

Creating and Managing Git Repositories in GitHub

Summer 2024 NASA Bootcamp

Git review

1. Used Git (local repo, command line)
2. Configure Git **to work with local repos**
git config (created *.gitconfig* file)
3. Created a new "repo"
git init
4. Worked with **files** using git commands:
add, commit, diff, status, log,
rm, mv, stash, reset, checkout
5. Created **branches**
branch, merge, checkout

Git

A tool that helps us keep track of changes to code in a **repository**

Branches allow you to test your code without losing your changes

GitHub

1. Interact with **GitHub** using **Git** (local and **remote repo**, CLI)

2. Configure Git **to work with remote repos**

SSH keys (created in *.ssh* directory)

3. Create a new "repo"

git clone

4. Working with **repos**

remote, fetch, pull, push

5. **Collaborating**

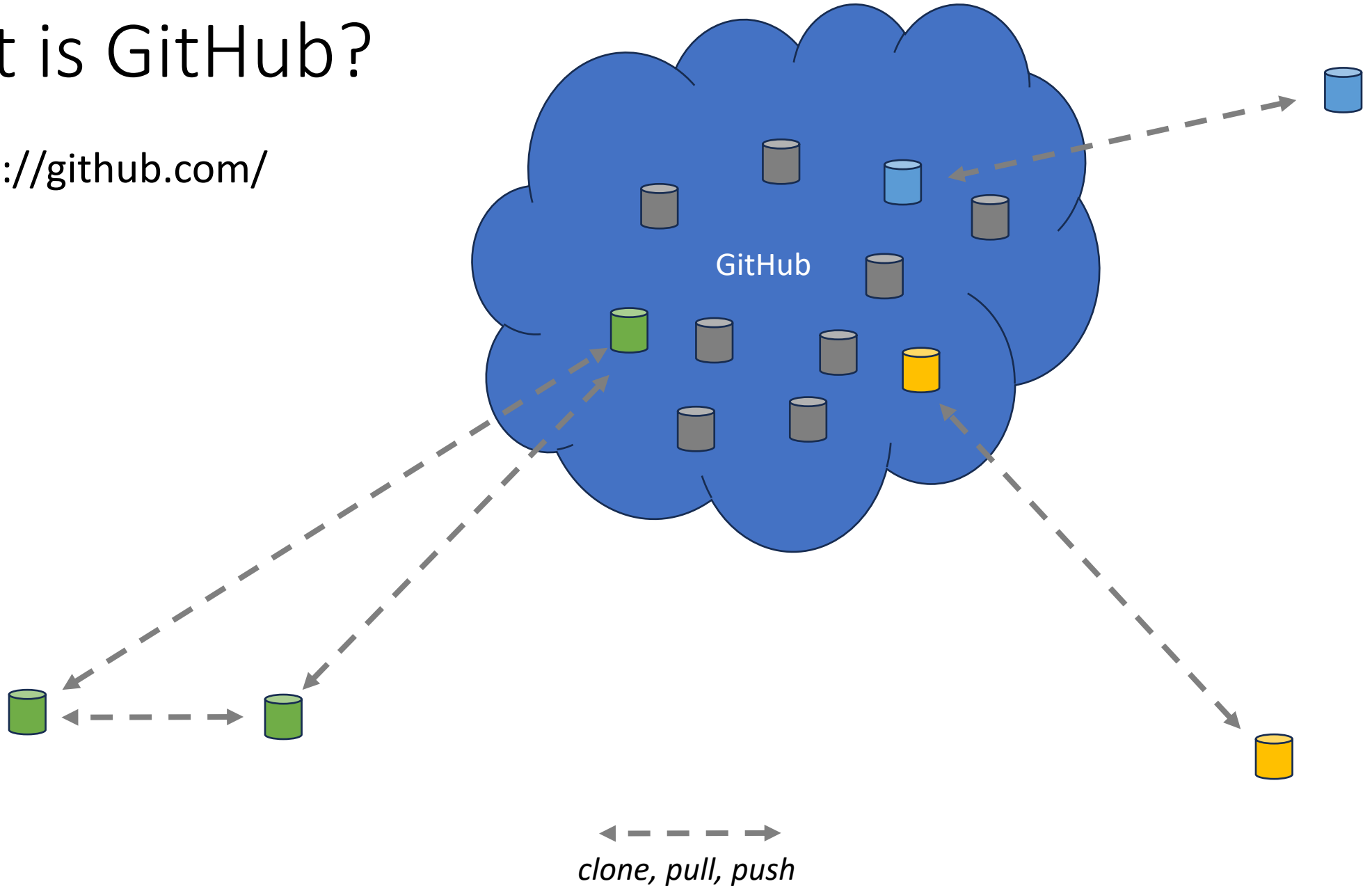
Making *pull requests*

Git allows you to synchronize code among different people

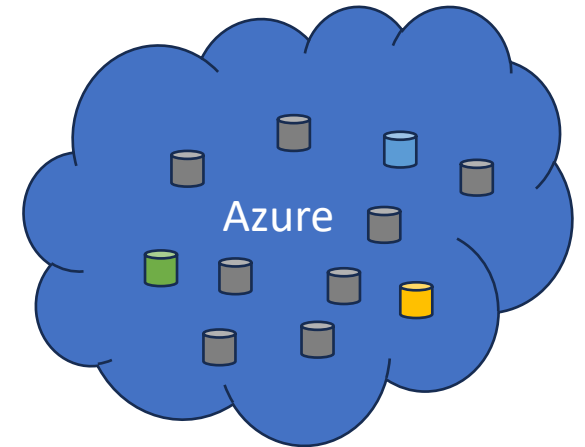
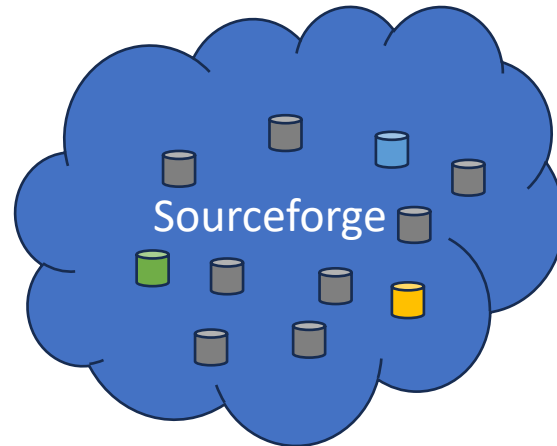
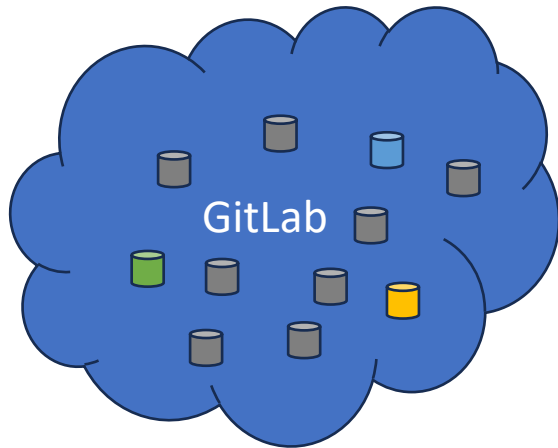
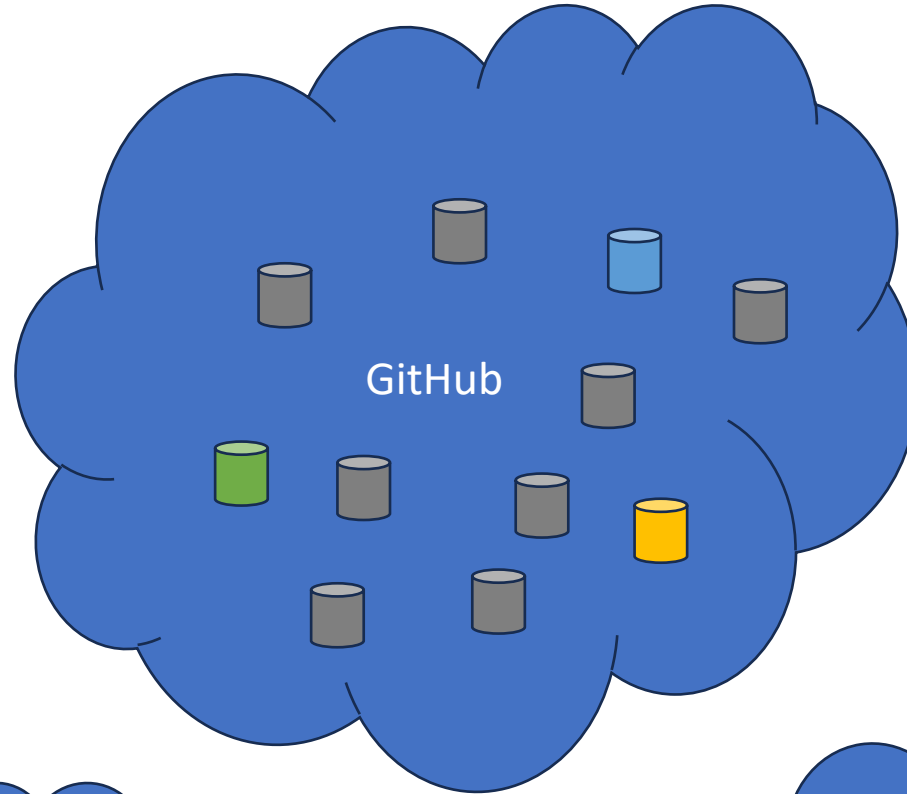
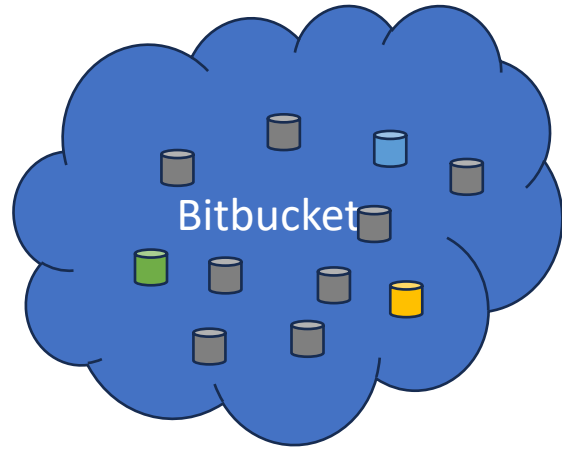
Working with GitHub

What is GitHub?

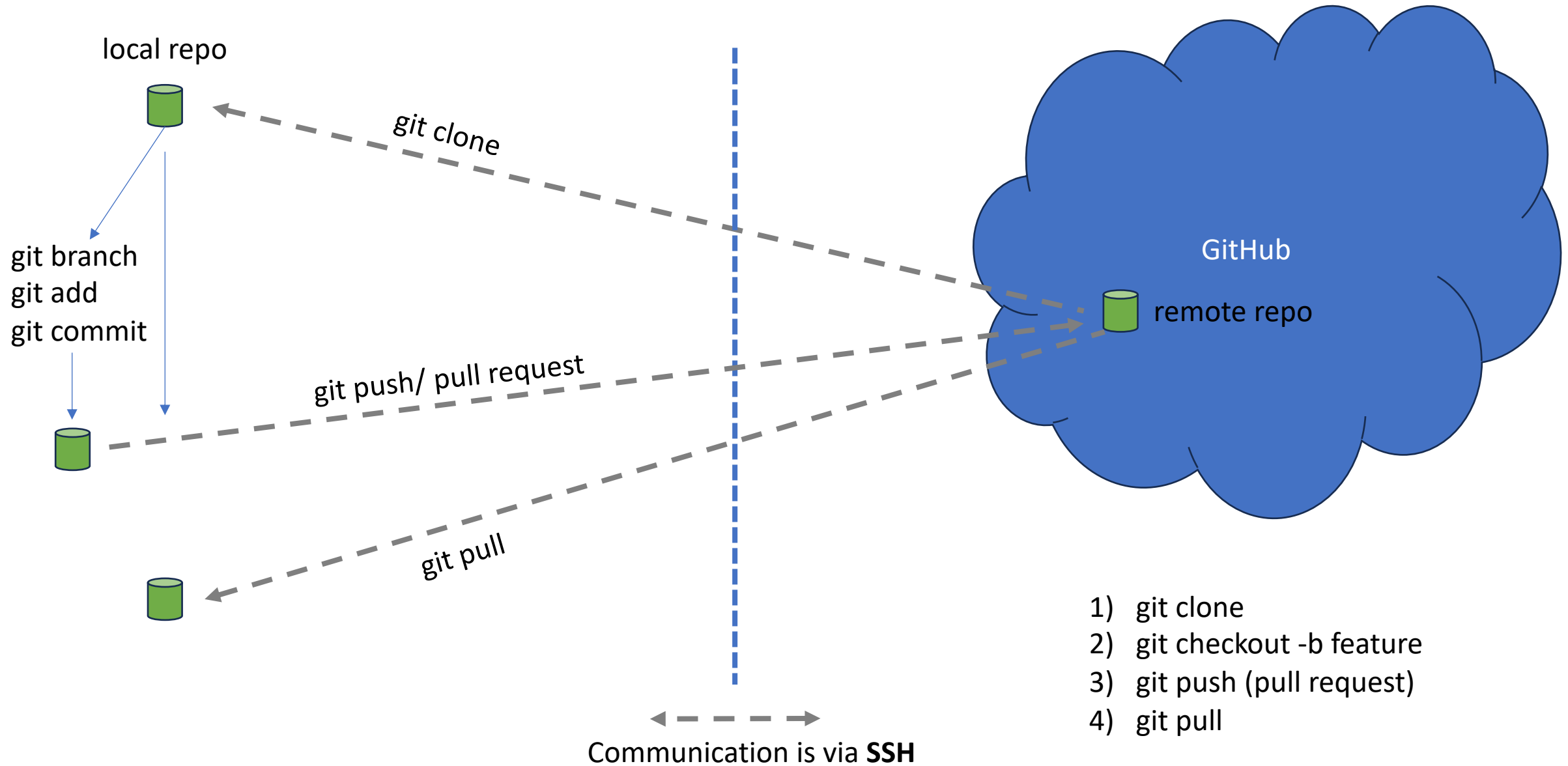
<https://github.com/>



Online Collaboration Tools



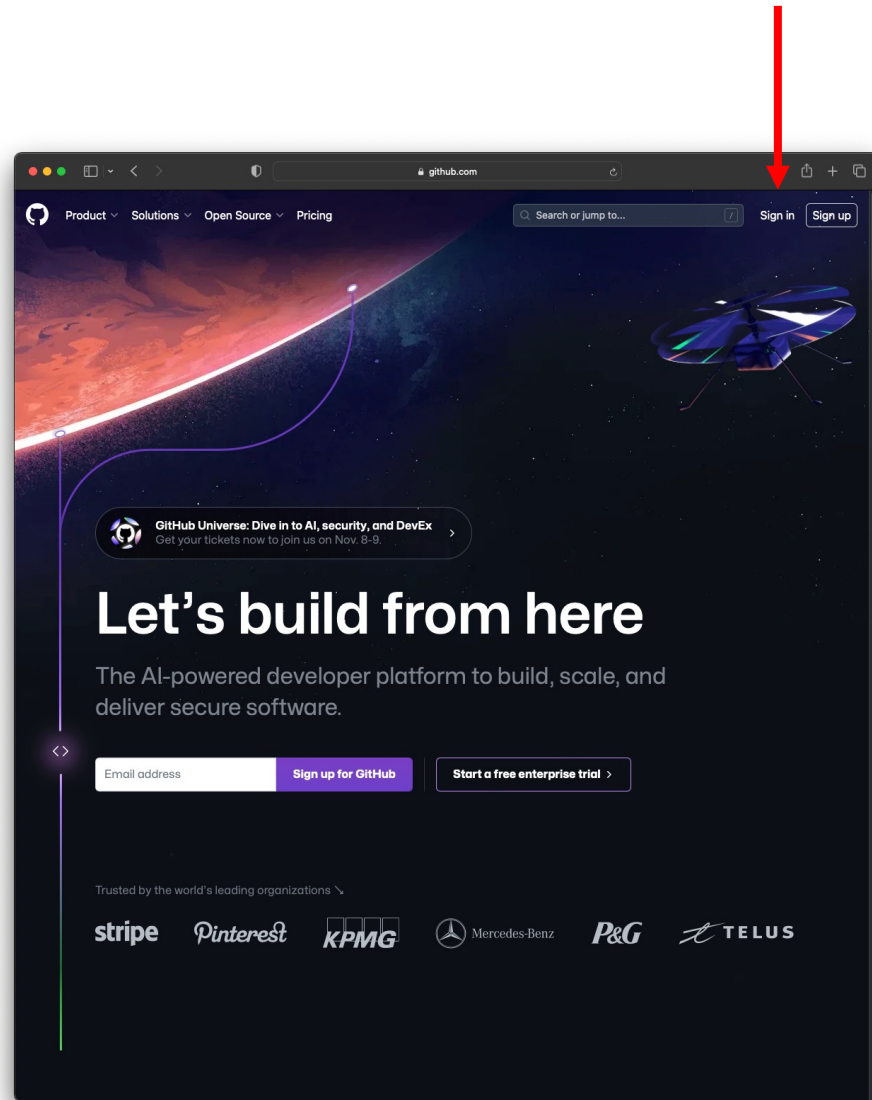
GitHub Workflow



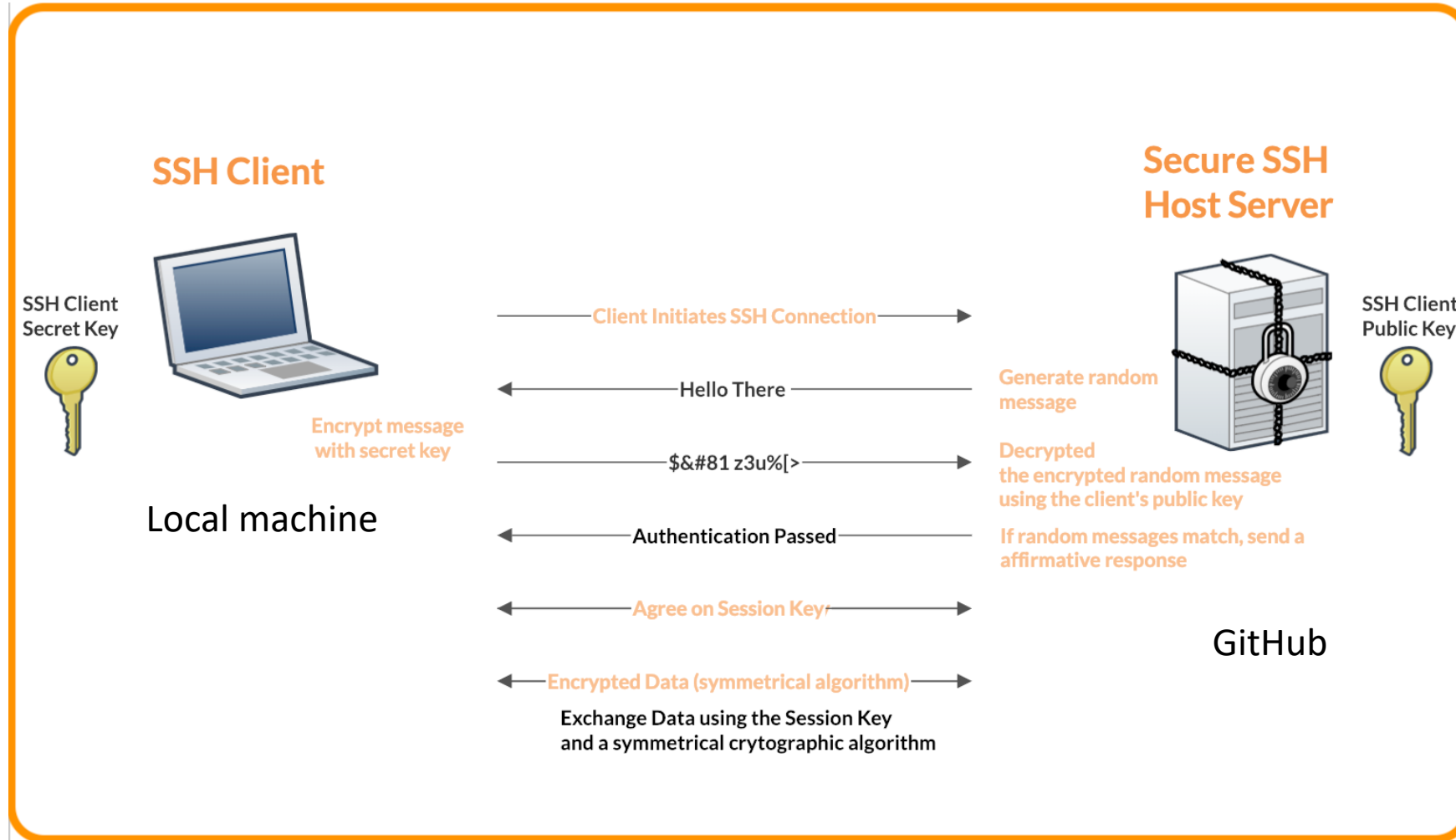
GitHub account

Exercise

Create a [GitHub](#) account



SSH Keys



SSH Keys

Exercise

Run the following command on your terminal¹:

```
$ ssh-keygen -t rsa -b 4096
```



Create a new SSH key using a default label

```
$ ssh-keygen -t rsa -b 4096 -C "youremail@domain.com"
```

Create a new SSH key using the email as label

This will create two files

<i>id_rsa</i>	←	private key
<i>id_rsa.pub</i>	←	public key

Then in your GitHub account, go to  Settings (upper right-hand corner), then in  SSH and GPG keys create a new SSH key entry and copy the **contents*** of your **public key**.


¹ Run `ls -la ~/.ssh` to check if you have any existing keys.

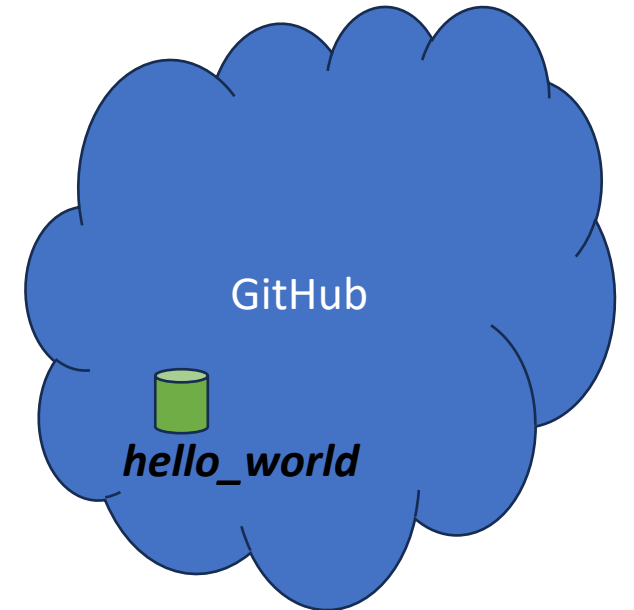
*make sure that you copy the contents exactly!

Creating a GitHub repository

Creating a Github Repo

Exercise

1. Go to [GitHub](#) and log in to your account.
2. In your GitHub account click on the  at the upper right corner.
3. Select New repository.
4. In the box **Repository name**, type *hello_world*.
5. In the **Description** box, type a short description.
6. Select **Public** repository (allows anyone to access the repository).
7. Select **Add a README file**.
8. Go to the bottom right corner and click on **Create repository**.



Working on Github

Exercise

On your new GitHub repository page, click on the  icon to edit the *README.md* file and replace its contents with the following:

My First and Simple Git Repository

This is my first attempt to create a Git repository on GitHub.

The goal here is to learn how to create and maintain a repository.

I am interested in (among other things):

- Creating a branch*
- Working on the branch and committing my changes.*
- Merging the branch to main.*

README.md is written using **Markdown** syntax.

Markdown cheat sheet is [here](#).

Click on **Commit changes...** to save the file.

Add a new file

Exercise

- On the main page of the repository, click on **Add file**
- Select **Create new file**
- In the box on the right of *hello_world*, type *hello.py*, and
- In the **Edit** box, add the following:

```
print("This is my first Git repository on GitHub.")
```

Click on **Commit changes...** to save the file and update the repository.

Note that the repository now has two files: *README.md* and *hello.py*.

Creating a Branch

Exercise

For this and the next exercise we are going to do all the operations within GitHub.


- Click on the dropdown menu that says ***main***.
- In the text box, type a branch name, call it *github_edits*.
- Click on ***Create branch: github_edits*** to create the new branch.

You should now have two branches: *main* and *github_edits*.

Add content to the branch

Exercise

Let's modify the file *hello.py* in *github_edits* and commit the changes there.

- Under the *github_edits* branch you created, click the *hello.py* file.
- To edit the file, click on the  icon.
- In the editor, include the following below the existing line:

```
def print_message(name='GitHub'):
    print(f"Welcome to the {name} tutorial!")
print_message()
```

Click on **Commit changes...** to save the file.

Since changes were made to the *hello.py* file that resides in the *github_edits* branch, the *main* branch will not have those changes.

Cloning a GitHub Repo

There are two ways to clone a remote repo: using HTTP or SSH protocols.

HTTP is an easy way to allow anonymous, **read-only** access to a repository

`git clone https://github.com/user/repo.git` ← HTTP URL

It is generally not possible to push commits to an HTTP address.

For **read-write** access, you should use SSH instead: ← SSH URL ✓

`git clone git@github.com:user/repo.git`

Cloning a GitHub Repo

Exercise

- Click on ***hello_world*** on the upper left corner
- Click on the **Code** box.
- Select the **SSH option** and copy the address in the box. It should look like:

git@github.com:userid/hello_world.git

where ***userid*** is your account name in GitHub.

From a terminal, issue the command:

git clone git@github.com:userid/hello_world.git

You should now have a local repo which is a clone of the remote repo.

Working on the Local Repo


Exercise

Go to your local repo and check your branches:

```
git branch -a
```

Create a new branch in your local repo as follows

```
git checkout -b local_edits
```

Open the file *hello.py* and include this code  `print("Hello Earth!")`
above the existing one.
`print()`

Add and commit the file as follows:

```
git add hello.py
```

```
git commit -m "Added a new code segment."
```

Pushing to Github

Exercise

The branch *local_edits* is on your **local** repository only. We want to push it to the **remote** repository on GitHub. This is done with the command:

```
git push --set-upstream origin local_edits
```

git push is the basic Git command used to push changes from your local repository to a remote repository.

--set-upstream (or simply *-u*) sets the remote branch as the *upstream* branch for your current local branch.

origin is the name of the **remote** repository. In Git, "origin" is a common default name for the remote repository from which you initially cloned your local repository. You can have multiple remotes with different names if needed.

local_edits is the name of the branch you want to push to the remote repository.

Pushing to Github

Exercise

Refresh your browser and check what is happening in your *hello_world* repository:

There are now three (branches):

- *main*
 - *github_edits*
 - *local_edits*
- } ***feature branches***

Note that each of these branches has a different version of the file *hello.py*. Therefore, any attempt to merge any of the **feature** branches into *main* may result in conflicts.

Finally, note that you can push to *any** remote repo, not just GitHub.

Collaborating

Pull Requests

A *pull request* (**PR**) is a fundamental concept in the process of collaborative software development using Git. If you make a *pull request*, you are *requesting* that a project maintainer *pull a branch* from your repo into their repo. This is how it works:

1. You create a feature branch in your local repo ✓
2. You make changes that you want to contribute ✓
3. You push your changes to the GitHub repo ✓
4. You open a PR
5. Upstream collaborators review and alter your changes
6. The project maintainer merges the feature branch and closes the PR




On GitHub, pull requests show differences in the content from both branches. The changes, additions, and subtractions are shown in different colors.

Starting a Pull Request

Exercise

You can open PRs in your own repository and merge them yourself. It's a great way to learn the GitHub flow before working on larger projects.

Select the *github_edits* branch.

Click the  Pull requests tab of your *github_edits* repository.

Click **New pull request**

In the **Compare & pull request** box, select the *github_edits* branch you made, to compare with *main*.

- Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.
- If you are satisfied with the changes, click **Create pull request**.

Give your pull request a title and write a brief description of your changes. 

Optionally, to the right of your title and description, You may select Reviewers, Assignees, Labels, Projects, or Milestone to add any of these options to your PR.

Click **Create pull request**.

Merging a Pull Request

Exercise

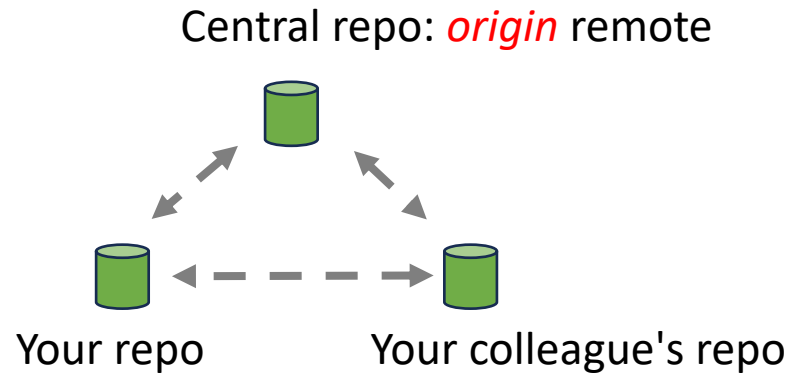
If the branch *github_edits* has no conflicts with the base branch:

- Click **Merge pull request**
- Click **Confirm merge** and add comment before clicking (if necessary).

Verify in *main* if the merge was successful.

- Go to the *main* branch.
- Click on the file *hello.py*.
- Verify that *hello.py*

Syncing



Users usually share branches

When you clone a repository with *git clone*, it automatically creates a *remote* connection called *origin* pointing back to the cloned repository.

When collaborating we may have multiple connections

git remote: lets you create, view, and delete connections to other repositories

Syncing

git remote commands

git remote: list connections you have with other repos

git remote -v: like above but also include URLs

git remote add <name> <URL>: create a new connection using <name> to access <URL>

git remote rm <name> : remove a remote

git remote add <old-name> <new-name> : rename a remote

A ***remote*** connection is NOT a direct link to a repository but just a name used to reference a URL

Syncing

Exercise

Run the following commands on your local repo:

```
$ git remote
```

```
$ git remote -v
```

```
$ git remote add cacruz-hello-world git@github.com:cacruz/hello_world.git
```

```
$ git remote -v
```

Run the following command on your local repo's top-level directory:

```
$ cat .git/config
```

git fetch

Downloads commits, files, and branches from a remote repo to your local repo.

You run git fetch to examine what all your collaborators have been working on.

git fetch is a safe way to review commits before merging them with your local repo

git fetch <remote>: fetch all branches from the remote repo

git fetch <remote> <branch>: fetch <branch> from the remote repo

git fetch --dry-run: perform a demo run

git pull

Download content from a remote repo and immediately update your local repo to match that content.

git pull will download the remote content for the active local branch and immediately execute *git merge*.

git pull = git fetch + git merge

If you have pending changes in progress this will create **conflicts!**

git pull <remote>: pull all branches from the remote repo

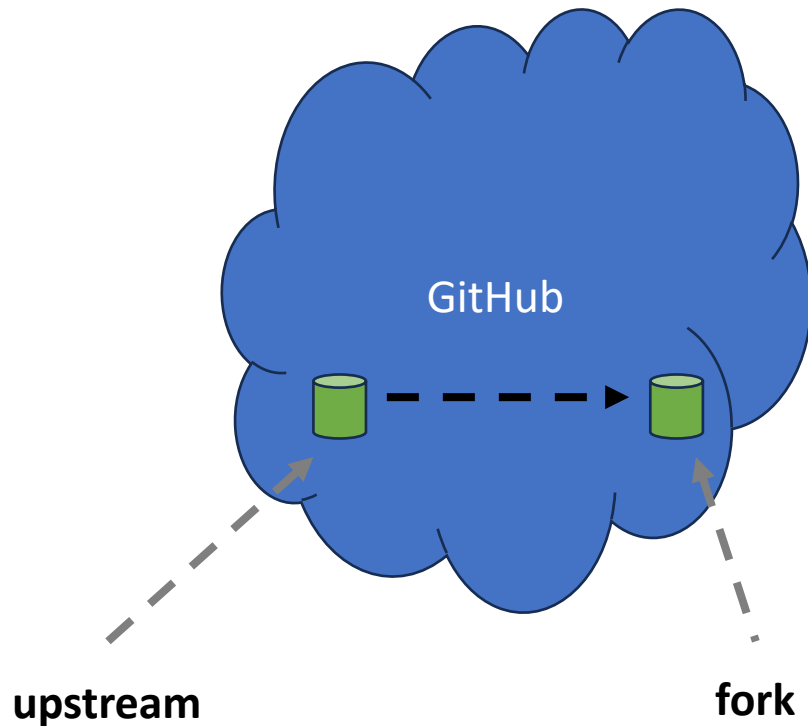
git pull --verbose: print detailed output during pull

git pull --no-commit: same as *git pull* but does not create a new merge commit

Fork



copying the code from someone else's repository to yours.



Fork to Collaborate


- Create a fork
- Work on the changes you want to implement
- Create a PR to merge the changes you've made from your fork into the original repository.
- Upstream developers need to approve your changes before your changes are included.

Forks create a safe environment for the original repository. No matter how much experimentation you do in your own fork, the original repository is not affected at all!

Working with Forks

Exercise

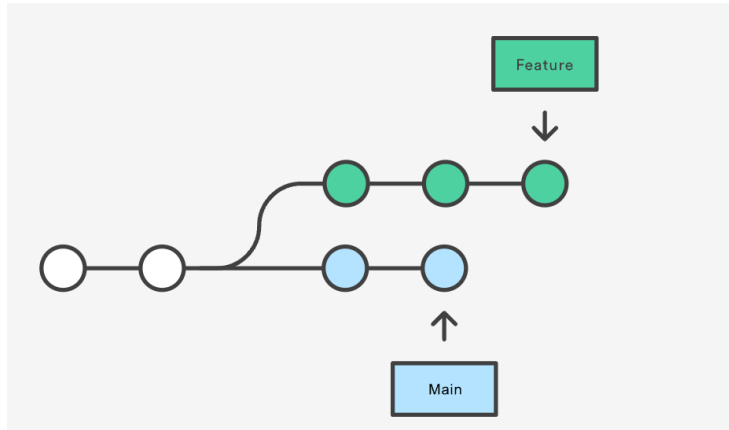
- Go to your GitHub account and fork the following repo: <https://github.com/cacruz/hello>
- Clone your fork to your computer (use SSH protocol)
- Create a feature branch and add your contribution.
 - Can you add an additional entry to the list?
- Push to your fork.
- Create a pull request.

Sometimes you may be able to push directly to the upstream repo, e.g.  True, if you are a "collaborator" with "push" permissions

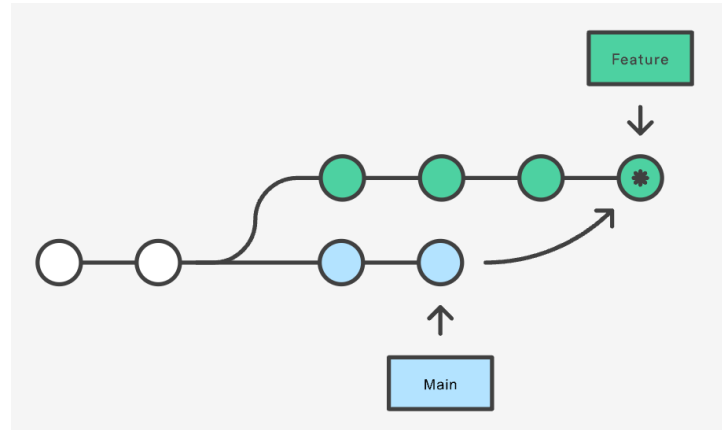
```
git remote add cacruz-hello git@github.com:cacruz/hello  
git push upstream hello-feature
```


Selected topics

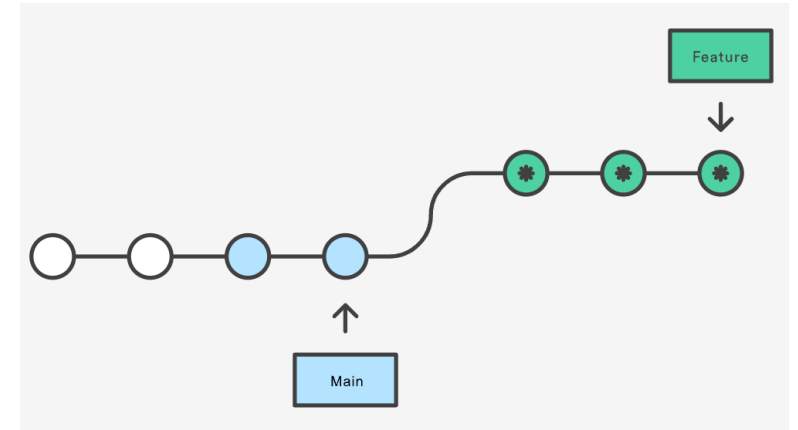
git rebase



branching



merging (* is a new commit)

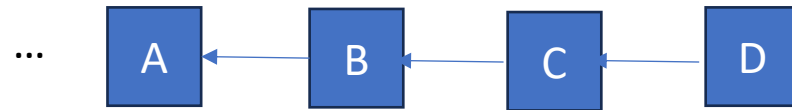


rebasing

git rebase is the process of moving or combining a sequence of commits to a new base commit. Like *git merge* but *rebase* rewrites history

git rebase <branch>

Squash commits



- * ac7dd5f ... Commit D (HEAD -> master)
- * 5de0b6f ... Commit C
- * 54a204d ... Commit B
- * c407062 ... Commit A

After Squashing commits B, C, and D:



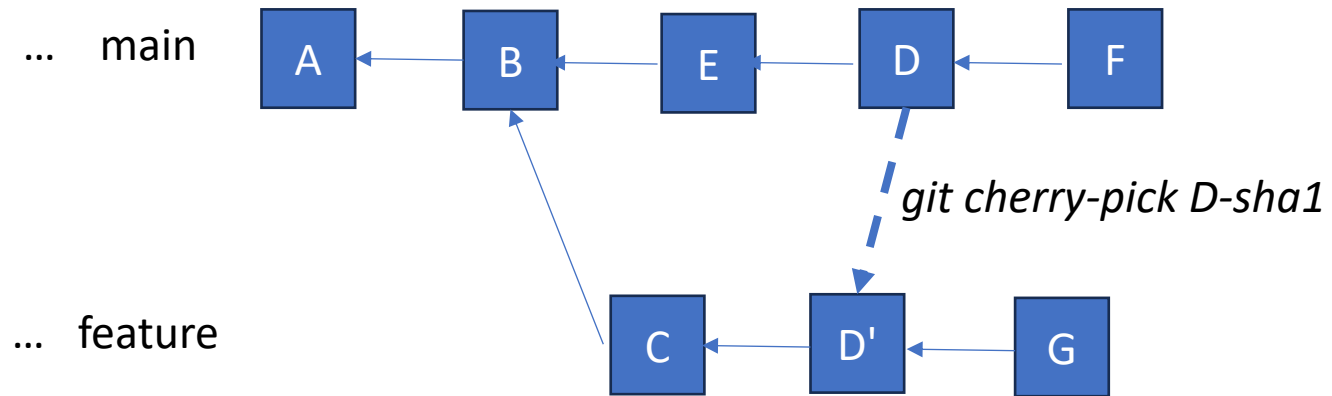
(The commit E includes the changes in B, C, and D.)

```
git rebase -i HEAD~4
```

```
git merge --squash new-feature
```

```
$ git rebase -i HEAD~4
[detached HEAD f9a9cd5] Commit A
Date: Mon Aug 23 23:28:56 2021 +0200
1 file changed, 1 insertion(+), 1 deletion(-)
Successfully rebased and updated refs/heads/master.
```

git cherry-pick



Like rebasing, but mainly used if you don't want to merge the whole branch and you want some of the commits.

- **git tag** creates refs that point to specific commits in Git history. A tag is like a branch but without history.

git tag -a 1.0 -m "first version"

- **git clean** removes untracked files from the working directory

git clean -n . <<< dry-run

git clean -f src

git clean -f -x

- **git archive** creates an archive file from specified Git Refs

git archive --output=./repo.tar --format=tar HEAD

- **gitk** is a graphical history browser installed with Git core packages

gitk (launches a git-log-like GUI)