Your branch is up to date with 'origin/lclbranch'.

$ git merge yzhbranch
Auto-merging GitStudy2.txt
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/lcl/GitStudy/.g
it/MERGE_MSG to DOS format...
dos2unix: converting file C:/Users/lcl/GitStudy/.git/MERGE_MSG to Unix format...
Merge made by the 'ort' strategy.
 GitStudy2.txt | 4 ++++
 1 file changed, 4 insertions(+)

$ git push
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cailian-Liu/GitStudy.git
   9ce72dd..a1085cc  lclbranch -> lclbranch
```

Pull requests

Pull requests are not a feature of the git software itself, but of the online providers. They let a team discuss and review a commit before merging it into a shared branch such as develop or main. Depending on the provider, branches can also be protected or assigned owners so that only the branch owner or developers with the right permissions can commit on certain branches.

The procedure for merging with a pull request on github, which you should try out:

- Commit and push your feature branch.
- On github.com in your repository, choose *Pull Requests* in the top bar, then *New Pull Request*.
- Set the *base* branch as the one you want to merge into, e.g. develop, and the *compare* branch as the one with your changes. Select *Create Pull Request*.
- Add a title and description to start a discussion, then press *Create Pull Request* again to create the request.

Anyone in the team can now go to *Pull Requests* in the top bar of the repository page and see the open requests. You can either comment on them, or if it is your role in the team to approve the request for this branch, you can approve the pull request which creates a merge.

Since a pull request is linked to a branch, you can use it for code review as follows:

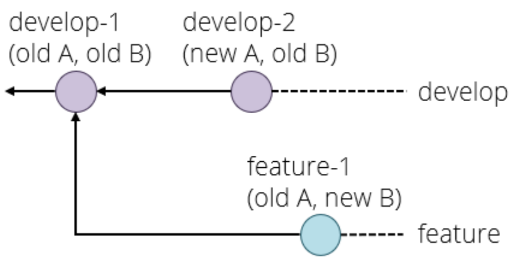
1. A developer creates a feature branch and submits a pull request.
2. The reviewer looks at the request. If they find bugs or other problems, they add a comment to the discussion.
3. The developer can address reviewer comments by making a new commit on their feature branch and pushing it, which automatically gets added to the discussion.
4. When the reviewer is happy, they approve the request which merges the latest version of the feature branch into the base branch (for example `develop`).

There is just one complication left. Suppose the following happens:

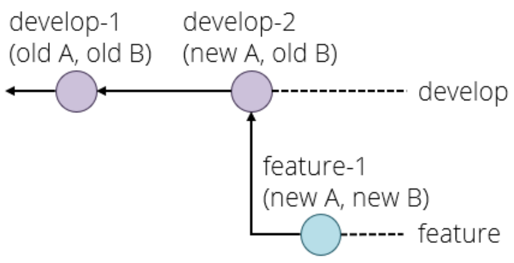
- Your project starts out with commit `develop-1` setting up the initial version of the develop branch. Imagine there are two files, A and B.

- You create a feature branch and make a commit `feature-1` which changes only file B.
- In the meantime, someone else makes a feature that changes file A, and merges it as `develop-2` to the develop branch.

You are now in the situation that `develop-2` has (new A, old B) and your `feature-1` has (old A, new B). Neither of these is what you want, you presumably want (new A, new B). We have met this situation before, but without branches. Graphically:



The solution here is to *rebase* your branch onto the latest commit on develop with `git rebase develop` and fix any conflicts that that causes, which produces the following situation:



If you now try and push your feature branch, you might get an error because the version of your feature branch on the origin repository still has the old version. The solution here is to force the push, which overwrites the old version, with

```
git push --force origin BRANCHNAME
```

This is a *think before you type* kind of command because it can break things for other developers if you do it on a shared branch. The basic safety rules are:

- Only rebase on *private* branches.
- Only force push on *private* branches, and only if it is absolutely necessary (for example to tidy up a rebase).

A private branch is one that you know no-one else is working on, for example your own feature branches.

If you ever get into a situation where you need to rebase or force push on a shared branch such as develop or main, you generally need to make sure that everyone on the team knows what is going on, synchronises their repositories both before and after the dangerous operation, and does not make any commits or pushes while someone is working on it - basically they need, in concurrency terms, an exclusive lock on the whole repository while doing this operation.

This is one reason why the main and develop branches are kept separate - and some workflows even include a third branch called `release`. If merges into main or release only ever come from the develop branch, then a situation where you need to rebase these branches can never happen.

To summarise, the pull request workflow is:

1. Commit and push your changes.
2. If necessary, rebase your feature branch on the develop branch.
3. Create a pull request.
4. If necessary, participate in a discussion or review and make extra commits to address points that other developers have raised.
5. Someone - usually not the developer who created the pull request - approves it, creating a merge commit in develop (or main).