Collaborative Git

CS445 Modern Asynchronous Programming

Maharishi University of Management

Department of Computer Science

Teaching Faculty: Assistant Professor Umur Inan

Prepared by: Assistant Professor Asaad Saad

Version Control System / Source Code Manager

A Version Control System (**VCS**) is a tool that manages different versions of source code.

A Source Code Manager (**SCM**) is another name for a version control system.

Git is an implementation to SCM (VCS)

History Revision Features

Label a change

Give a detailed explanation of why a change was made

Move between different versions of the same document

Undo change A, make edit B, then get back change A without affecting edit B

Branch

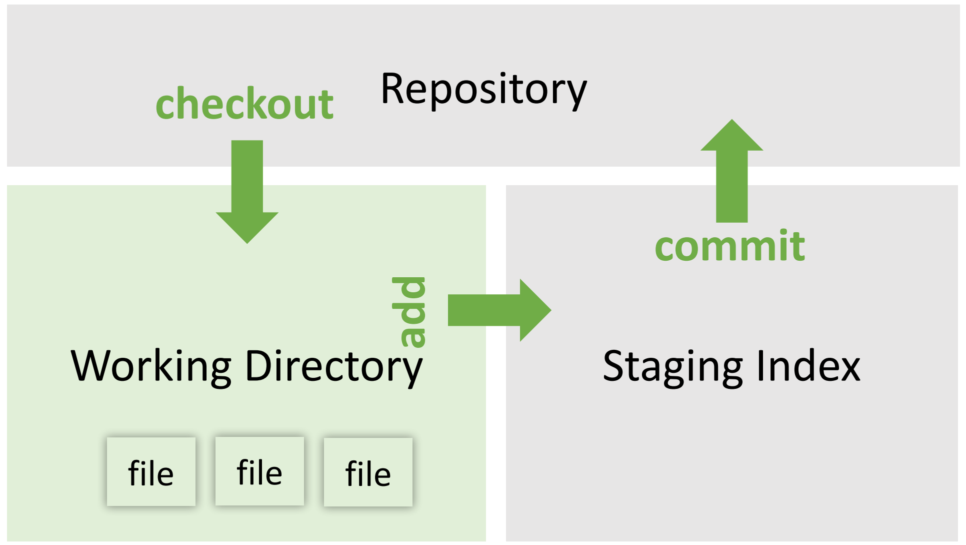
A branch is when a new line of development is created that splits the main line of development. This alternative line of development can continue without altering the main line.

By default, any repo will have one **Master** branch

The special **HEAD** pointer has a reference to currently **Active Branch**

Git Structure

- **Repository (repo)**
- **Working Directory**
- **Staging Area (Staging Index)**



Commit: When content of Staging Area is moved to the Repo. Each commit has a unique ID hash **SHA** (Secure Hash Algorithm).
Checkout: When content in the repository is copied into the Working Directory.

SHA will look like: `e2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6`

`git init`

Create a new, empty repository in the current directory.

It sets up all the necessary files and directories that Git will use to keep track of everything. All these files are stored in a directory called **`.git`**

This **`.git`** directory is the repo, it is where Git records all of the commits and keeps track of everything.

`git status`

Display the current status of the repository

- Tell us about new files that have been created in the **Working Directory** that Git hasn't started tracking, yet
- Files that Git is tracking in the **Staging area** that have been modified
- The current Git **Branch** in your local repository

`git diff`

To see changes that have been made to files in the **staging area** but haven't been committed yet.

`.gitignore`

To keep files/directories in your working directory from being staged and committed to the repo.

`git add`

Move files from the Working Directory to the Staging Index.

The period . can be used in place of a list of files to tell Git to add the current directory (and all nested files)

`git commit -m "message"`

Takes files from the Staging Index and saves them in the repository

The goal is that each commit has a single focus. Each commit should record a single-unit change.

A commit should explain **what** the commit does (not how or why).

Do not use the word "and", if you have to use "and", your commit message is probably doing too many changes - break the changes into separate commits.

Updating The Last Commit

To **update the most-recent commit** instead of creating a new commit

```
git commit –amend -m "Your new commit message"
```

Amended commits are actually entirely new commits and the previous commit will no longer be on your current branch.

`git log`

Displays all the **commits of a repository**

`git log --oneline` lists one commit per line

`git log --stat` to see what files were modified and how many lines of code were added or removed.

`git log --patch` (**`-p`**) to display the actual changes made to a file.

press **Q** to quit out of the log

`git show`

Will only display the most recent commit.

To display details about a certain commit:

```
git show <SHA> = git log -p <SHA>
```

`git tag -a <tagname>`

Add a marker on a specific commit. The tag does not move around as new commits are added.

`git tag` will display all tags that are in the repository.
`git tag -a <tagname> -m <message> <SHA>`
`git tag -d <tagname>` will delete a tag

Annotated tags include a lot of extra information such as the person who made the tag, the date the tag was made, and a message for the tag

`git branch`

List all branch names in the repository, A branch is used to do development or make a fix to the project that won't affect the project.

Once you make the change on the branch, you can combine that branch into the master branch (merging).

`git branch <branchname>` will create a new branch
`git branch -d <branchname>` will delete a branch

You can't delete a branch that you're currently on. So to delete a branch, you'd have to switch to either the master branch or create and switch to a new branch.

`git checkout <branchname>`

Switch between branches (sets **HEAD** to *`<branchname>`* or sets the **Active Branch**)

- Remove all files and directories from the Working Directory that Git is tracking
- Files that Git tracks are stored in the repository, so nothing is lost
- Go into the repository and pull out all of the files and directories of the commit that the branch points to

`git merge <name-of-branch-to-merge-in>`

To combine Git branches, Git will **combine** the lines of code that were changed on the separate branches together and **make a commit** to record the merge on the current checked-out active branch.

Fast-forward Merge
When the branch being merged in is ahead of the checked-out branch. The checked-out branch's pointer will just be moved forward to point to the same commit as the other branch.

When we combine two divergent branches, a commit is going to be made and a default message will be supplied.

Merge Conflict

A merge conflict will happen when the exact same line is changed in separate branches.

`<<<< HEAD`: code in current checked-out active branch.

`|||| merged common ancestors`: original code.

`==== until >>>> <branchname>`: code in the branch we want to merge.

To solve a merge conflict

1. Choose which line to keep
2. Remove all lines with indicators
3. Commit Merge Conflict: save the file, add it to the staging index, and commit it.

`git revert <SHA-of-commit-to-revert>`

Reverting creates a new commit that undo a previous commit.

This command will **undo the changes** that were made by the provided commit and **create a new commit** to record the change.

Traversing Commits

The current active branch/commit is where HEAD is pointing at, and we already know that we can reference commits by their SHA, by tags, branches.
There will be times when we'll want to reference a commit relative to another commit (current commit).

The following indicate the parent commit of the current commit

```
HEAD^
HEAD~1
```

The following indicate the grandparent commit of the current commit

```
HEAD^^
HEAD~2
```

^ vs ~

Usually a merge commit has two parents, the first parent is the branch you were on when you ran **`git merge`** while the second parent is the branch that was merged in.

HEAD~1	Always indicates the First parent
HEAD^	First parent
HEAD~2	Grandparent
HEAD^2	Second parent

Traversing Example

```
* 9ec05ca (HEAD -> master) Revert "Set page heading to "MSD""
* db7e87a Set page heading to "MSD"
*   796ddb0 Merge branch 'heading-update'
|\
| * 4c9749e (heading-update) Set page heading to "MSD"
* | 0c5975a Set page heading to "MUM"
|/
*   1a56a81 Merge branch 'sidebar'
|\
| * f69811c (sidebar) Update sidebar with course name
| * e6c65a6 Add new sidebar content
* | e014d91 (footer) Add links to social media
* | 209752a Improve site heading for SEO
* | 3772ab1 Set background color for page
|/
* 5bfe5e7 Add starting HTML structure
* 6fa5f34 Add .gitignore file
* a879849 Add header to blog
* 94de470 Initial commit
```

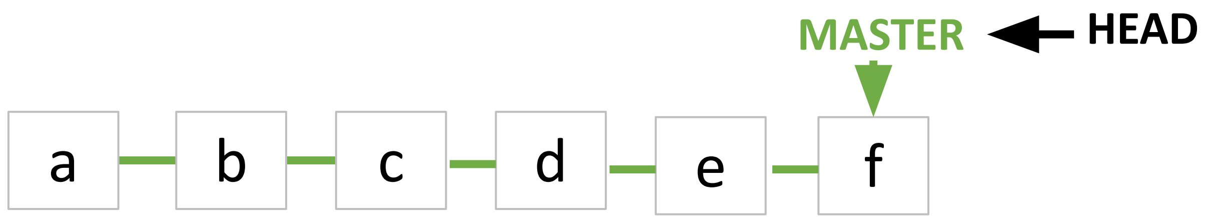
```
Which commit is
referenced by:
HEAD^
HEAD~1
HEAD^^
HEAD~2
HEAD^^^
HEAD~3
HEAD^^^2
HEAD~6
HEAD~4^2
```

`git reset <SHA-of-commit-to-reset>`

This command will erase commits from the repository.

1. Move the **HEAD** or current branch pointer to the referenced commit
2. Erase or Stage or Unstage commits

`git reset` Flags



`git reset [--mixed] HEAD~1` Moves $\boxed{f}$ to the Working Directory

`git reset --soft HEAD~1` Moves $\boxed{f}$ to the Staging Index

`git reset --hard HEAD~1` Moves $\boxed{f}$ to the Trash

⚠️Backup and Restore

The **`git reset`** command will **erase** commits from the current branch.

It is recommended that before we do any resetting, to create a **backup** branch on the most recent commit so we can get back to the commits if we make a mistake by merging the backup branch into master.

Remote Repository

Git is a distributed version control system which means there is not one main repository of information. Each developer has a copy of the repository.

A remote repository is the same Git local repository but it exists somewhere else.

Github is a hosting service for version control repositories can be managed from your browser. This is where we will host our remote repositories to be shared with other developers.

You can add as many remote repositories as you want.

Starting October 1, 2020 all ***master*** *branches will be called* ***main*** *branches to avoid any unnecessary references to slavery.*

Star a Repository (Bookmark)

Starring (**Bookmark**-*ing*) is helpful if you want to keep track of certain repositories.

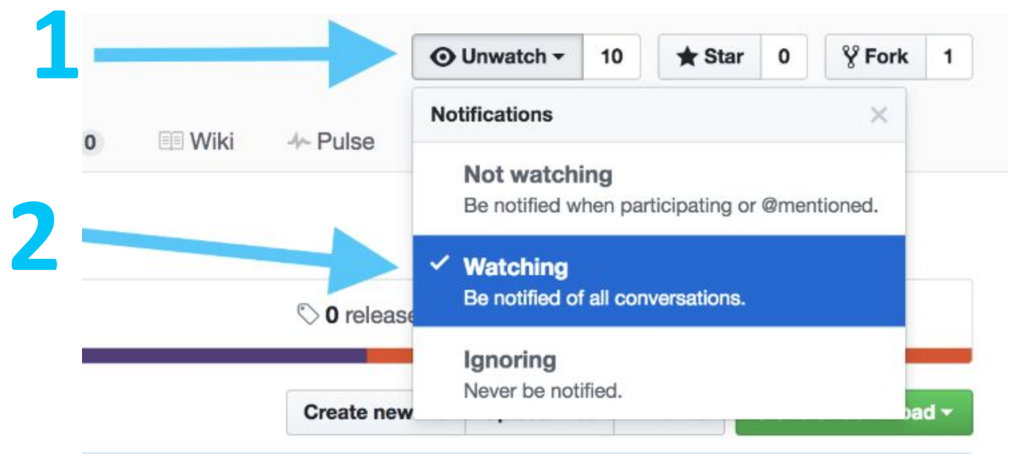
You have to manually go to the stars page to view the repositories and see if they've changed. You can go to `https://github.com/stars` to list out the repositories that you have starred.

Stars usually measure a repo's **popularity**.

Watching a Repository

To keep up with a project's changes and to be **notified** of when things change.

GitHub will notify you whenever anything happens with the repository like people pushing changes to the repository, new issues being created, or comments being added to existing issues.



Notifications

Not watching
Be notified when participating or @mentioned.

✓ Watching
Be notified of all conversations.

Ignoring
Never be notified.

Issues

An issue can just be any change that needs to be made to the project. Each issue:

- Have a label(s)
- Can be assigned to an individual
- Can be assigned a milestone (to be resolved by the next major release)
- Have its own comments, so a conversation can form around the issue.

You may subscribe to an issue so you'll be notified of new comments and code changes

You can communicate back and forth with a project maintainer on a specific change

`README.md`

A **README** is a text file that introduces and explains a project. It contains information that is commonly required to understand what the project is about.

This is where someone who is new to your project will start out.

While READMEs can be written in any text file format, the most common format is **Markdown**.

Code hosting services such as GitHub will also look for your README and display it along with the list of files and directories in your project.

Remote Shortname

A shortname is just a short and easy way to refer to the location of the remote repository.

A shortname is local to the current repository.

The word "**`origin`**" is the default name that's used to refer to the main remote repository.

```
// displays all remotes
git remote

// link the local repo to a remote with a shortname "origin"
git remote add origin https://github.com/asaadsaad/msd

// displays both the shortname and the URL
git remote -v
```

`git clone <path-to-repository-to-clone>`

Create an identical copy of an existing repository:

- Takes the path to an existing repository
- By default will create a directory with the same name as the repository that's being cloned (can be given a second argument that will be used as the name of the directory)
- Create the new repository inside of the current working directory

⚠️ If someone clones someone else's repo locally, they can only make changes to their own local copy, they won't have permission to make changes to the remote project on GitHub.

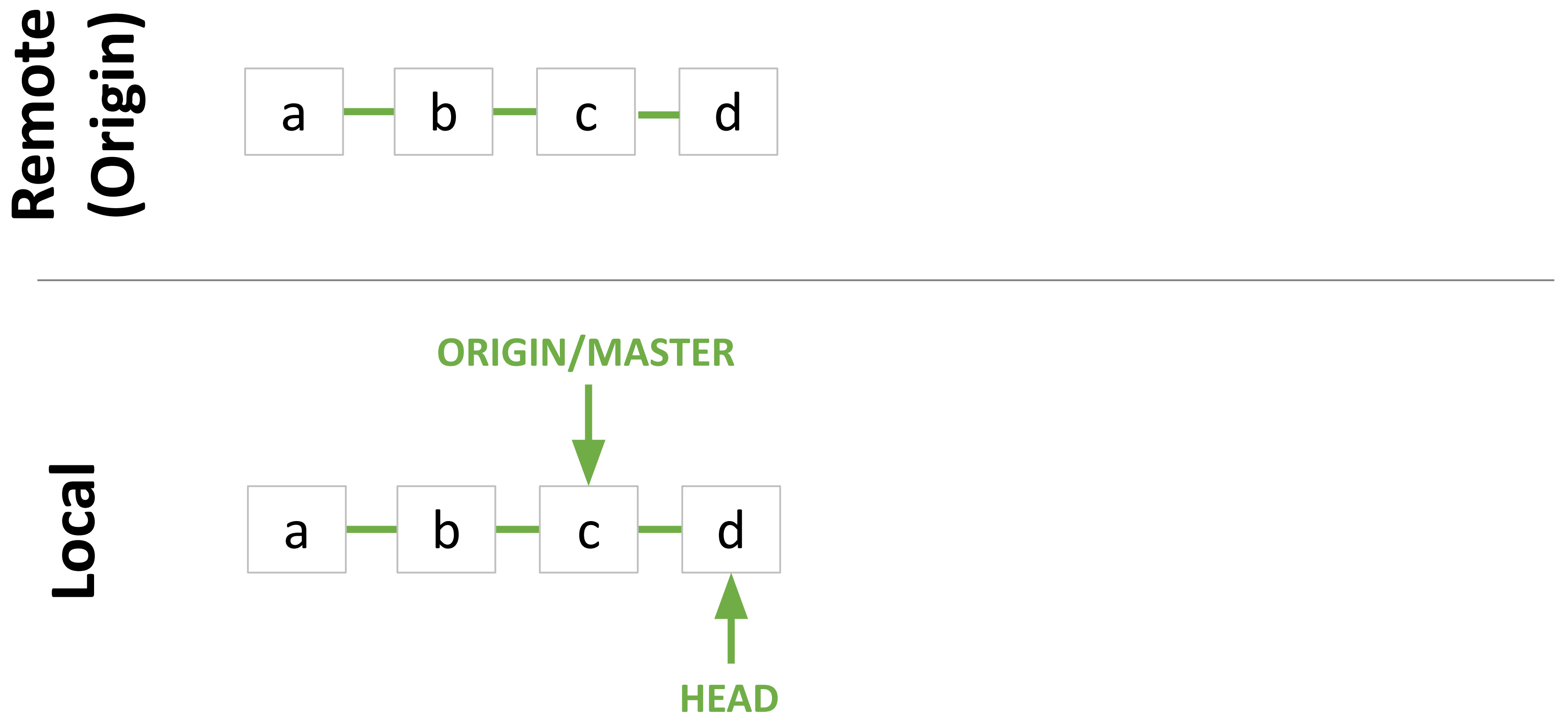
`git push <remote-shortname> <branch>`

The **`git push`** command is used to send commits from a local repository to a remote repository.

A new marker **`origin/master`** is called a **tracking branch**.

The tracking branch is telling us that the remote origin has a master branch that points to a certain commit (and includes all of the commits before it).

git push origin master



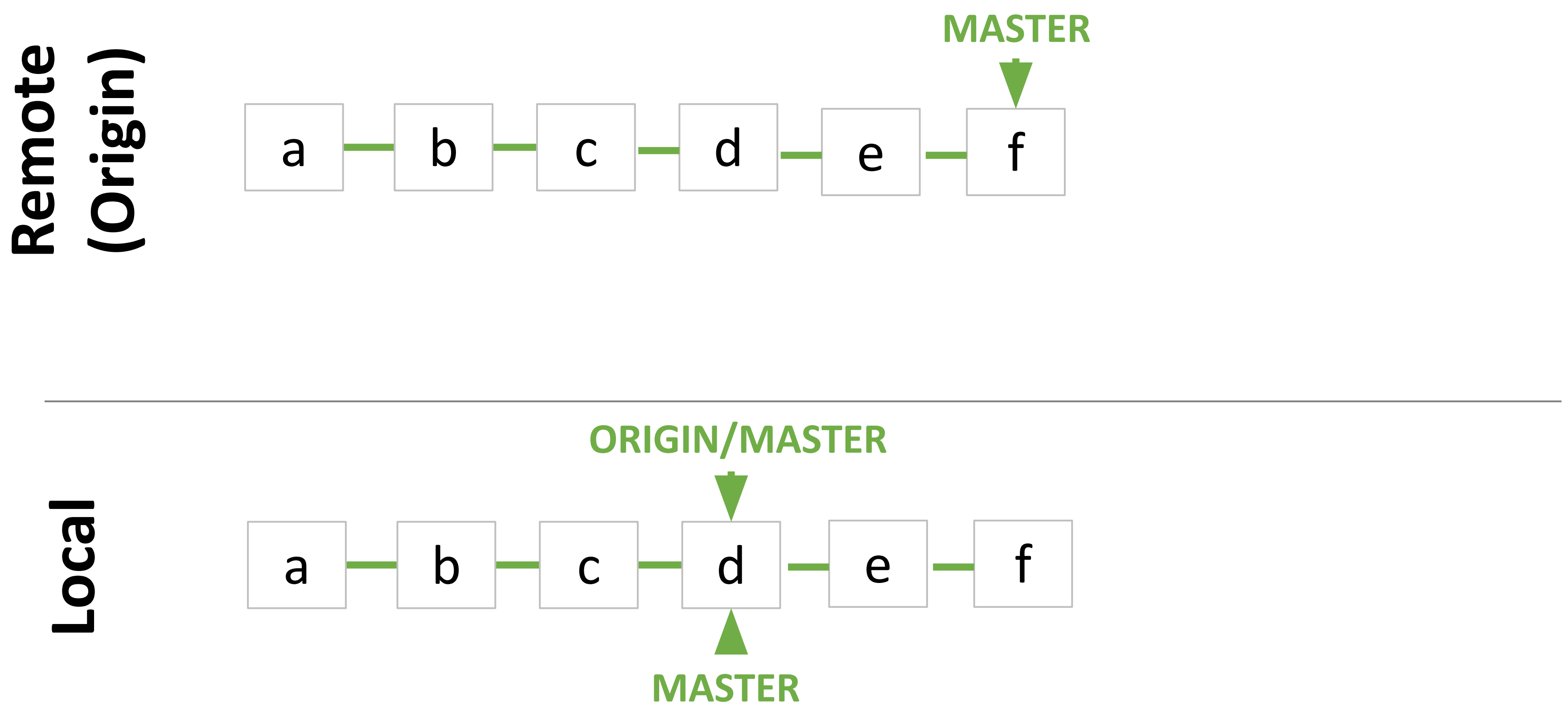
`git pull <remote-shortname> <branch>`

If there are changes in a remote repository that we would like to include in your local repository, then we want to pull in those changes.

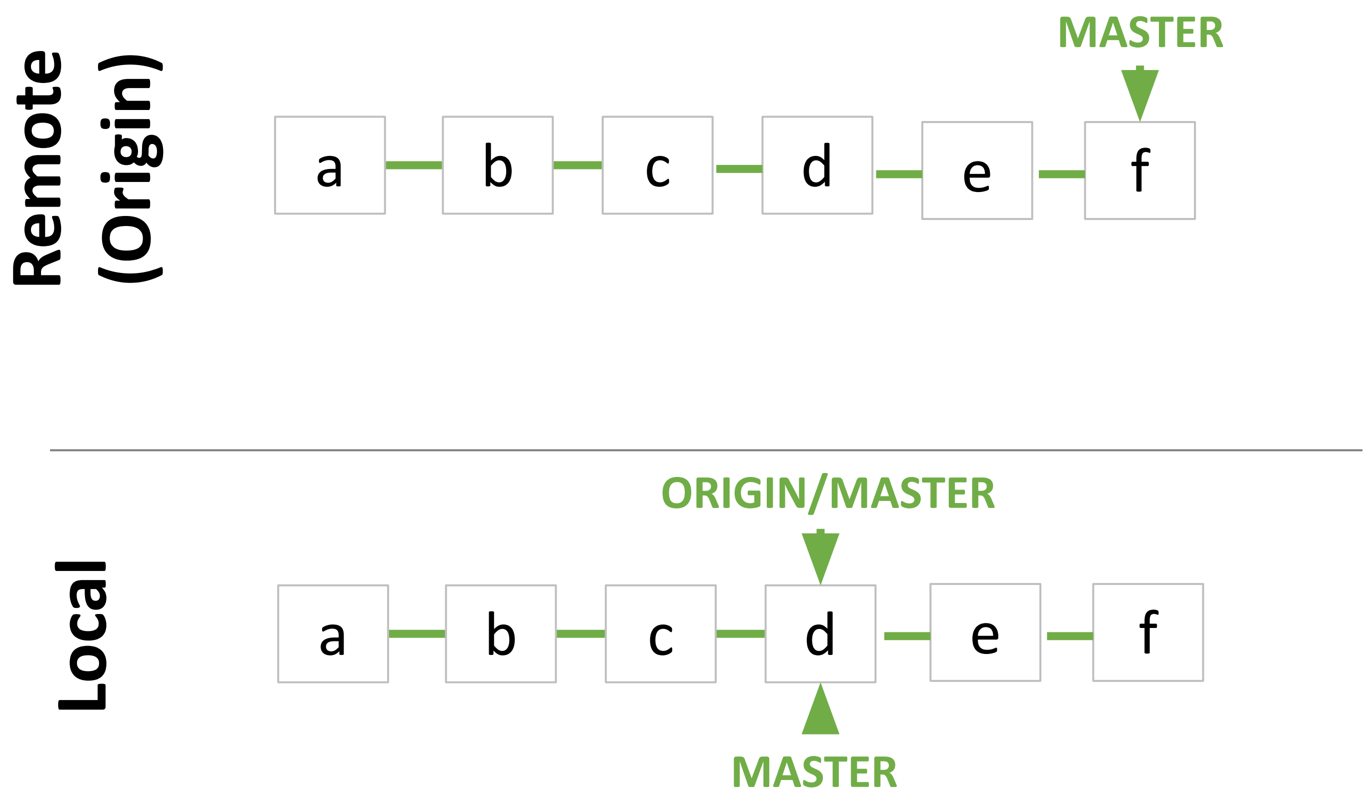
- Adds the commits from the remote branch to the local repository
- Moves the tracking branch
- The local tracking branch (origin/master) is merged into the local branch (master)

⚠️ Although, changes can be manually added on GitHub, it is not recommended, so don't do it.

git pull origin master



`git fetch <remote-shortname> <branch>`



The **`git fetch`** command just retrieves the commits from the remote and moves the tracking branch. It does not merge the local branch with the tracking branch.

```
1. git fetch origin master

2. git merge origin/master
```

Forking A Repository

When you **clone** a repository, you get an identical copy of the repository on your local machine. You may not push your changes to it.

When you **fork** a repository you duplicate it, a new duplicate copy of the remote repository is created. This new copy is also a remote repository, but it now belongs to you.

⚠️ Forking is not done on the command line

Working with your Fork

Because forking a repository gives you a copy of it in your account, you can clone it down to your computer, make changes to it, and then push those changes back to your forked repository.

Keep in mind that you push the changes back to your remote repository not to the original remote repository that you forked from.

Semantic Versioning

Given a version number **MAJOR.MINOR.PATCH**, increment the:

- **MAJOR** version when you make incompatible API changes.
- **MINOR** version when you add functionality in a backwards compatible manner.
- **PATCH** version when you make backwards compatible bug fixes.

Main Points

Before you start any work, make sure to look for the project **`CONTRIBUTING.md`** file.

Look at the GitHub issues for the project

- Look at the existing issues to see if one is similar to the change you want to contribute
- If necessary create a new issue
- Communicate the changes you'd like to make to the project maintainer in the issue

When you start developing, commit all of your work on a topic branch:

- Do not work on the master branch
- Make sure to give the topic branch clear, descriptive name

As a general best practice for writing commits:

- Make frequent, smaller commits
- Use clear and descriptive commit messages
- Update the **`README.md`** file, if necessary

Pull Request

A pull request is a request to the original or source repository's maintainer to include changes in their project that you made in your fork of their project. You are requesting that they pull in changes you've made.

How to Create a PR

To create a pull request:

- Fork the source repository
- Clone your fork down to your machine
- Make some commits
- Push the commits back to your fork
- Create a new pull request and choose the branch that has your new commits

Keep in Mind!

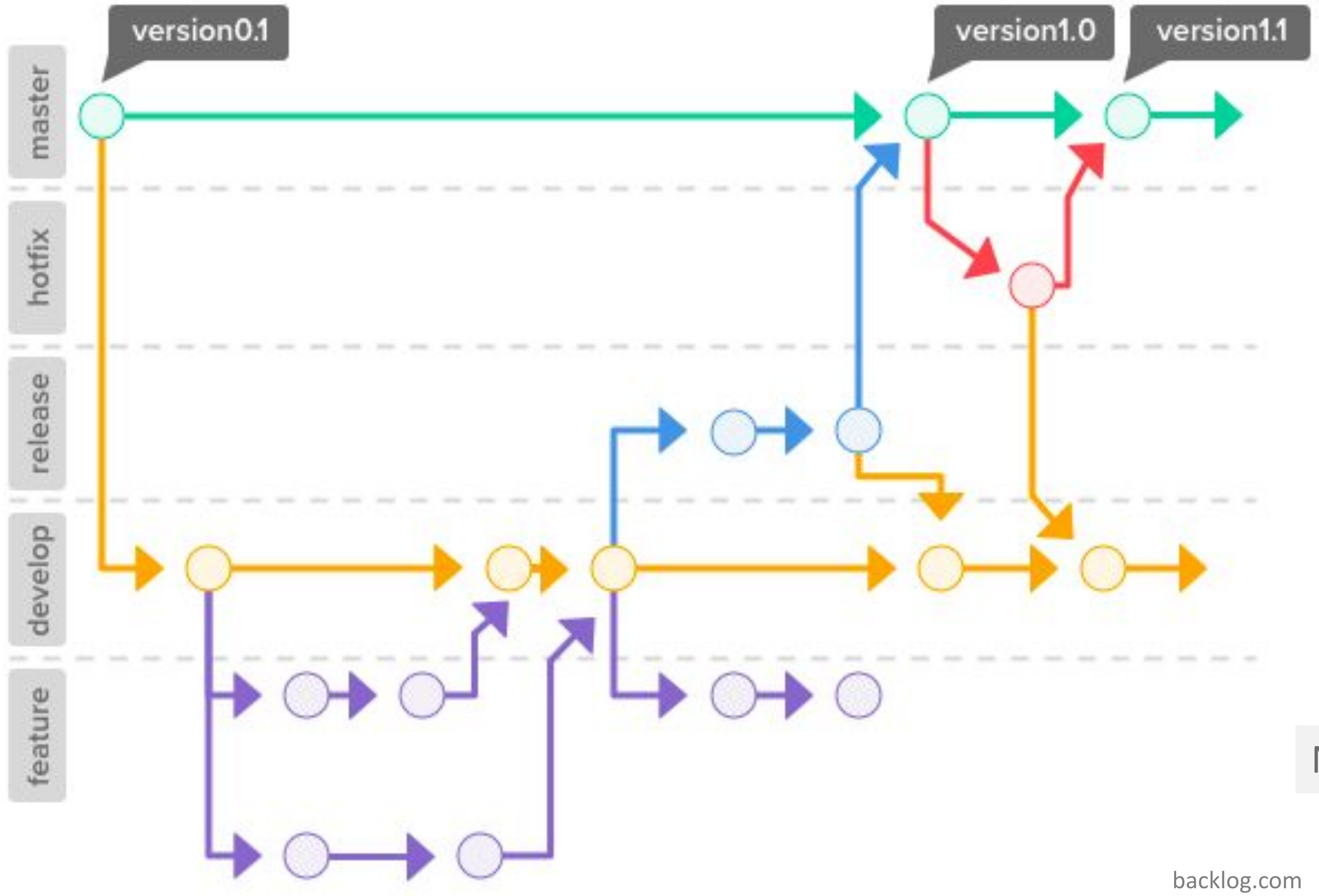
The project owner may decide not to accept your changes right away.

They might request you to make some additional changes to your code before accepting your request and merging in your changes.

Be Kind - the project's owner is a regular person just like you

Be Patient - they will respond as soon as they are able

Branching workflows



production-ready

New feature (Local)

backlog.com