Recitation

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Version Control System

ITSC 3155 – Software Engineering
Department of Computer Science
College of Computing and Informatics

Agenda

- What is Version Control?
- What is Git?
- Git vs. GitHub
- Git Operation level.
- What is Branch?
- Git & GitHub Flow
- Initializing
- Merging Branches
- Reverting Changes
- Teamwork Best Practice
- Git Commands
- Conventional Commits
- Activity Time

What is Version Control?

Practice of tracking and managing changes to software code.

Collaboration

Manage Changes

Centralized Code

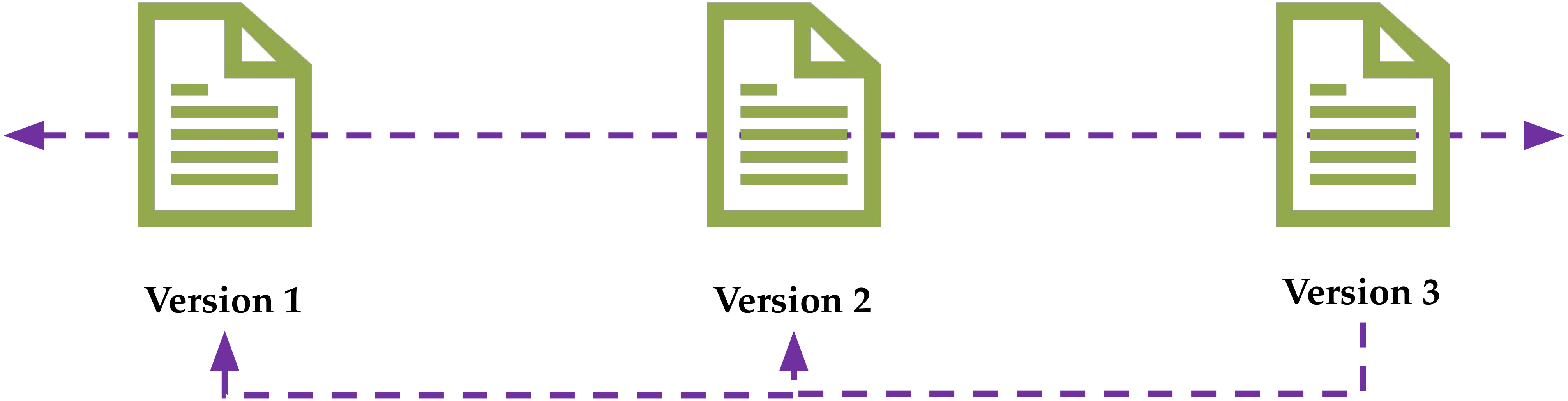
Accelerate Production

What is Git?

Git is a free and open-source Version Control System (VCS) that can be used to manage all types of projects, from small to large, with speed, efficiency, and transparency.

Record changes made to a file overtime
Revert or Recall changes

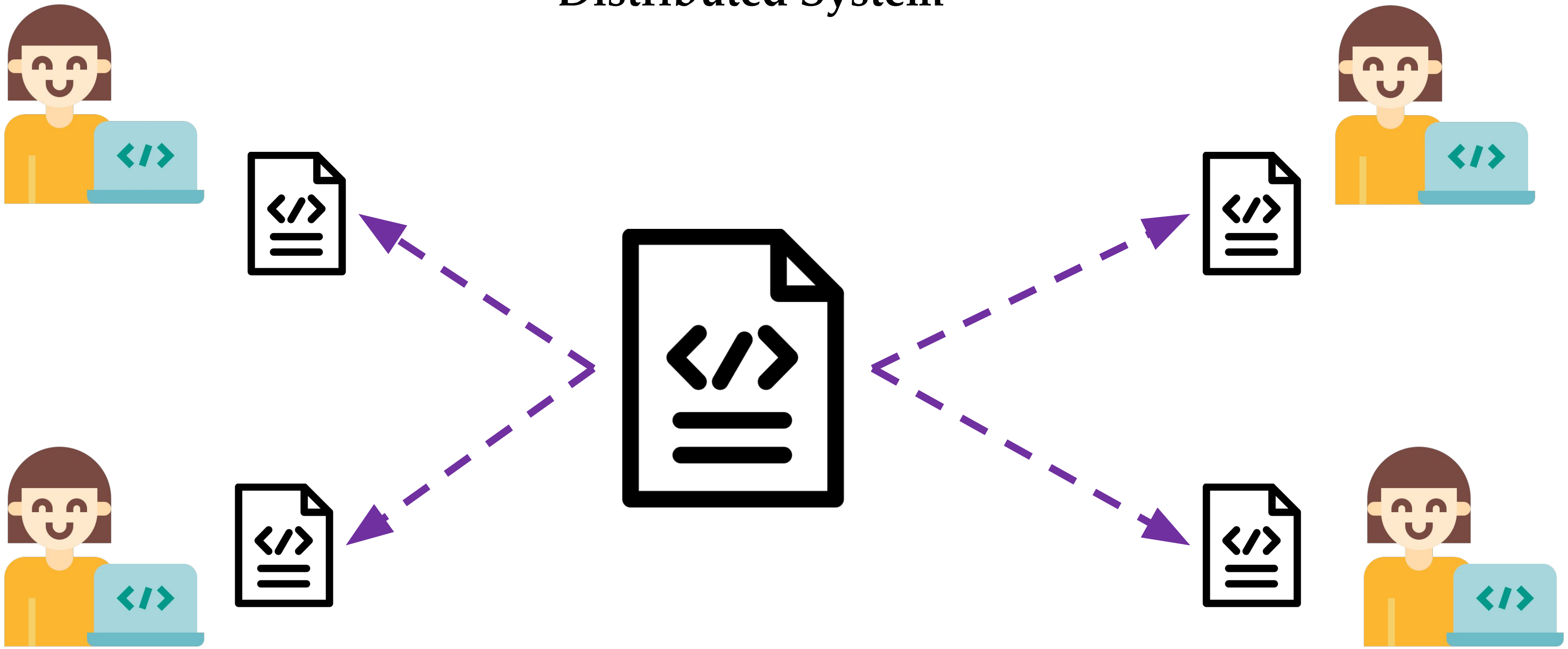
What is Git?



What has changed? Who has made the change? When was it changed?

What is Git?

Distributed System

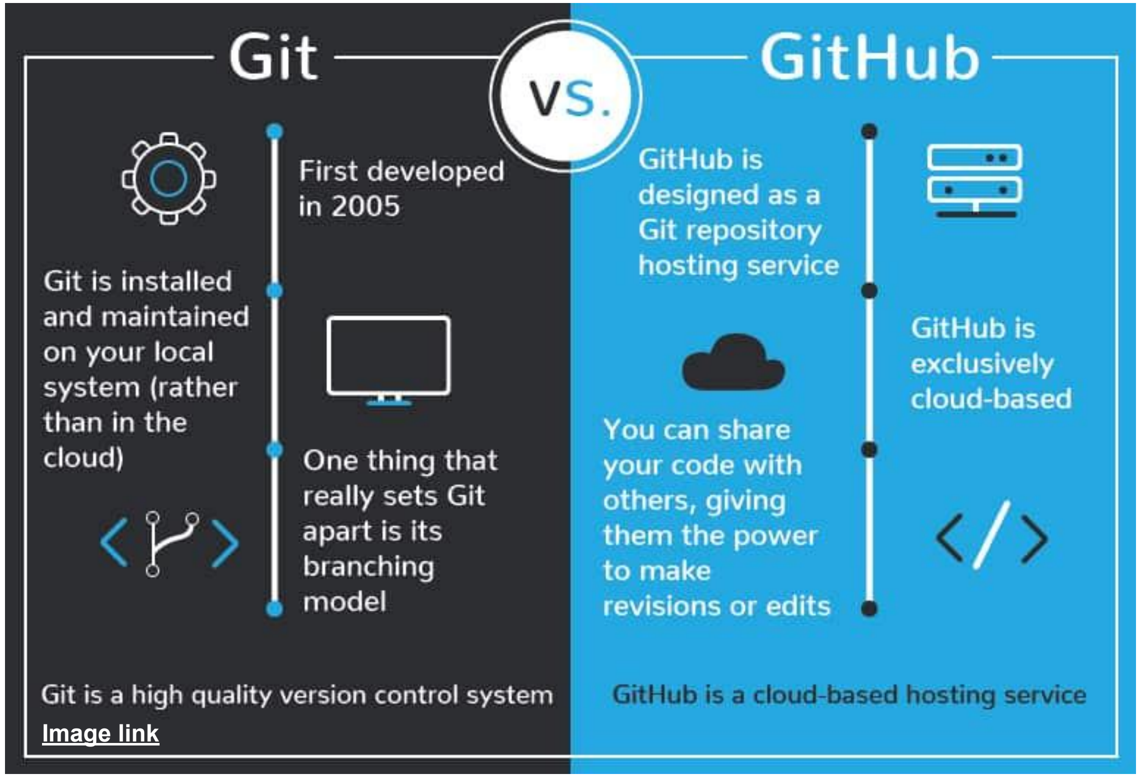


A distributed control system like Git allows collaboration between many clients simultaneously within a single project, enabling many different project workflows that are not supported by traditional control systems.

What is GitHub?

A collaboration platform built on the top of Git

- Builds a platform for developers to create, share, and grow.
- Fosters a community for developers and supports open source.
- Supports beginners and experienced developers



First developed in 2005

Git is installed and maintained on your local system (rather than in the cloud)

One thing that really sets Git apart is its branching model

GitHub is designed as a Git repository hosting service

GitHub is exclusively cloud-based

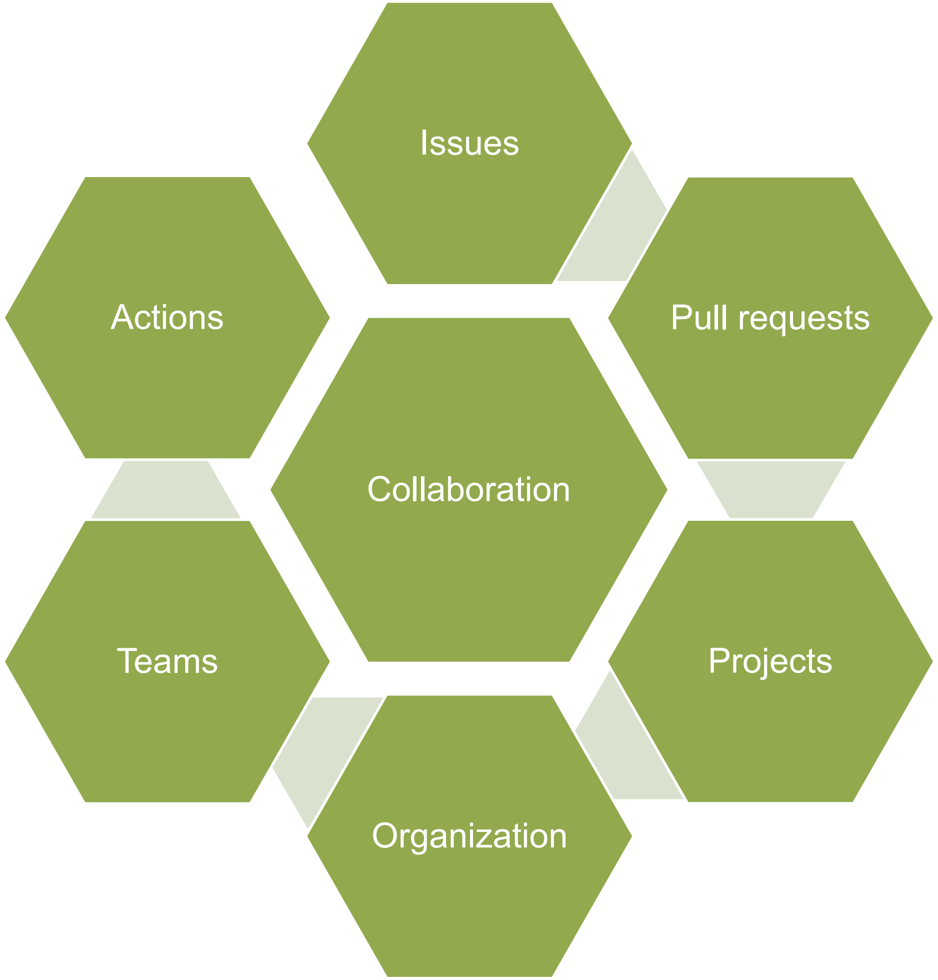
You can share your code with others, giving them the power to make revisions or edits

Git is a high quality version control system

GitHub is a cloud-based hosting service

Image link

What is GitHub?



Where Should I Run Commands?

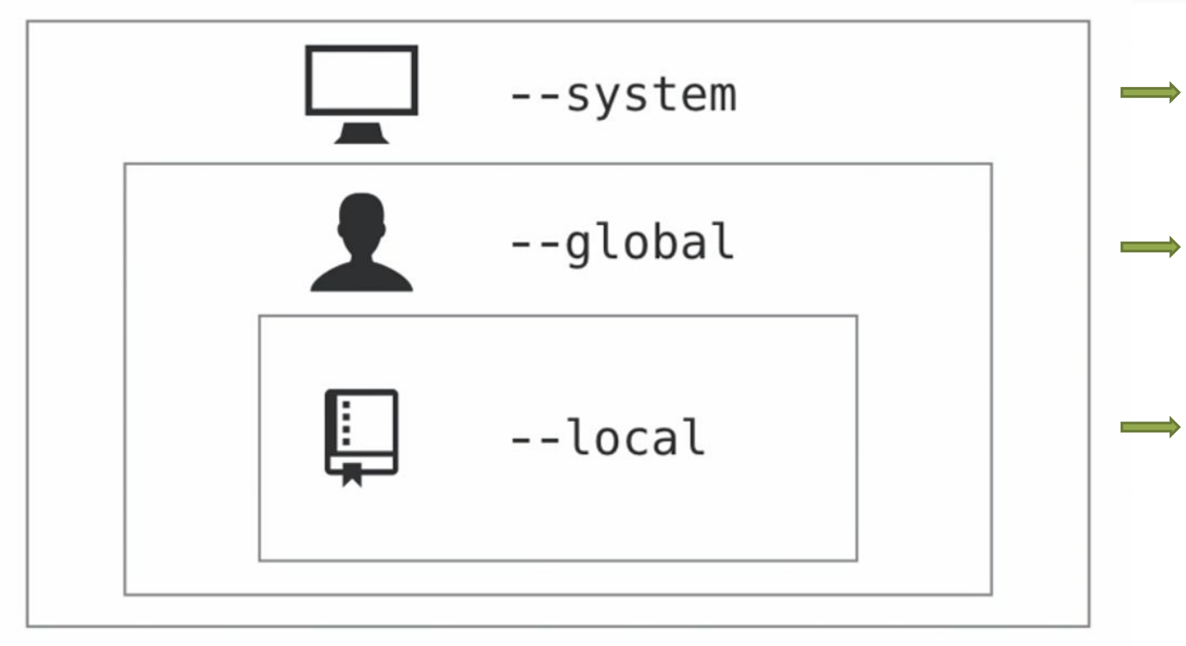
In the past it was preferable to execute Git commands on the command line, where you have complete freedom and flexibility to make the necessary changes.

Command Line
GUIs
IDEs

Why Command Line?

- Run all Git commands in their full functionality
- Knowing Git on the command line transfers over other tools.
- Less likely to change

Git Configuration Levels



All users on a machine

Current user

Current repository

```
git config --global user.name ”username"

git config --global user.email "email@example.com"
```

Operation levels

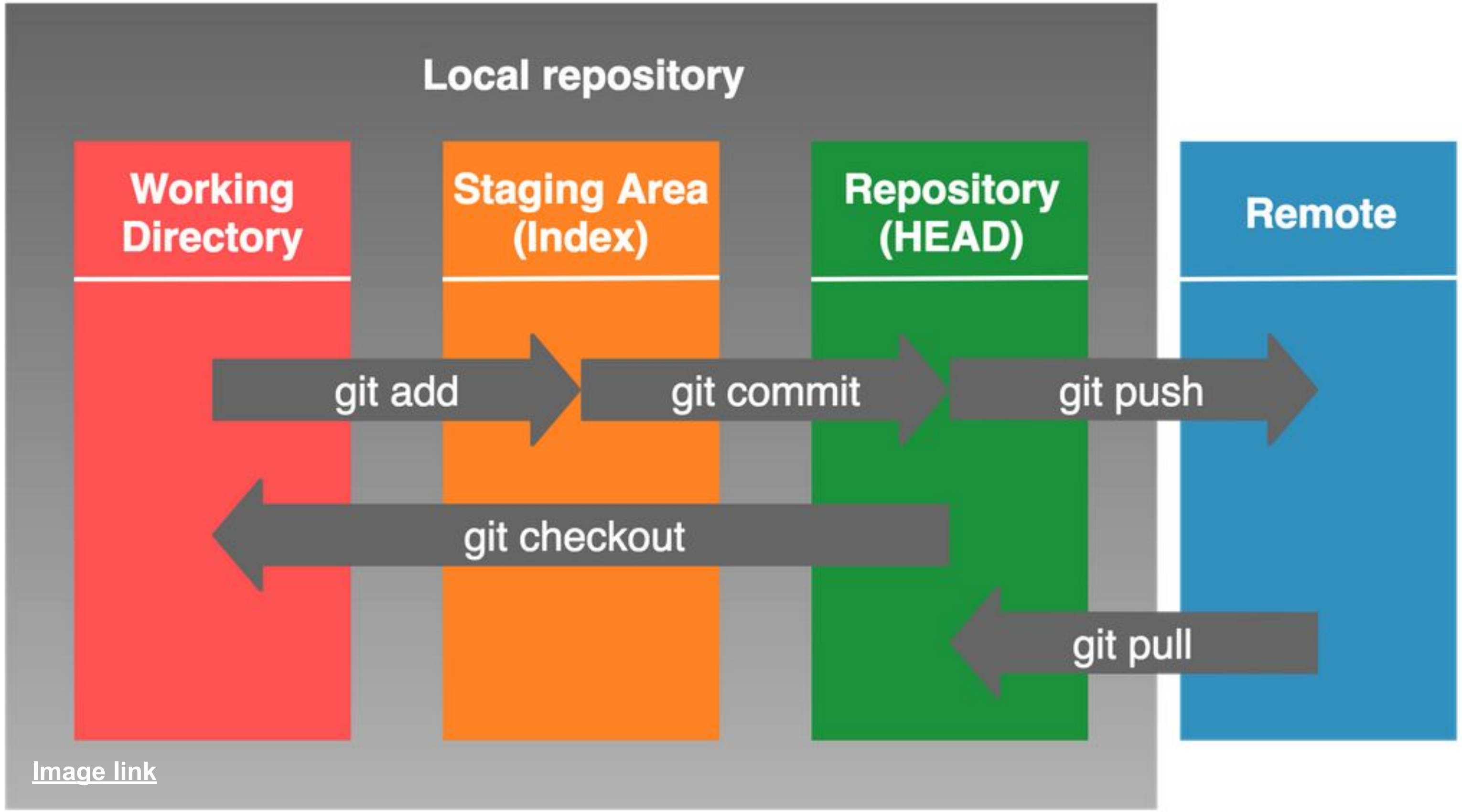
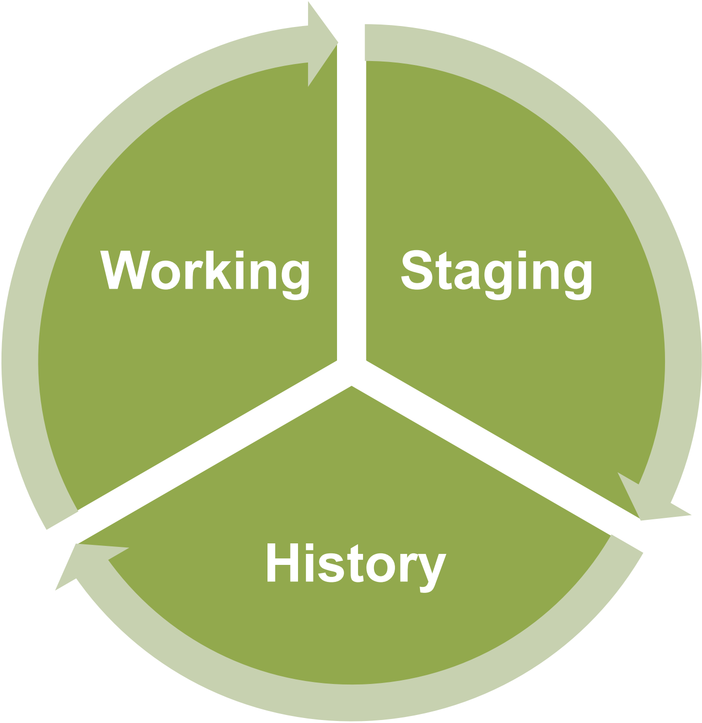
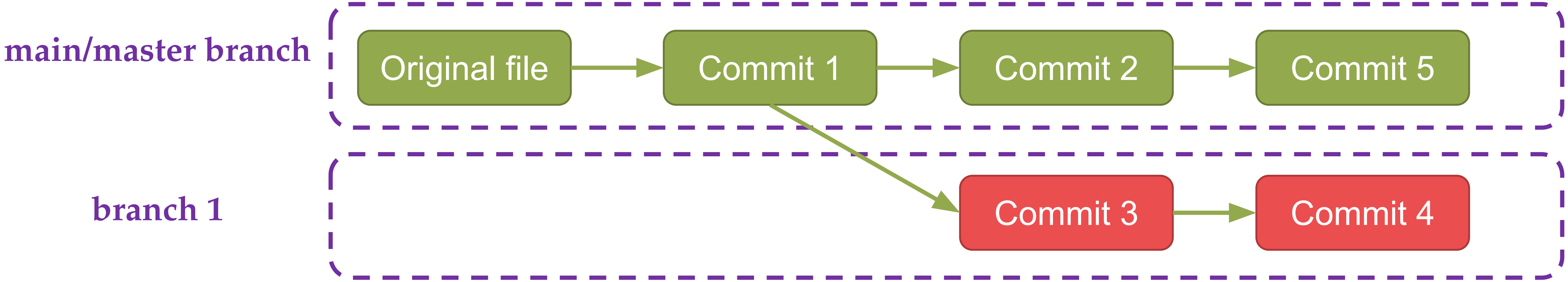


Image link



Three states of storing files

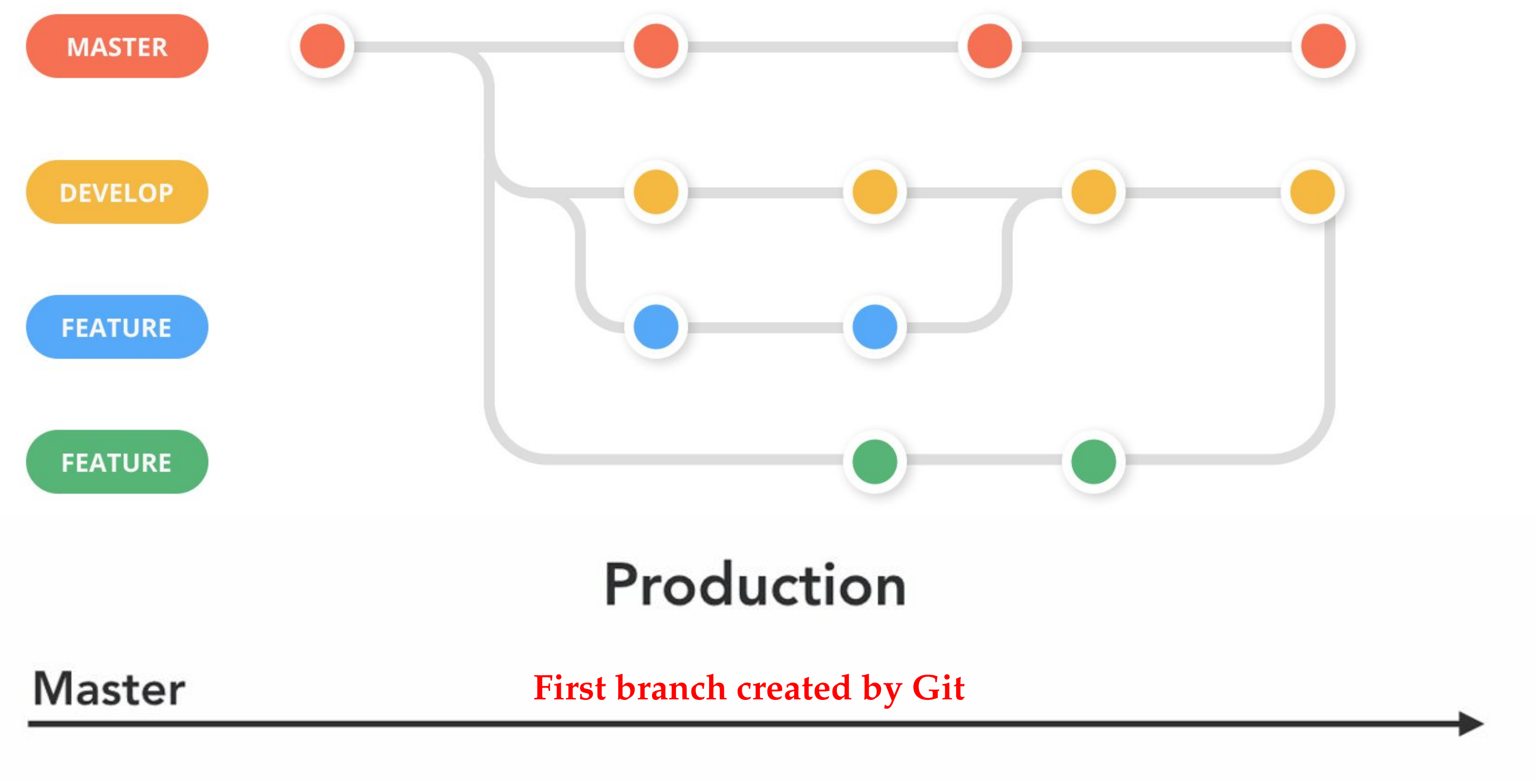
What is Branch?



- `git branch <branch_name>`
- `git checkout <branch_name> -> set HEAD`
- `git push origin <new_remote_branch>`

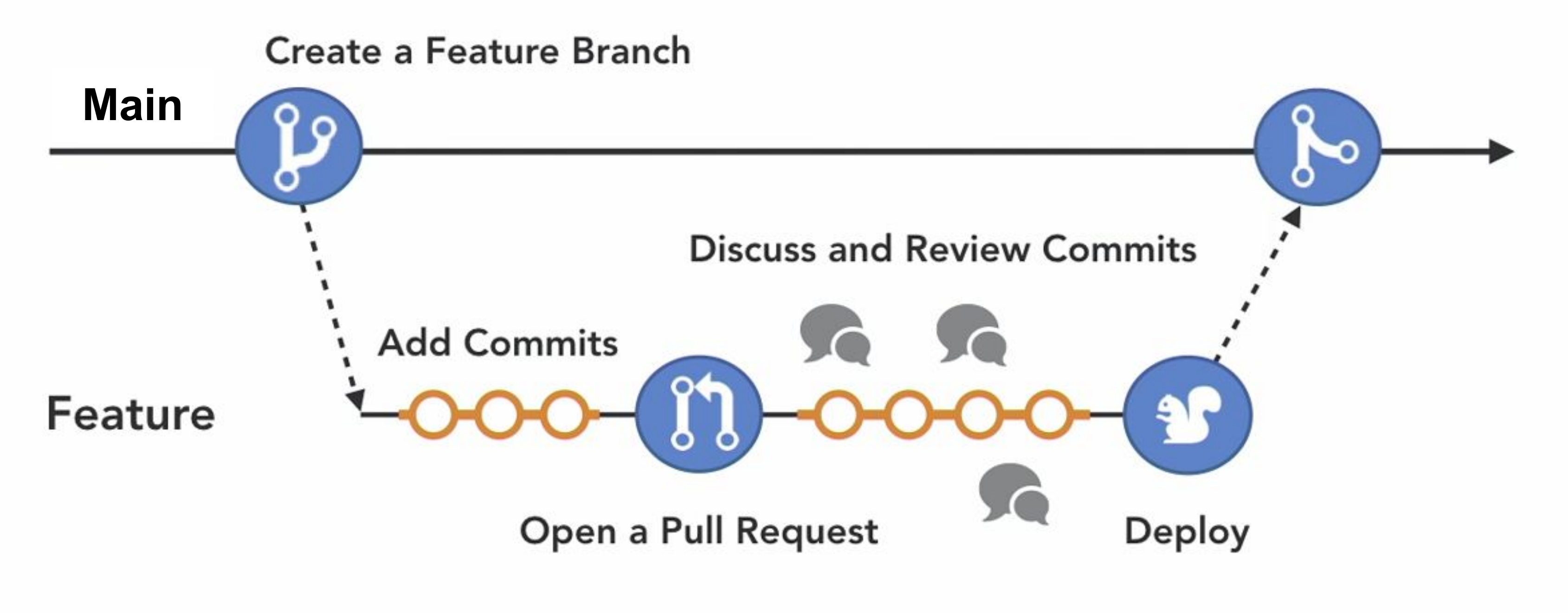
Git doesn't commit to new branch automatically. Commits show which changes belong to which branch.

Git Flow



First branch created by Git

GitHub Flow



With pull request, we show changes we've made that we want to add them to the production branch.

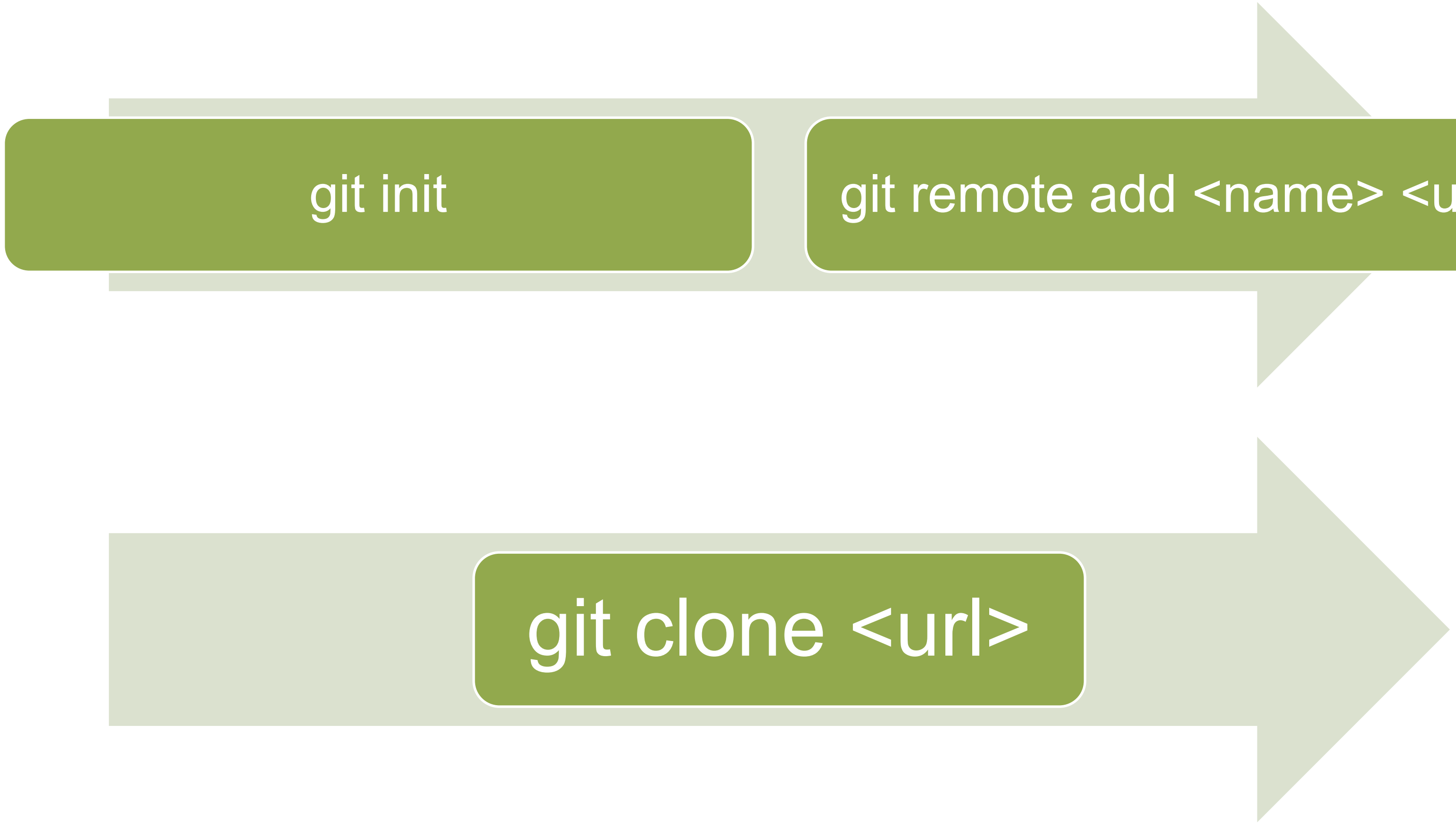
Everyone can view and discuss the changes by putting comments

Open pull requests early! To get feedbacks

Initializing Git

Form Local

Form Remote



git init

git remote add <name> <url>

git clone <url>

Merging Branches

Fast Forward Merge

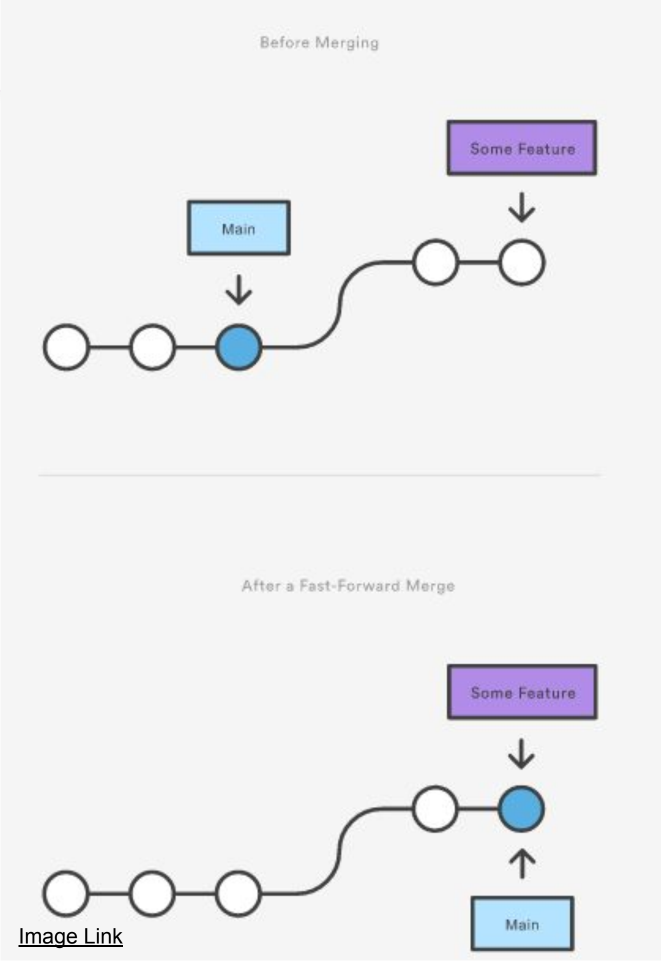


Image Link

3-way Merge

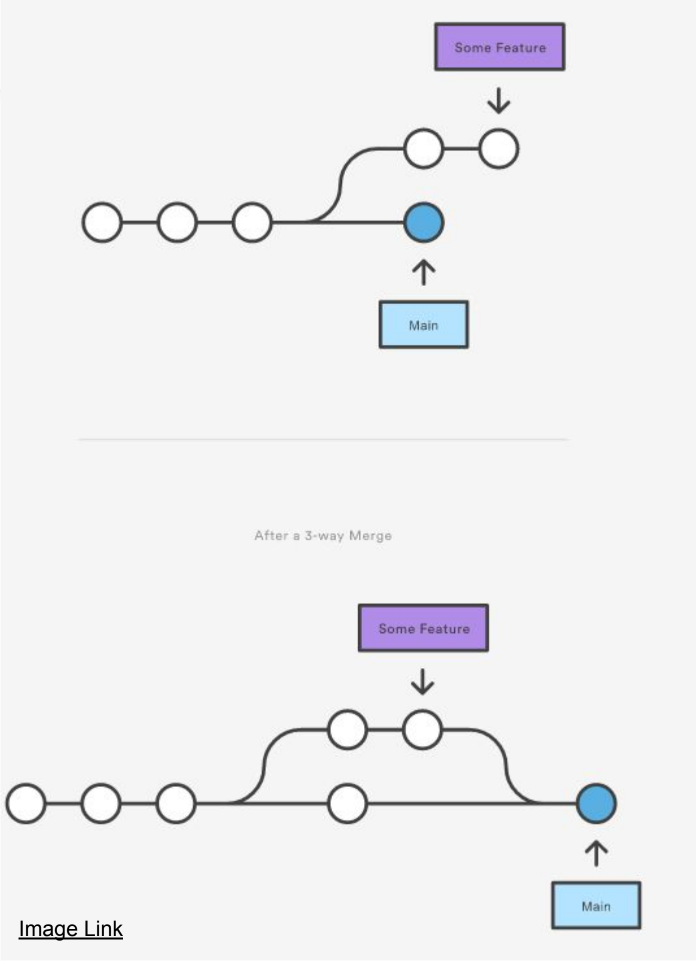


Image Link

If Git encounters a piece of data that is changed in both histories it will be unable to automatically combine them. This scenario is a version control conflict and Git will need user intervention to continue.

Reverting Changes

- You're allowed to permanently delete commits before pushing them.
- `git reset <commit> -> delete all commits up to and including the referenced commit. (HEAD^, HEAD~2)`
- `--hard , --soft`

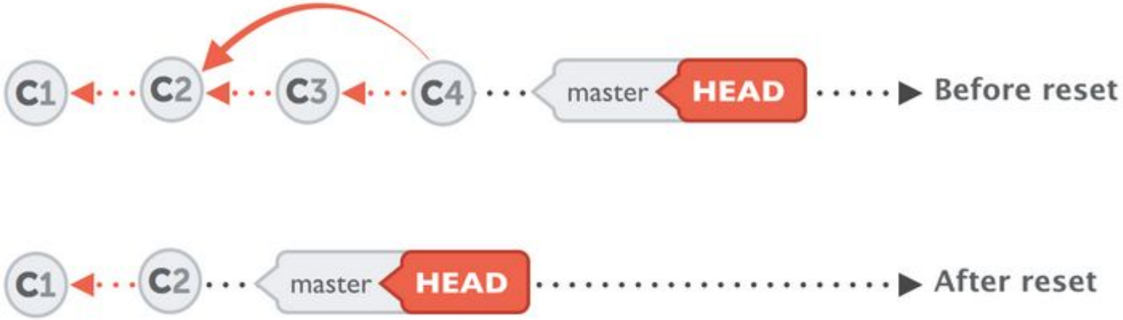


Image link

Reverting Changes

- You’re not allowed to permanently delete commits after pushing them.
- `git revert <commit> (HEAD, HEAD^, HEAD~2)`

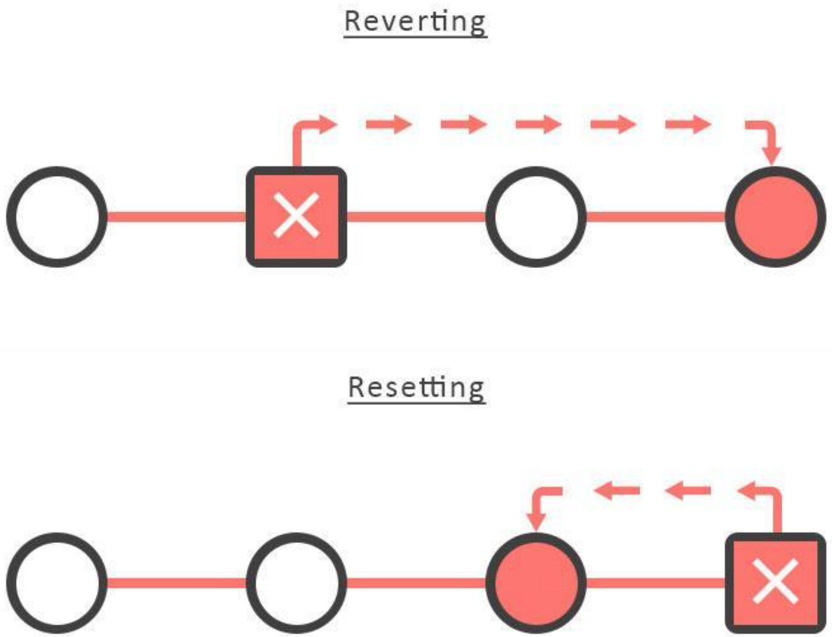
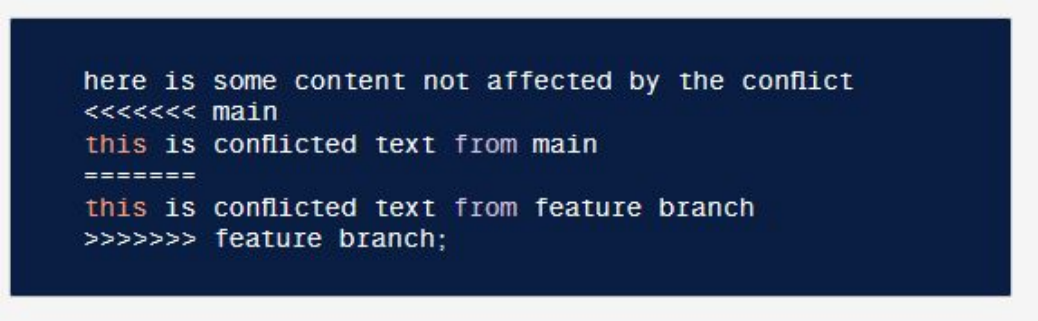


Image link



```
here is some content not affected by the conflict
<<<<<<< main
this is conflicted text from main
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

Teamwork Best Practice

- Each group works on a branch.
- Each branch is used for one task.
- After the task is done, it should be merged to the main branch.
- Delete branches after merging.

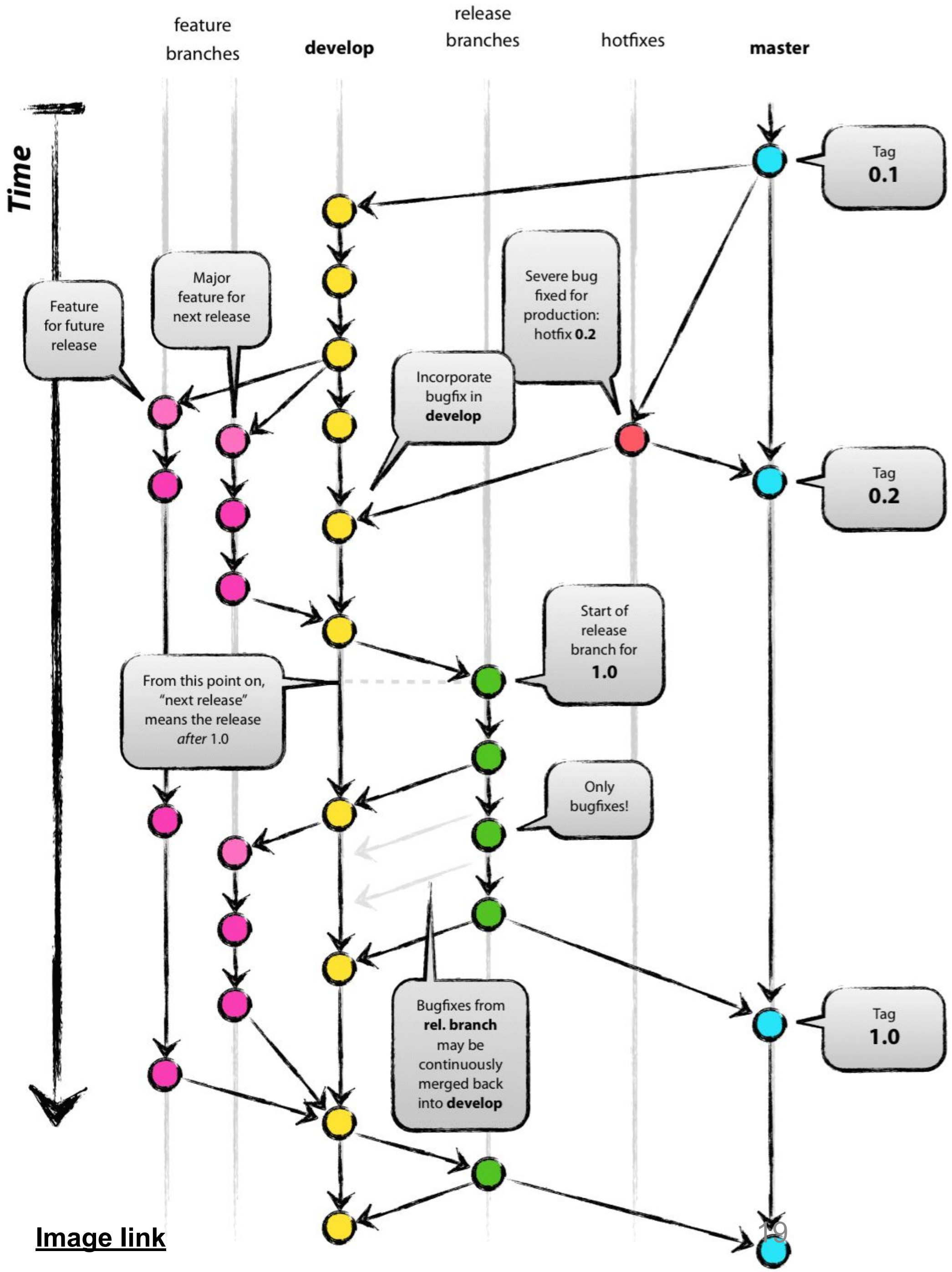
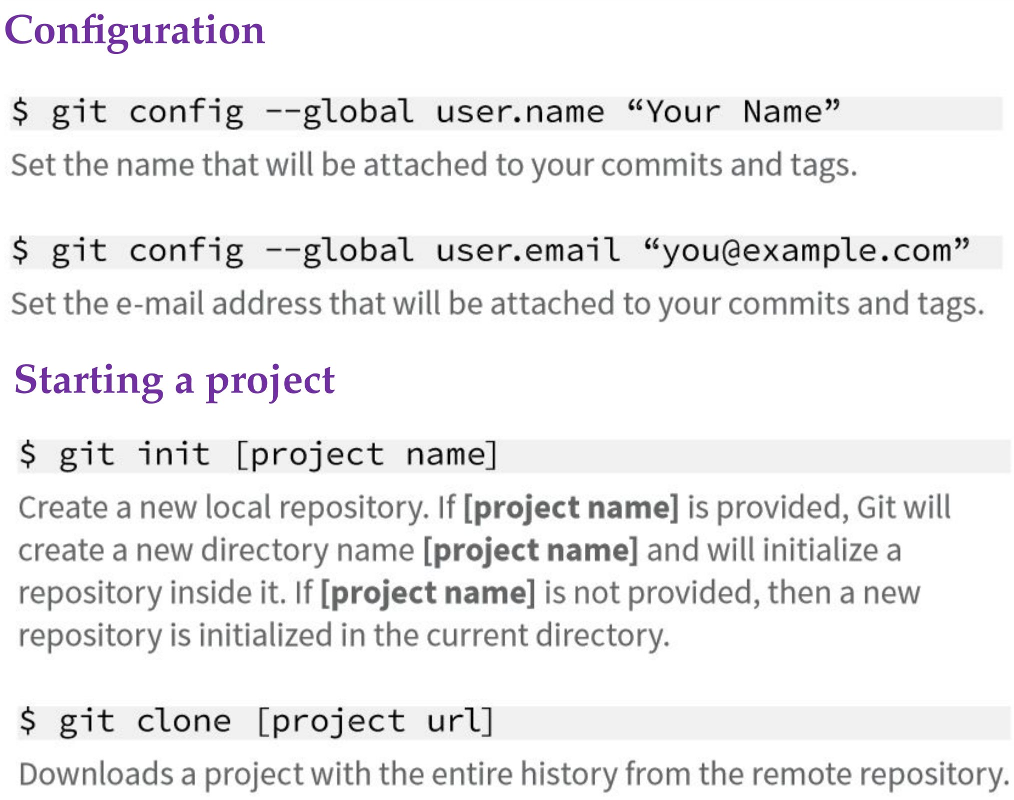

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Image link

Git Commands



Configuration

```
$ git config --global user.name “Your Name”
```

Set the name that will be attached to your commits and tags.

```
$ git config --global user.email “you@example.com”
```

Set the e-mail address that will be attached to your commits and tags.

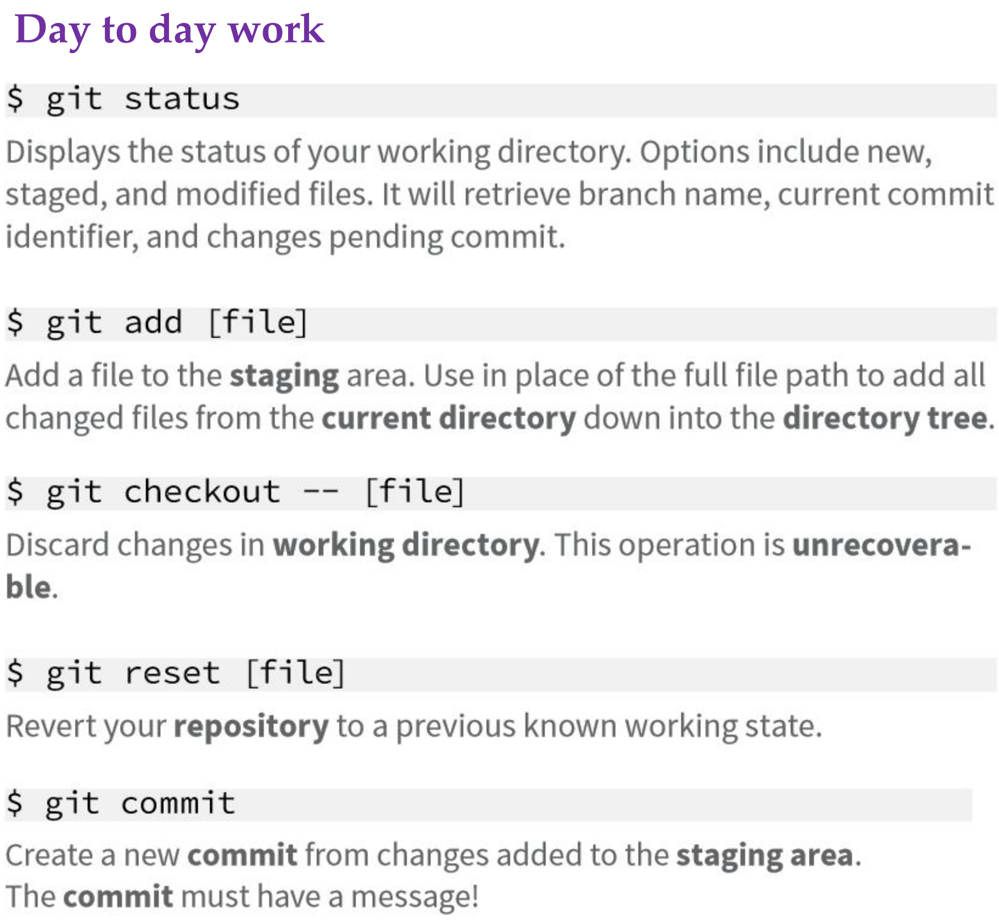
Starting a project

```
$ git init [project name]
```

Create a new local repository. If **[project name]** is provided, Git will create a new directory name **[project name]** and will initialize a repository inside it. If **[project name]** is not provided, then a new repository is initialized in the current directory.

```
$ git clone [project url]
```

Downloads a project with the entire history from the remote repository.



Day to day work

```
$ git status
```

Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.

```
$ git add [file]
```

Add a file to the **staging** area. Use in place of the full file path to add all changed files from the **current directory** down into the **directory tree**.

```
$ git checkout -- [file]
```

Discard changes in **working directory**. This operation is **unrecoverable**.

```
$ git reset [file]
```

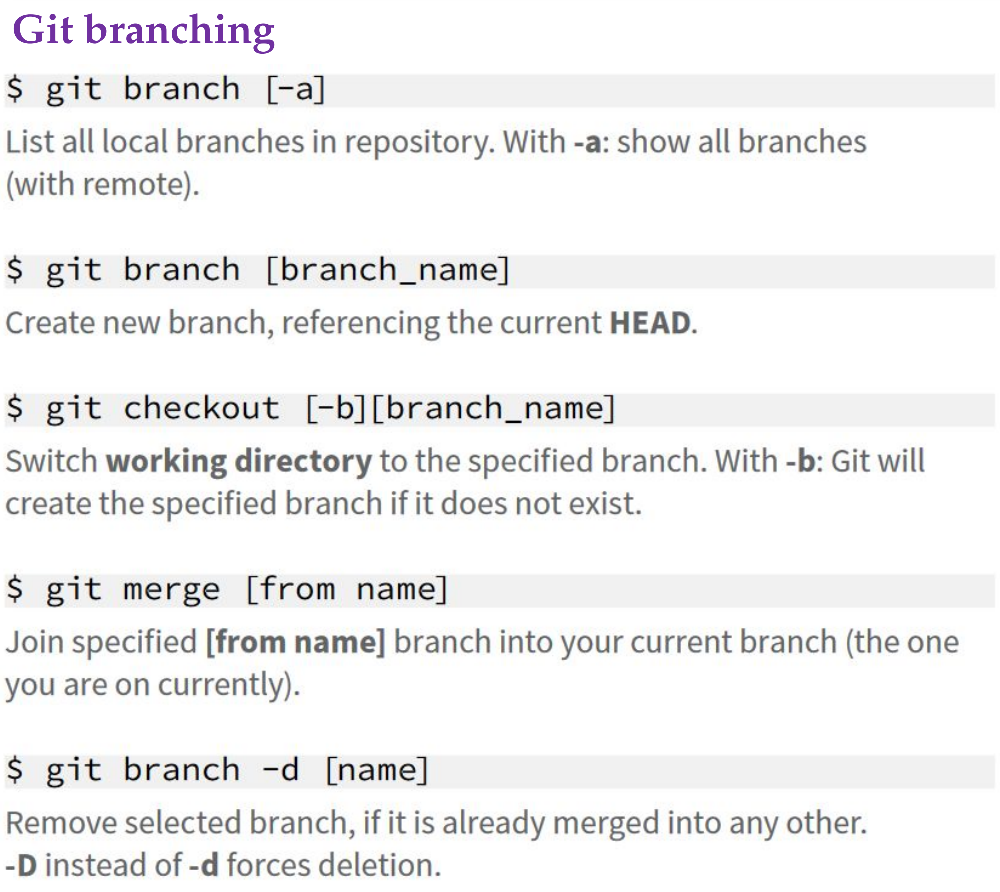
Revert your **repository** to a previous known working state.

```
$ git commit
```

Create a new **commit** from changes added to the **staging area**. The **commit** must have a message!

Gitlab Cheat Sheet

Git Commands



Git branching

```
$ git branch [-a]
```

List all local branches in repository. With **-a**: show all branches (with remote).

```
$ git branch [branch_name]
```

Create new branch, referencing the current **HEAD**.

```
$ git checkout [-b][branch_name]
```

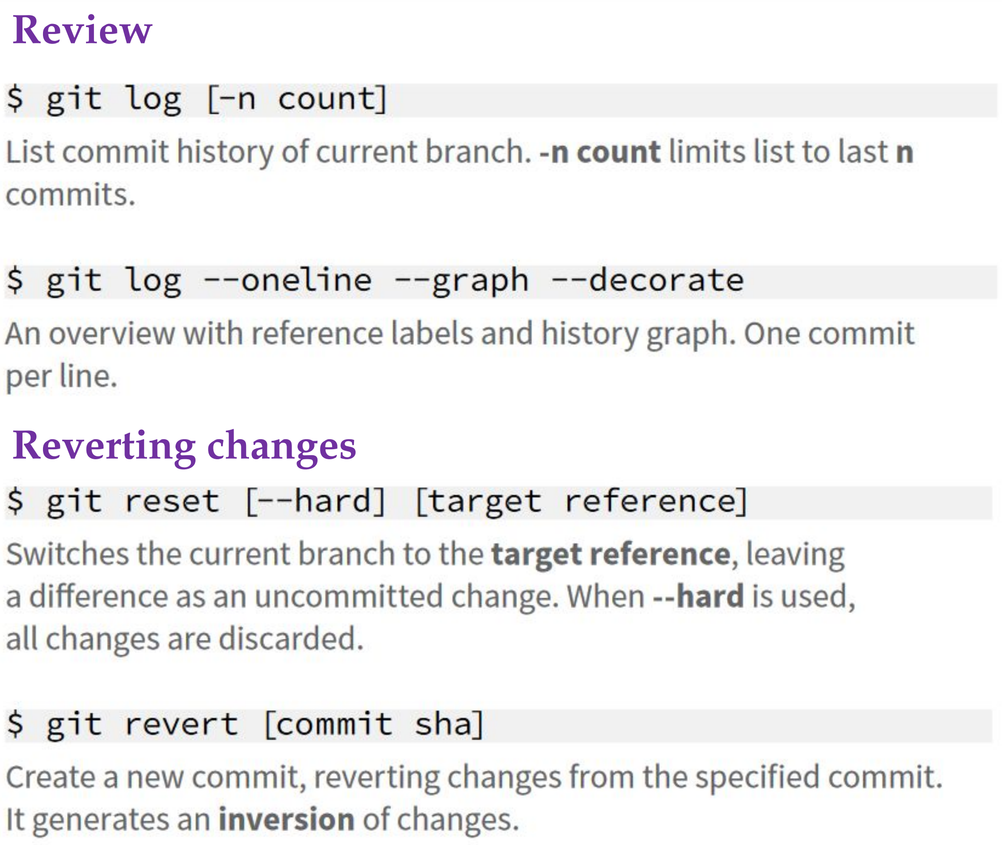
Switch **working directory** to the specified branch. With **-b**: Git will create the specified branch if it does not exist.

```
$ git merge [from name]
```

Join specified **[from name]** branch into your current branch (the one you are on currently).

```
$ git branch -d [name]
```

Remove selected branch, if it is already merged into any other. **-D** instead of **-d** forces deletion.



Review

```
$ git log [-n count]
```

List commit history of current branch. **-n count** limits list to last **n** commits.

```
$ git log --oneline --graph --decorate
```

An overview with reference labels and history graph. One commit per line.

Reverting changes

```
$ git reset [--hard] [target reference]
```

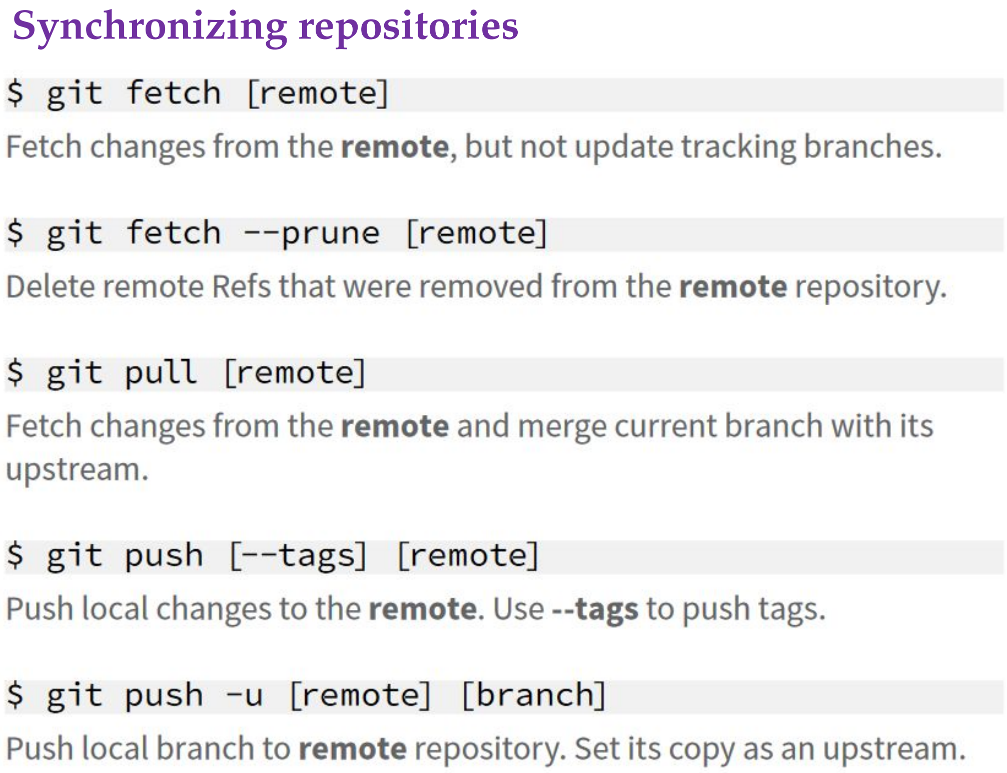
Switches the current branch to the **target reference**, leaving a difference as an uncommitted change. When **--hard** is used, all changes are discarded.

```
$ git revert [commit sha]
```

Create a new commit, reverting changes from the specified commit. It generates an **inversion** of changes.

Gitlab Cheat Sheet

Git Commands



Synchronizing repositories

```
$ git fetch [remote]
```

Fetch changes from the **remote**, but not update tracking branches.

```
$ git fetch --prune [remote]
```

Delete remote Refs that were removed from the **remote** repository.

```
$ git pull [remote]
```

Fetch changes from the **remote** and merge current branch with its upstream.

```
$ git push [--tags] [remote]
```

Push local changes to the **remote**. Use **--tags** to push tags.

```
$ git push -u [remote] [branch]
```

Push local branch to **remote** repository. Set its copy as an upstream.

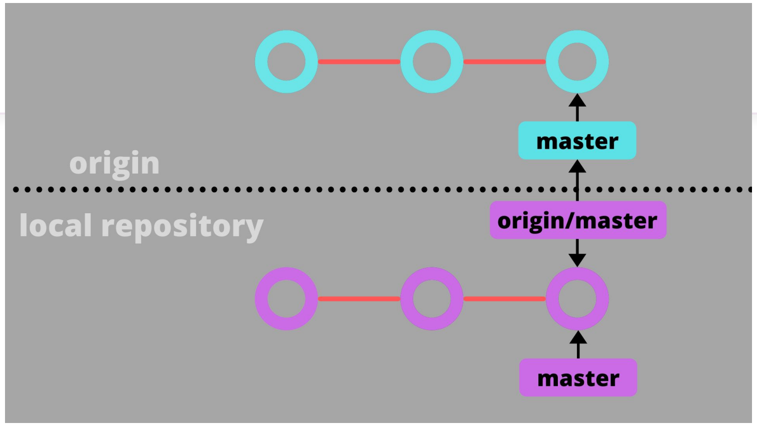
- fetch doesn't merge the changes into the local repository. Just store them in .git folder.
- You can see how the new commits are going to affect the local repository and then merge them.
- When you have multiple branch, sometimes git pull would be problematic on conflicts.

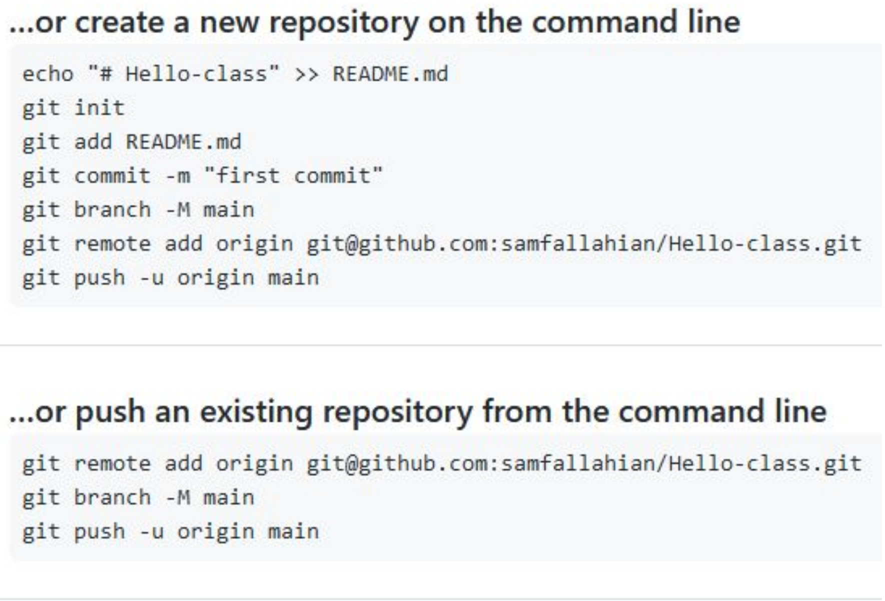
It is better to use git fetch and then git merge instead of git pull.

- 'git fetch' is used to retrieve changes from a remote repository but does not automatically integrate those changes into your working branch.
- 'git pull' is a combination of two commands: git fetch followed by git merge or git rebase. It not only downloads changes from the remote repository but also automatically integrates them into your current working branch..

Gitlab Cheat Sheet

Git Commands



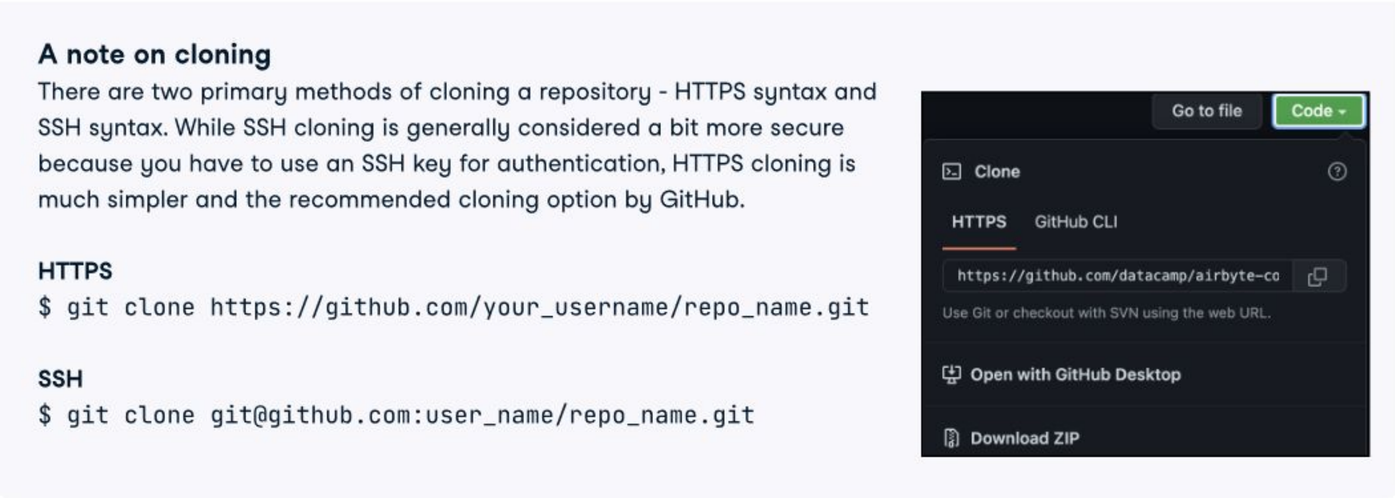


...or create a new repository on the command line

```
echo "# Hello-class" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:samfallahian/Hello-class.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:samfallahian/Hello-class.git
git branch -M main
git push -u origin main
```



A note on cloning

There are two primary methods of cloning a repository - HTTPS syntax and SSH syntax. While SSH cloning is generally considered a bit more secure because you have to use an SSH key for authentication, HTTPS cloning is much simpler and the recommended cloning option by GitHub.

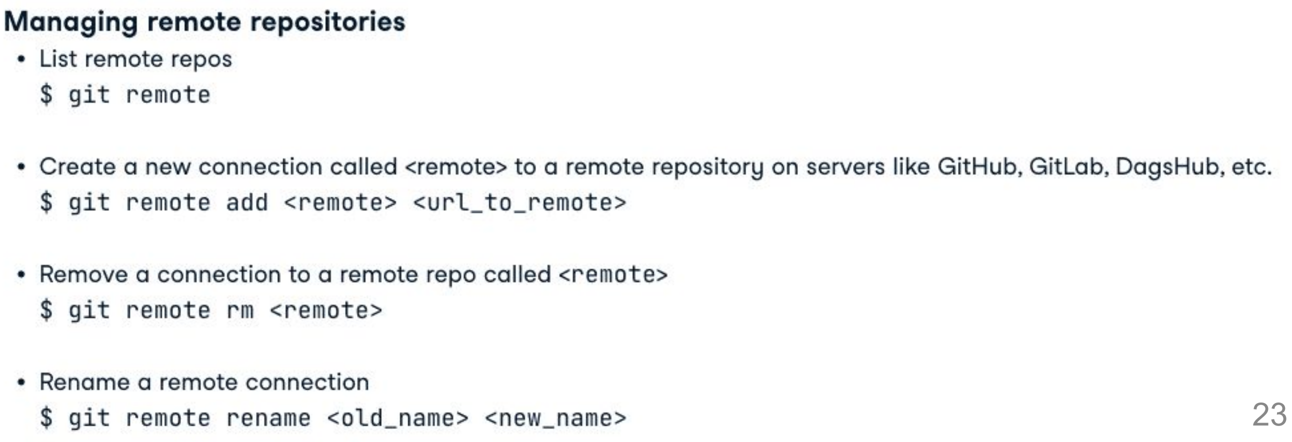
HTTPS

```
$ git clone https://github.com/your_username/repo_name.git
```

SSH

```
$ git clone git@github.com:user_name/repo_name.git
```

Go to file
Code
Clone
HTTPS
GitHub CLI
https://github.com/datacamp/airbyte-co
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP



Managing remote repositories

- List remote repos
  `$ git remote`
- Create a new connection called <remote> to a remote repository on servers like GitHub, GitLab, DagsHub, etc.
  `$ git remote add <remote> <url_to_remote>`
- Remove a connection to a remote repo called <remote>
  `$ git remote rm <remote>`
- Rename a remote connection
  `$ git remote rename <old_name> <new_name>`

Link

Conventional Commits

The Conventional Commits specification is a lightweight convention on top of commit messages. It provides an easy set of rules for creating an explicit commit history; which makes it easier to write automated tools on top of.

- Automatically generating CHANGELOGs.
- Automatically determining a semantic version bump (based on the types of commits landed).
- Communicating the nature of changes to teammates, the public, and other stakeholders.
- Triggering build and publish processes.
- Making it easier for people to contribute to your projects, by allowing them to explore a more structured commit history.

Link

Let’s get started!