



Git-GitHub



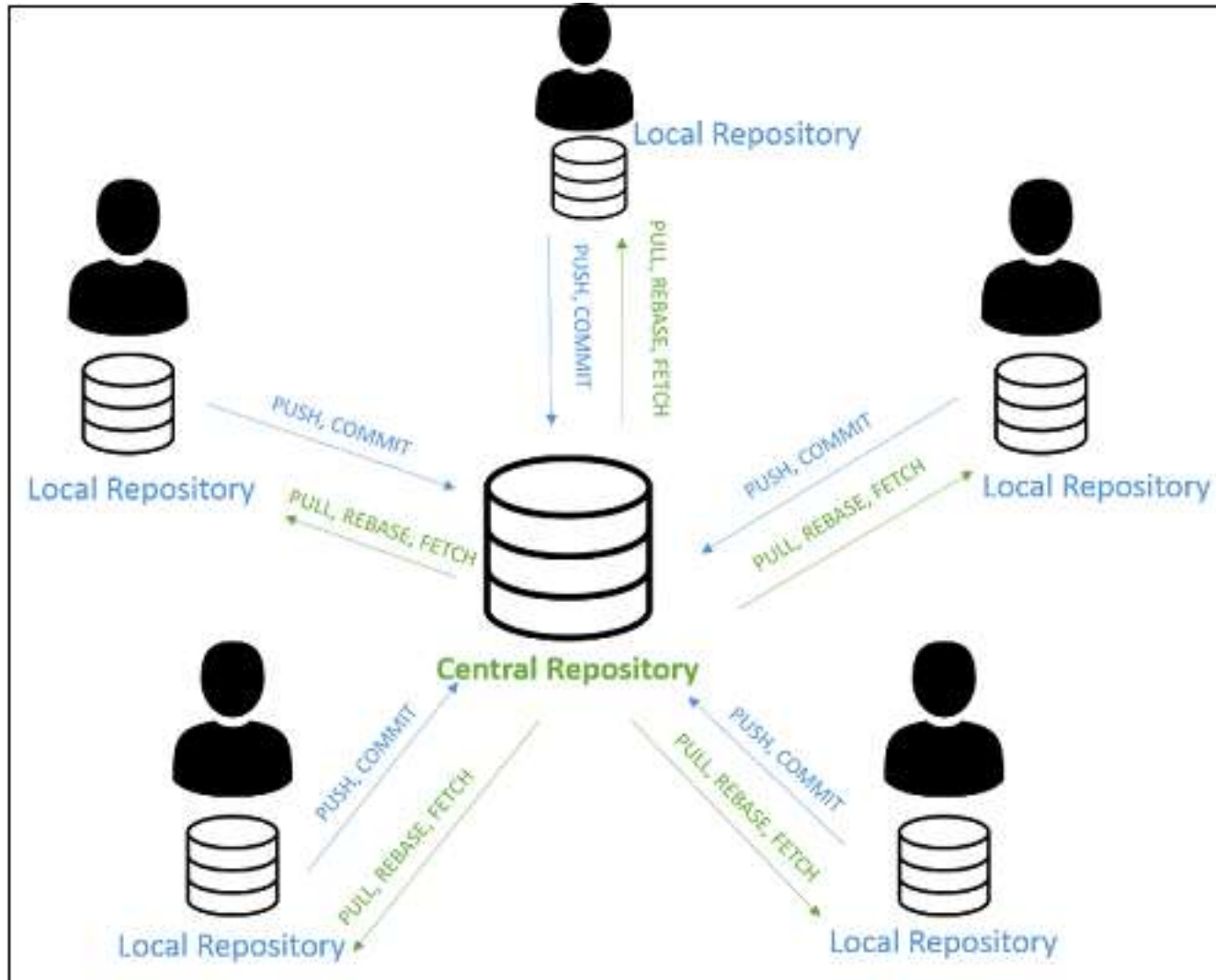
Prepared by Dhruv Rana - CloudSpikes

Overview

- Git
 - Version control system.
 - Source code management.
 - Alternate tools: SVN, StarTeam, etc.
- Github
 - Repository site.
 - Team collaboration.
 - CI/CD support.
 - Integration with Jira.

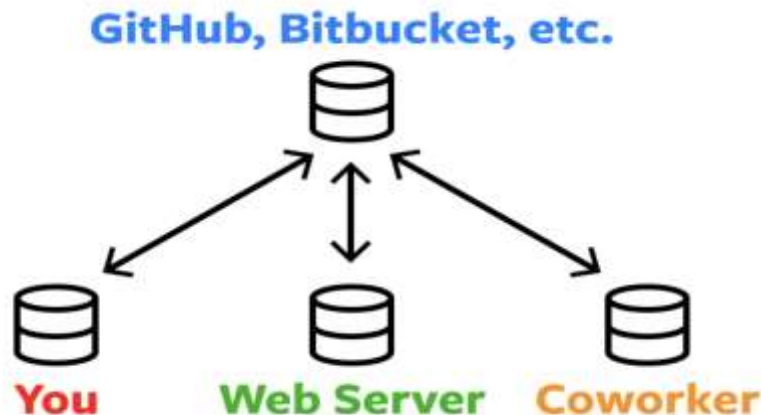


Overview



Overview

- What is Git?
 - Git is a **free and open-source distributed version control system** designed to handle everything from small to very large projects with speed and efficiency.



GIT Installation & Setup

- All OS: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
- Windows OS: <https://git-scm.com/download/win>
- Let's define ourselves first.
 - `git config --global user.email "you@example.com"`
 - `git config --global user.name "Your Name"`

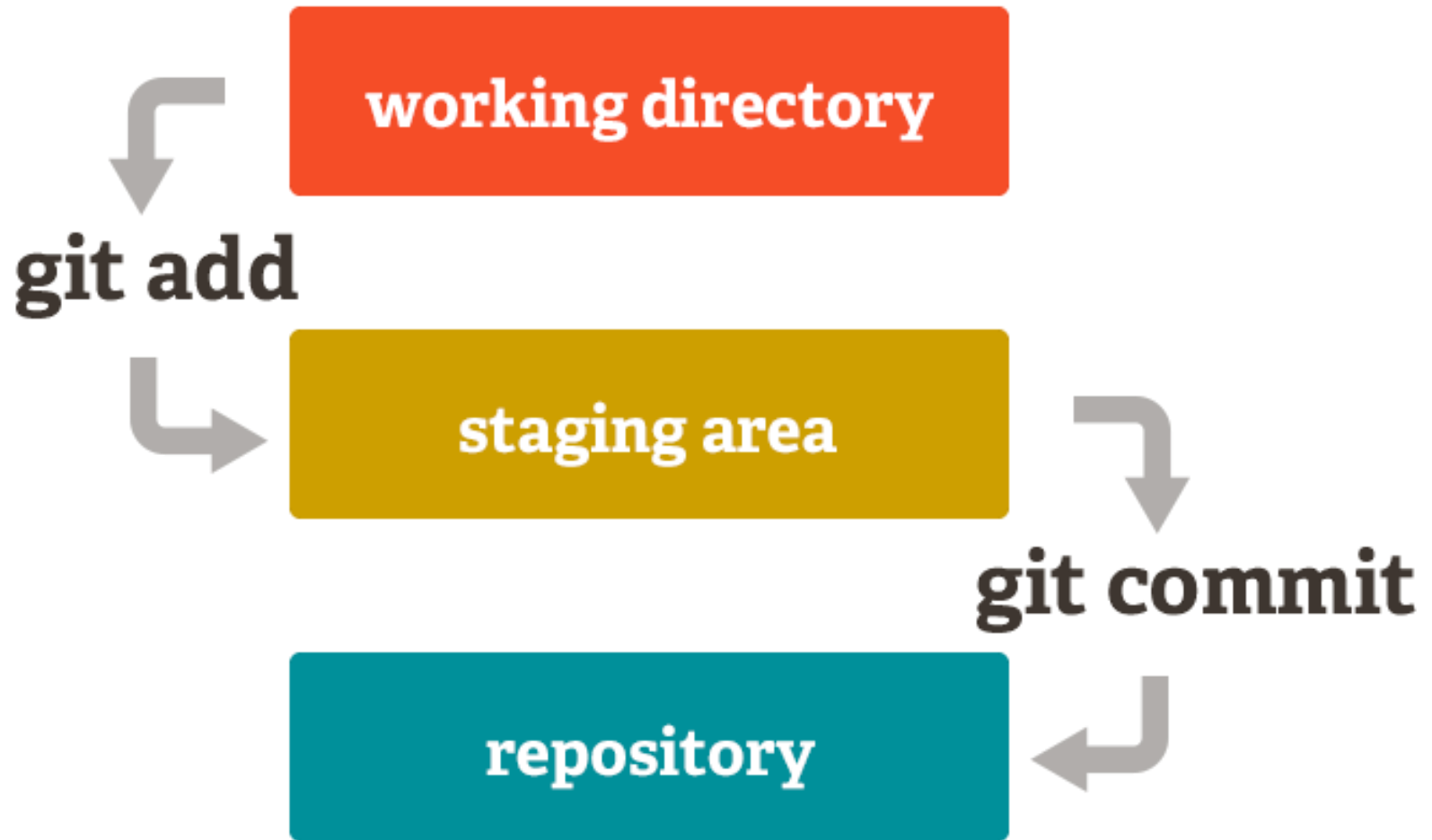
Local first

- First, let's make a git folder on our computer.
 - **git clone https/ssh-url**
- After cloning check the current branch.
 - **git branch** (list only local branch)
 - **git branch -r** (list only remote branch)
 - **git branch -a** (list only local as well as remote branch)

Check the status & add

- Check the current status on local.
 - **git status**
 - This will display track & untrack files/folders
 - New files/folders are called untracked files.
 - Existing modified files/folders are known as tracked files.
- Add the files/folders you need.
 - **git add .** (Add all files/folders)
 - **git add files/folders** (Add only specific files/folders)

Git commit process



Check the status & commit the changes

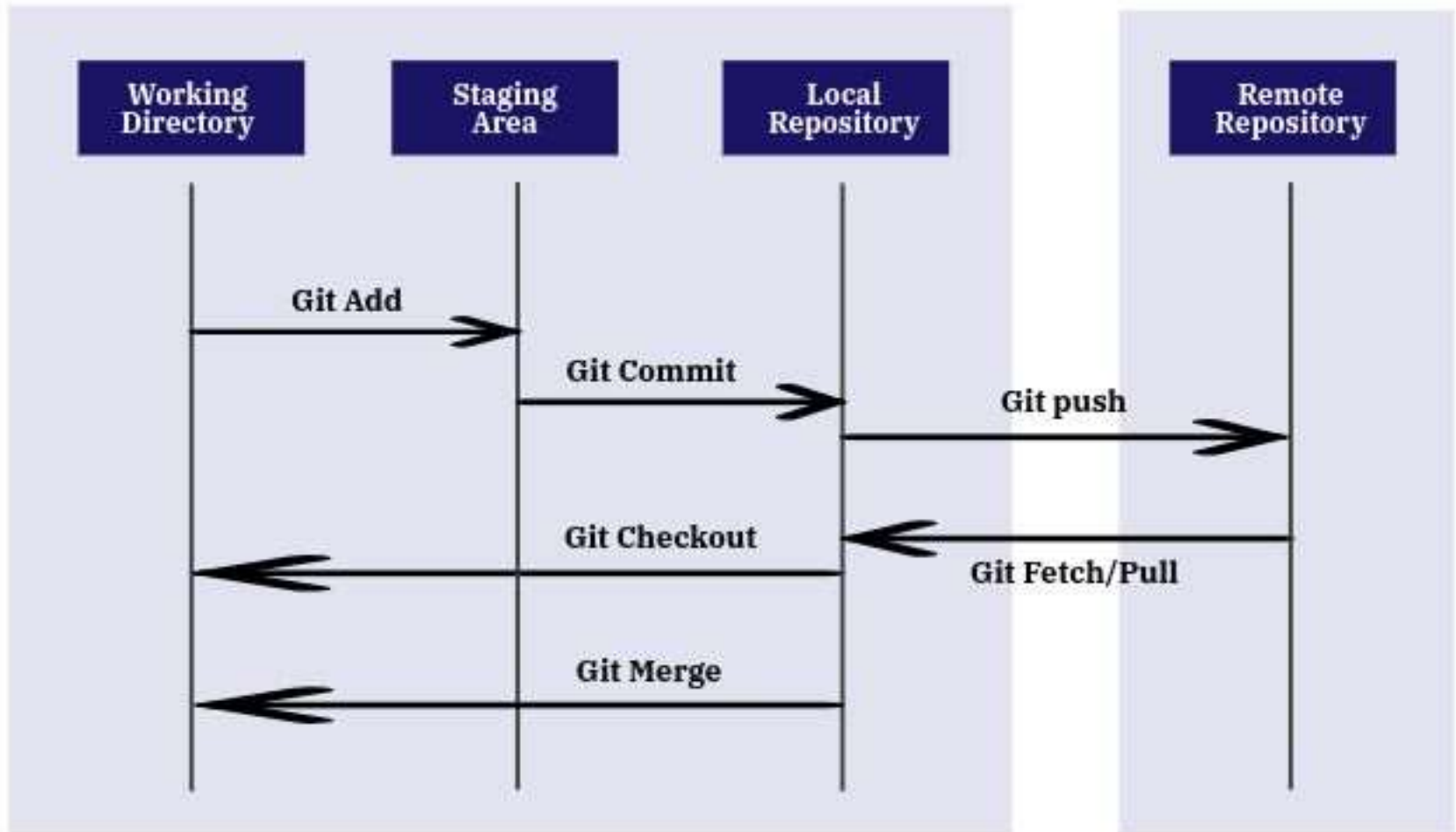
- Check the current status on local after adding.
 - **git status**
- Now, commit the changes (added files/folders)
 - **git commit -m "commit message"**
 - -m parameter is used to pass the mandatory commit message to the committed changes.

Check the commit history & push the commit/s

- Check the recent commits.
 - **git log** (Displays all the commits)
 - **git log -n 5** (Check recent 5 commit history)
- Now, push the commits to the remote repo
 - **git push**
 - **git push --set-upstream-to origin <branch>**
 - Need to set upstream branch when a new branch is forked from an existing branch.



GIT WORKFLOW



Prepared by Dhruv Rana - CloudSpikes

Practical hands-on sessions.



Prepared by Dhruv Rana - CloudSpikes

Pulling a repository/editing

- let's pull a repository from github
 - **git pull** <https://github.com/cosai/test>
- Edit the file a.txt
- Add the file by: **git add a.txt**
- Commit the added changes: **git commit -m 'something added'**
- Finally, push the commit to the remote repo: **git push**



Remove a file

- Removing a file: **git rm somefile.txt**
- Commit the change: **git commit -m 'removed'**
- Finally, push the new commit: **git push**

Temporary backup your changes

- First check your changes by: **git status**
- Then back up the tracked changes: **git stash**
- You can now check out to any other branch now: **git checkout new-branch-name**
- Come back to the previous branch: **git checkout previous-branch-name**
- You can list the stashed changes by: **git stash list**

Temporary backup your changes

- If you want to delete the listed stash changes, you can execute: **git stash drop**
- And if you want to bring back the backed-up changes: **git stash pop**

One step back!

- An easy way to revert last commit (1)
 - **git reset HEAD^**
 - **git log** (to check if the previous commit is gone)
 - **git push origin +HEAD**

“One step
back does
not mean
that you
have failed”

S.L.G.

25 Sep 2013 10:33 pm

Playing around GIT Branches

- Fork/Create a new branch from current branch:
 - **git branch** (check current branch)
 - **git checkout -b “new-branch-name”**
 - **git branch** (check the current branch as the new branch)
- Switch between branches:
 - **git branch** (check current branch)
 - **git checkout “destination-branch-name”**
 - **git branch** (check the current branch as the destination branch)

GIT Merge. Fetch vs Pull

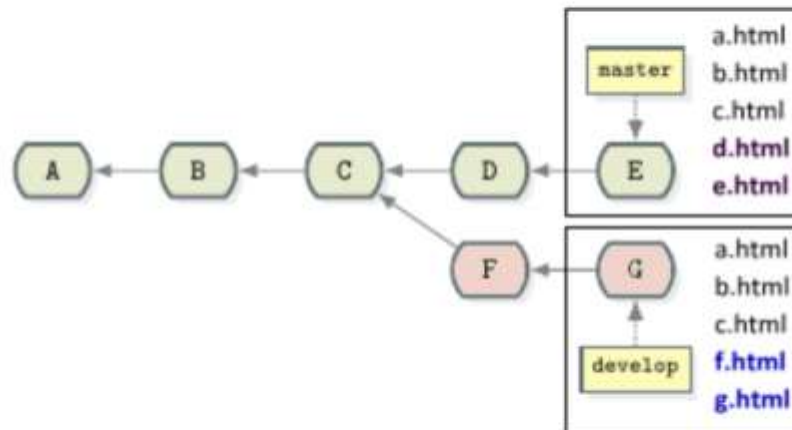
- Merge branch:
 - **git branch** (check current branch)
 - **git checkout** “main-branch-name”
 - **git merge** (previous feature branch)
- Fetch vs Pull:
 - Git fetch gives the information of a new change from a remote repository without merging into the current branch.
 - Git pull Brings a copy of all the changes from a remote repository and merges them into the current branch

GIT Squash Merge

- Say your bug fix branch is called **bugfix** and you want to merge it into the **master**:
 - **git checkout master** (Switches to your master branch)
 - **git merge --squash bugfix** (Takes all commits from the bugfix branch and groups it for a 1 commit with your current branch)
 - **git commit** (Creates a single commit from the merged changes)
 - Omitting the -m parameter in the above commit command lets you modify a draft commit message containing every message from your squashed commits before finalizing your commit.

GIT Rebase

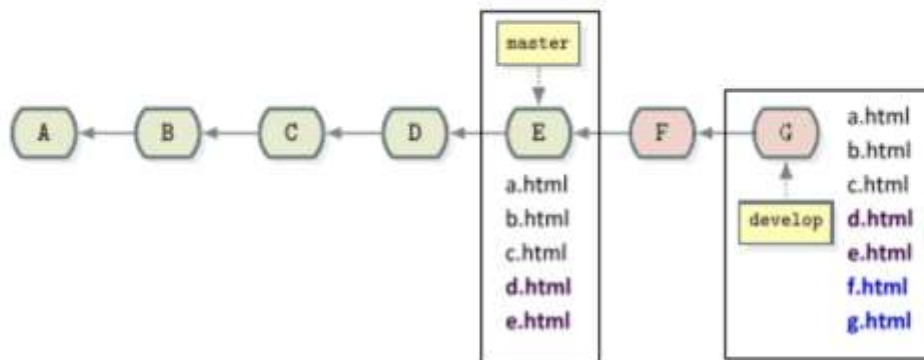
Prior to the rebase, the develop branch had only five files.



Each branch has five files before the git rebase to master operation.

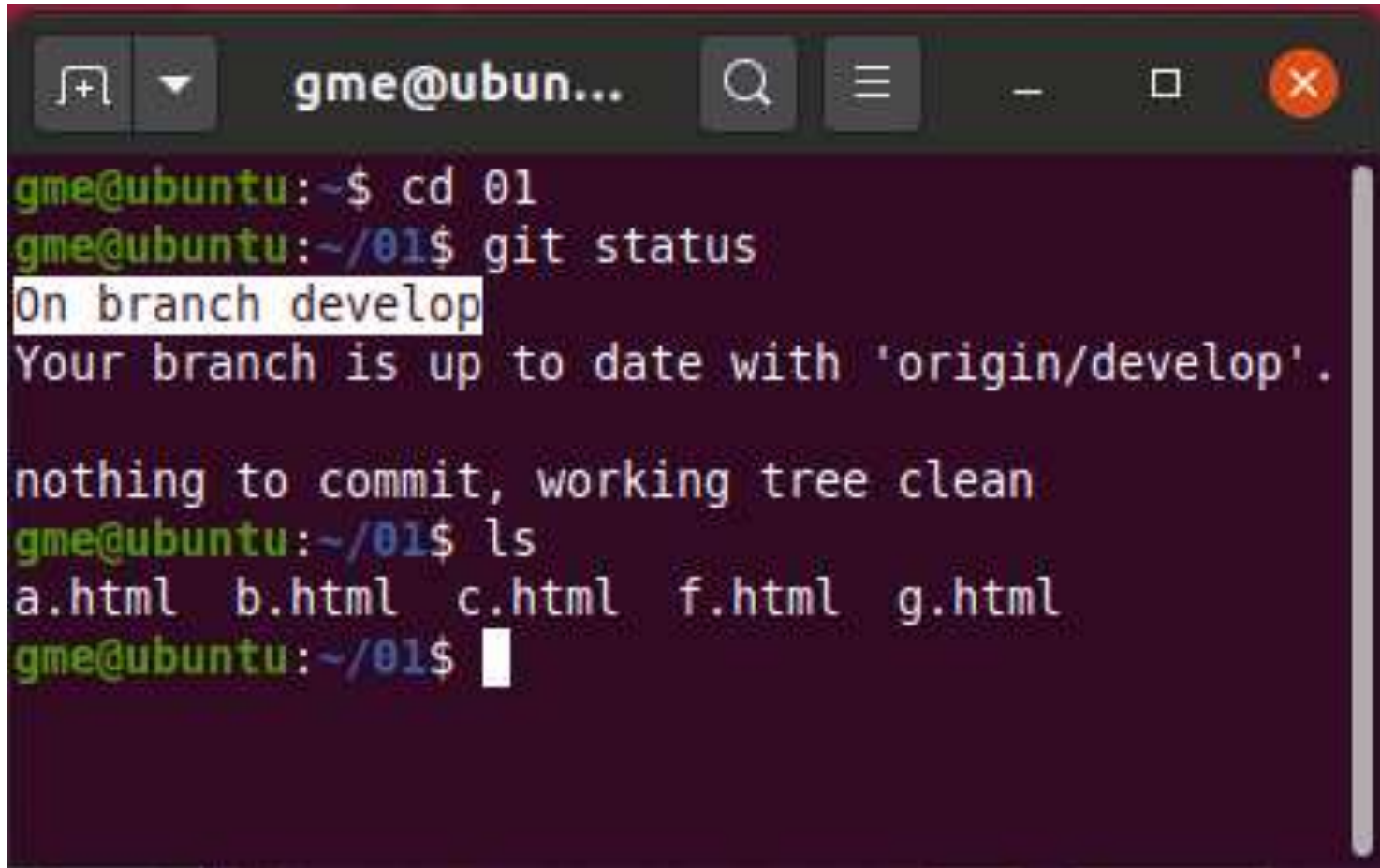
GIT Rebase

After the Git rebase to master, the develop branch has seven files. It has retained all of its original files and acquired two new files from the tip of the master branch. However, the number of files in the master branch remains unchanged.



After the git rebase of develop to master, the develop branch acquires all of master's files.

GIT Rebase

A terminal window with a dark background and light-colored text. The window title bar shows 'gme@ubun...' and standard window controls. The terminal output shows the user navigating to a directory named '01', checking the git status, and listing files. The 'On branch develop' line is highlighted with a light blue background. The window has a scrollbar on the right side.

```
gme@ubuntu:~$ cd 01
gme@ubuntu:~/01$ git status
On branch develop
Your branch is up to date with 'origin/develop'.

nothing to commit, working tree clean
gme@ubuntu:~/01$ ls
a.html  b.html  c.html  f.html  g.html
gme@ubuntu:~/01$
```

GIT Init

A new repo from scratch

Say you've just got some data from a collaborator and are about to start exploring it.

- Create a directory to contain the project.
- Go into the new directory.
- Type git init.
- Write some code.
- Type git add to add the files.
- Type git commit.

The first file to create (and add and commit) is probably a ReadMe file, either as plain text or with Markdown, describing the project.

Markdown allows you to add a bit of text markup, like hyperlinks, **bold**/*italics*, or to indicate code with a monospace font. Markdown is easily converted to HTML for viewing in a web browser, and GitHub will do this for you automatically.

GIT Init

A new repo from an existing project

Say you've got an existing project that you want to start tracking with git.

- Go into the directory containing the project.
- Type `git init`.
- Type `git add` to add all of the relevant files.
- You'll probably want to create a `.gitignore` file right away, to indicate all of the files you don't want to track. Use `git add .gitignore`, too.
- Type `git commit`.

Connect it to GitHub

You've now got a local git repository. You can use git locally, like that, if you want. But if you want the thing to have a home on GitHub, do the following.

- Go to GitHub.
- Log in to your account.
- Click the new repository button in the top-right. You'll have an option there to initialize the repository with a README file, but I don't.
- Click the "Create repository" button.

Now, follow the second set of instructions, "Push an existing repository..."

```
$ git remote add origin git@github.com:username/new_repo
```

```
$ git push -u origin master
```

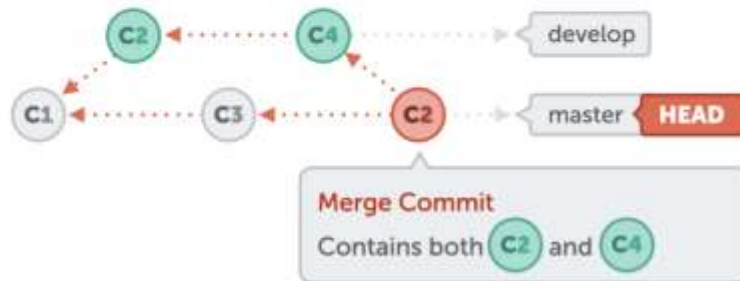
GIT Remove tracked & untracked changes

- Remove tracked files/folders:
 - **git status** (check tracked files/folders)
 - **git checkout .** (removing tracked files/folders)
 - **git status** (check tracked files/folders after removing them to confirm the removal)
- Remove untracked files/folders:
 - **git status** (check untracked files/folders)
 - **git clean -df** (removing untracked files/folders)
 - **git status** (check untracked files/folders after removing them to confirm the removal)

GIT Cherry pick

With the "cherry-pick" command, Git allows you to integrate selected, individual commits from any branch into your current HEAD branch.

Contrast this with the way commit integration *normally* works in Git: when performing a Merge or Rebase, **all commits** from one branch are integrated.



Cherry-pick, on the other hand, allows you to select **individual** commits for integration. In this example, only C2 is integrated into the master branch, but not C4.



How to use GIT Cherry pick?

In its most basic form, you only need to provide the SHA identifier of the commit you want to integrate into your current HEAD branch:

```
$ git cherry-pick af02e0b
```

This way, the specified revision will directly be committed to your currently checked-out branch. If you would like to make some further modifications, you can also instruct Git to only add the commit's **changes** to your Working Copy - without directly committing them:

```
$ git cherry-pick af02e0b --no-commit
```

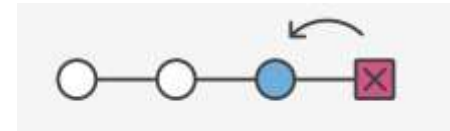
The short answer is: as rarely as possible. The reason why you should use cherry-pick rarely is that it easily creates "duplicate" commits: when you integrate a commit into your HEAD branch using cherry-pick, Git has to create a new commit with the exact same contents. It is, however, a **completely new commit object** with its own, new SHA identifier.

Whenever you can use a traditional Merge or Rebase to integrate, you should do so. Cherry-pick should be reserved for cases where this is not possible, e.g. when a Hotfix has to be created or when you want to save just one/few commits from an otherwise abandoned branch.

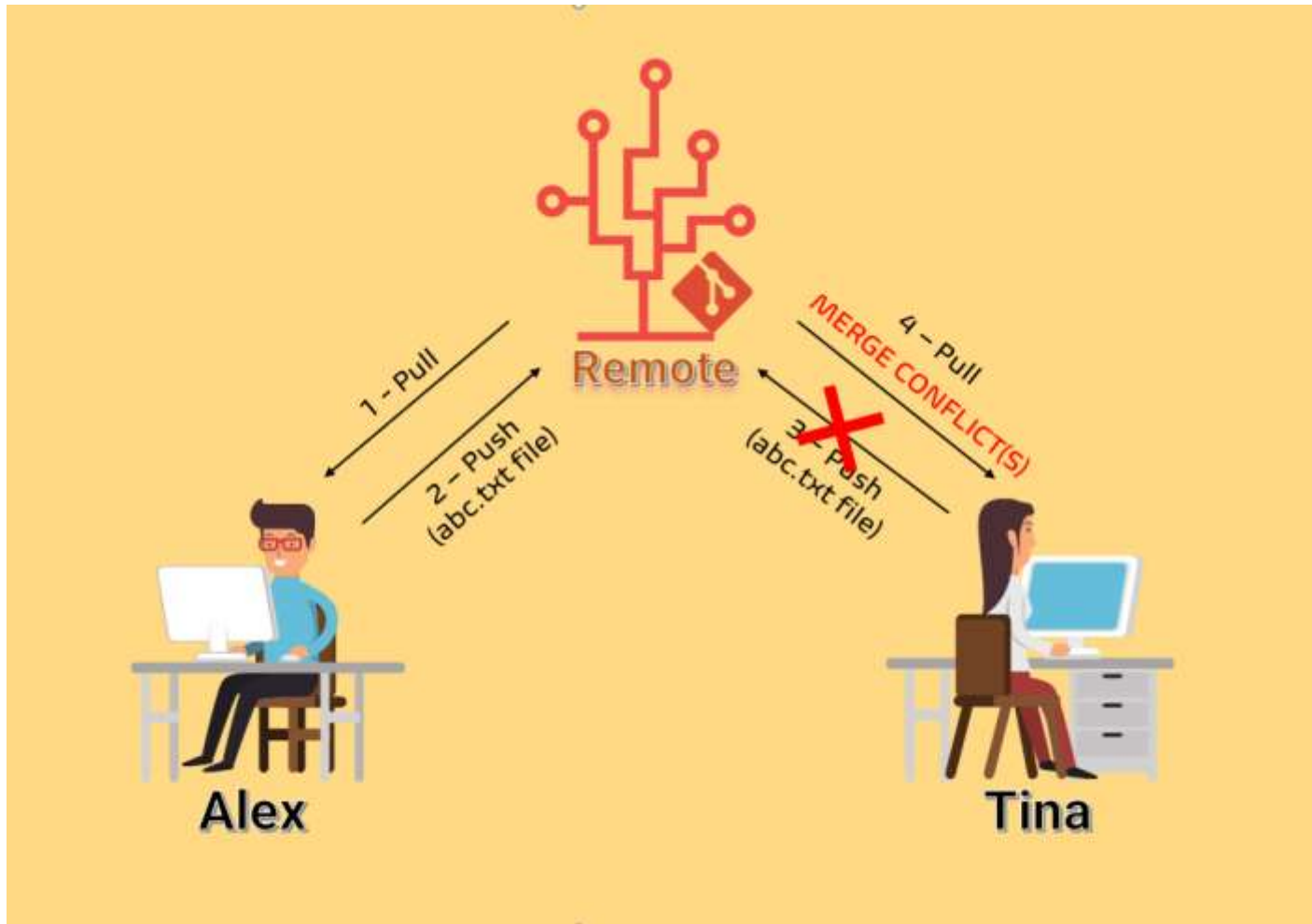
Prepared by Dhruv Rana - CloudSpikes

GIT revert & reset

- Revert commit:
 - **git log** (check commit history)
 - **git revert HEAD** (go back to the last commit)
 - **git log** (check commit history after reverting)
- Reset changes:
 - Use the git reset command to remove the added files/folders by checking the git status.
 - For e.g., **git reset file-name**



GIT Conflict Scenario



Prepared by Dhruv Rana - CloudSpikes

Steps to Resolve GIT Merge Conflicts

1. First of all, get all the new commits by: **git pull**
2. Then check for any files in status for tracked changes: **git status**
3. Once you find the tracked changes, accept incoming, remove, or keep both changes using **VS Code IDE** with ease.
4. Finally, **commit & push** the changes that will push a new commit with the resolved conflict as all the missing commit changes will be now incorporated as needed.

Any questions or queries?

- Reach out to me on Insta, LinkedIn, Email or WhatsApp:

- Insta: [@dhruv_rana_29_10](#)
- LinkedIn: [@dhruv-r-b033949b](#)
- Email: support@cloudspikes.ca
- WhatsApp: +1-647-376-7753

