A Tutorial on Git and Github

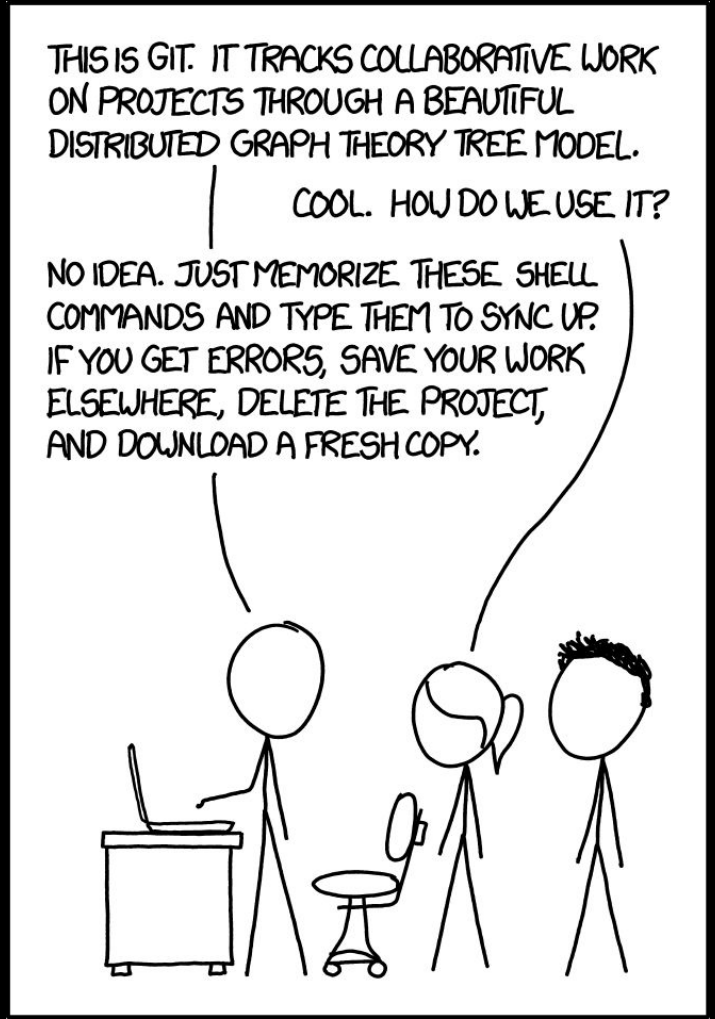
—

By Reyhane Askari and Arnaud Bergeron

Table of contents:

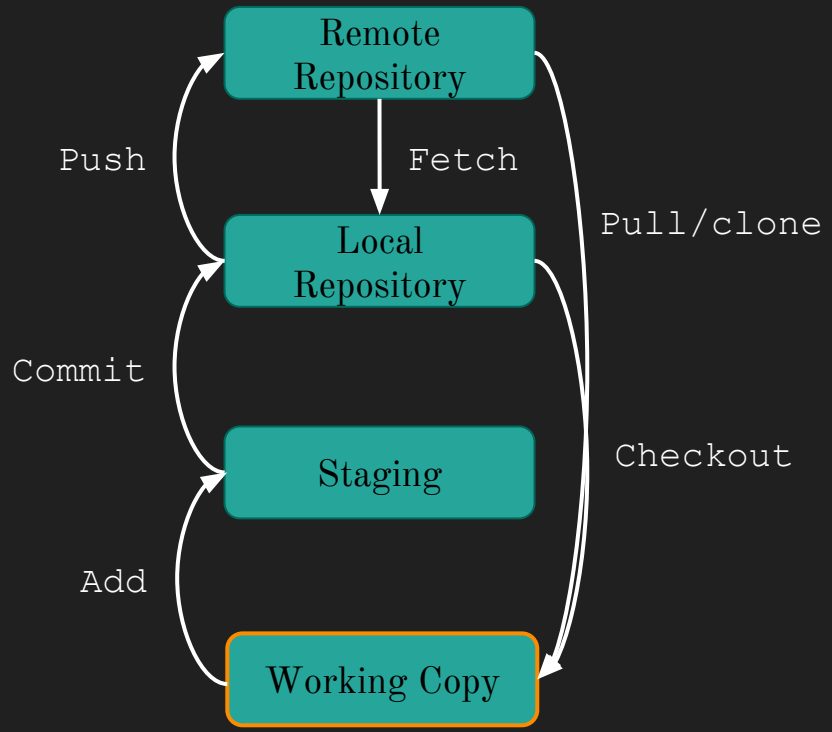
Version Control System

- Manage multiple versions of your source code.
- Great way to work as a team where you work concurrently.
- Go back and forth between the versions of your code.



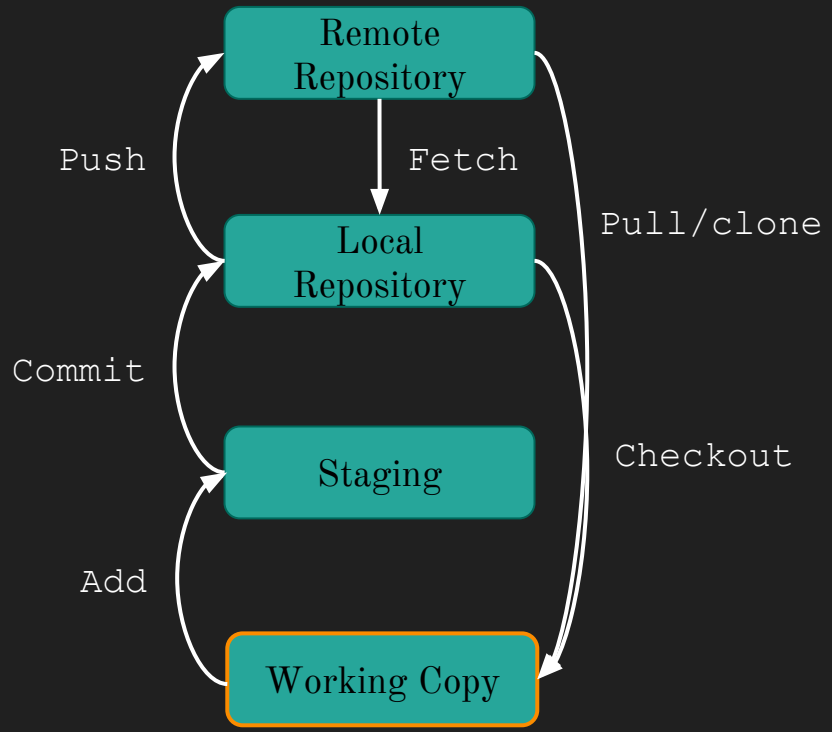
Source: https://xkcd.com/1597/

Git Basics



- `git clone /url/to/repository`
- `git config user.name <username>`
- `git config user.email <email>`
- `git add <filename>`
- `git status`
- `git diff`
- `git add <filename> -p`
- `git add .`

Git Basics

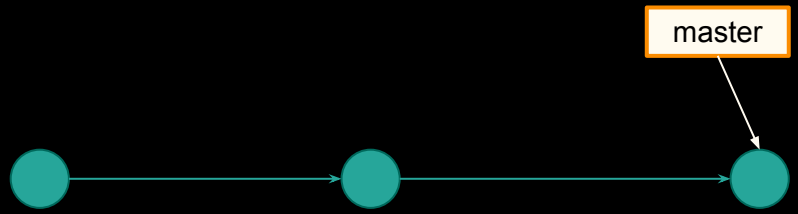


- `git commit -m "Commit message"`
- `git status`
- `git log`

- `git push`

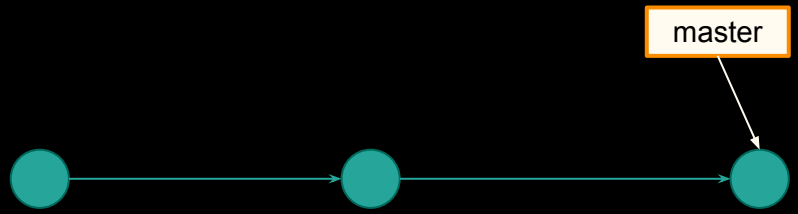
- `git pull`

Branch Management



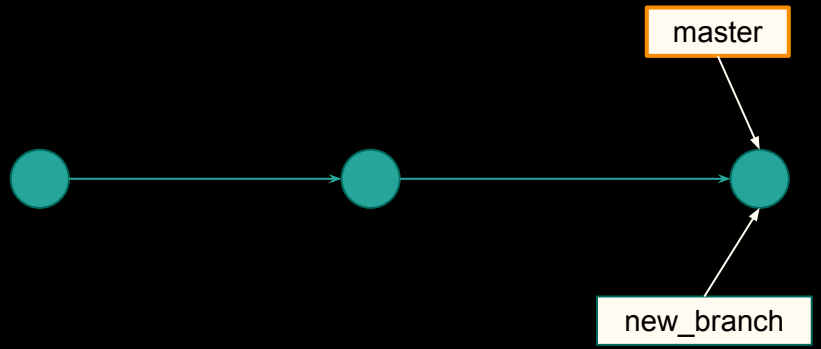
- The full project code and history is called a repository.
- A commit is a snapshot of your code at different stages of the project.
- Also, a branch is a pointer to a commit that you can change.

Branch Management



Initial State

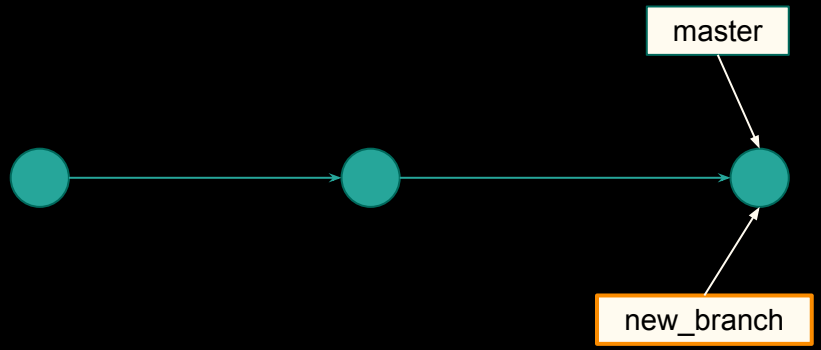
Branch Management



Create a new branch:

```
git branch new_branch
```

Branch Management



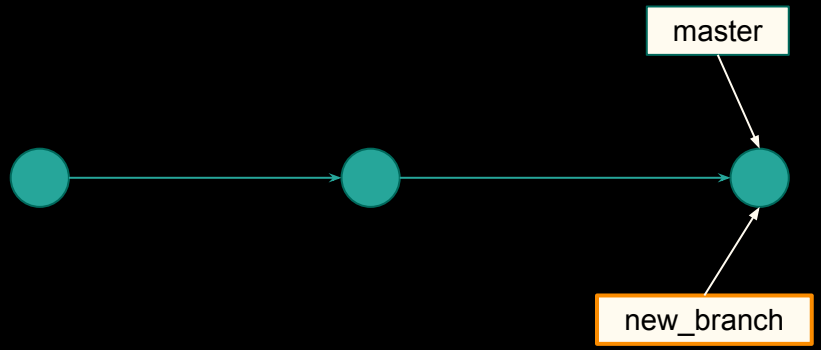
Create a new branch:

```
git branch new_branch
```

Move between branches:

```
git checkout new_branch
```

Branch Management



Create a new branch:

```
git branch new_branch
```

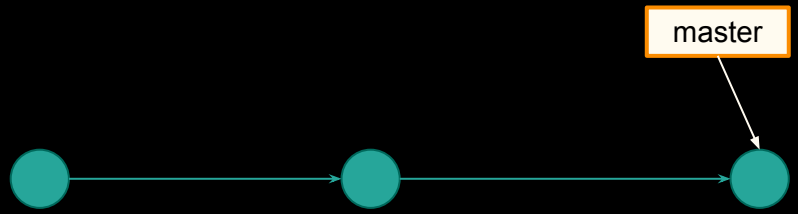
Move between branches:

```
git checkout new_branch
```

Create a new branch and check it out:

```
git checkout -b new_branch
```

Branch Management



Create a new branch:

```
git branch new_branch
```

Move between branches:

```
git checkout new_branch
```

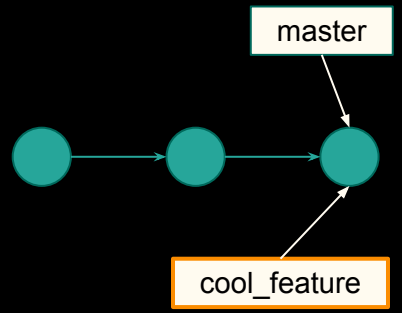
Create a new branch and check it out:

```
git checkout -b new_branch
```

Delete a branch:

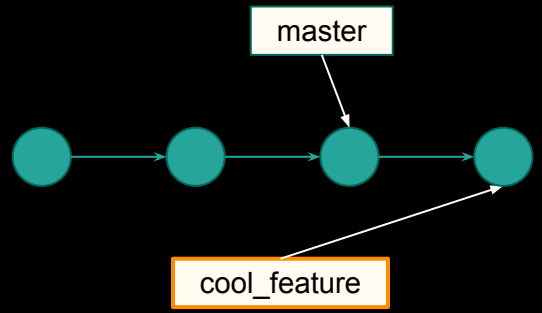
```
git branch -d new_branch
```

Branch Management




```
git checkout -b cool_feature
```

Branch Management

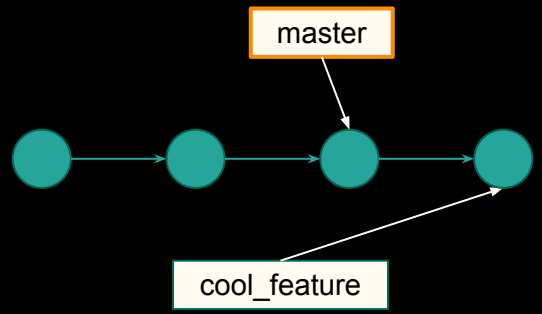



```
git checkout -b cool_feature
```

Add some code

```
git commit -m "Cool feature"
```

Branch Management



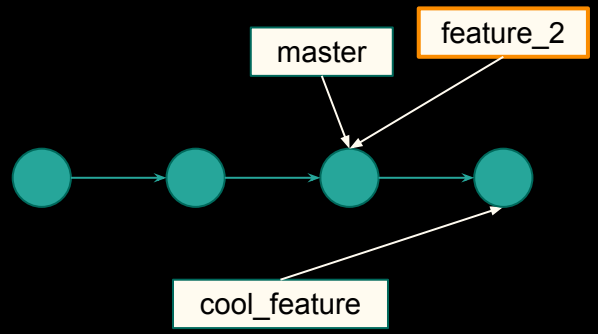

```
git checkout -b cool_feature
```

Add some code

```
git commit -m "Cool feature"
```

```
git checkout master
```

Branch Management




```
git checkout -b cool_feature
```

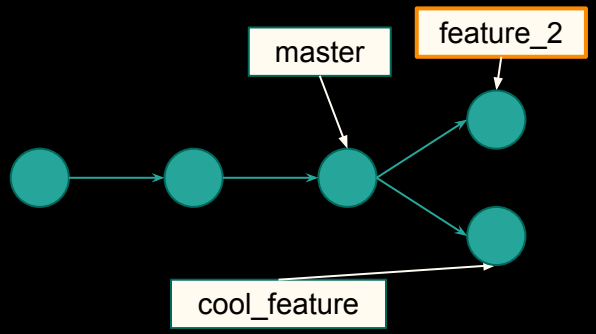
Add some code

```
git commit -m "Cool feature"
```

```
git checkout master
```

```
git checkout -b feature_2
```

Branch Management




```
git checkout -b cool_feature
```

Add some code

```
git commit -m "Cool feature"
```

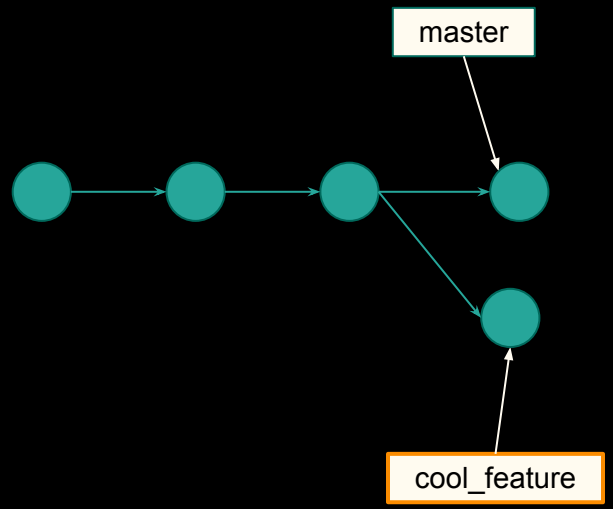
```
git checkout master
```

```
git checkout -b feature_2
```

Add some more code

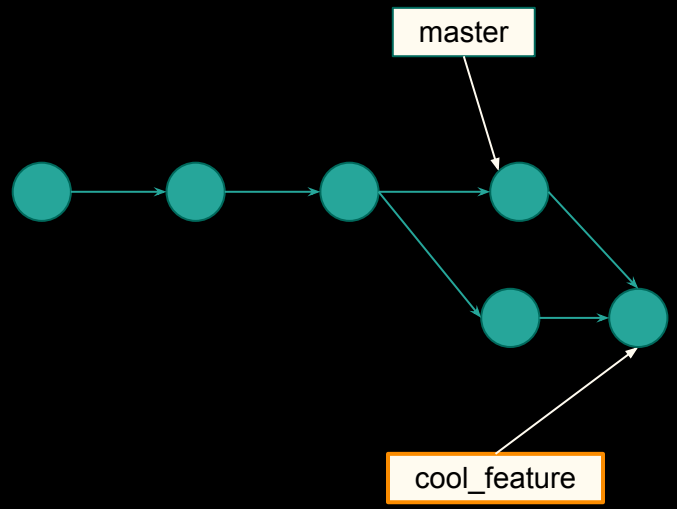
```
git commit -m "Second feature"
```

Merge



You've just made some changes but someone else also did. What to do?

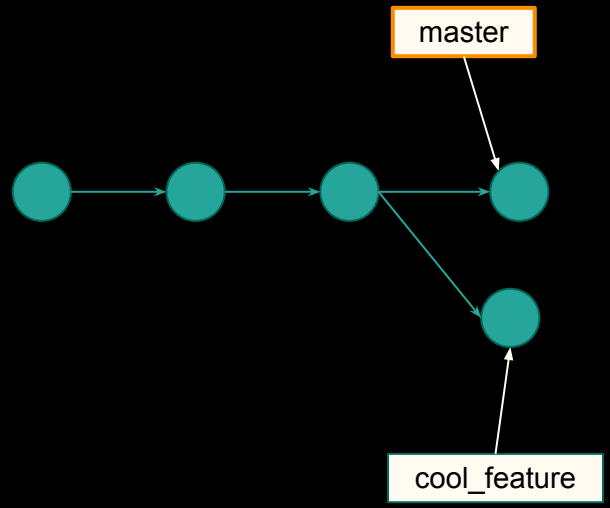
Merge



You've just made some changes but someone else also did. What to do?

```
git merge master
```

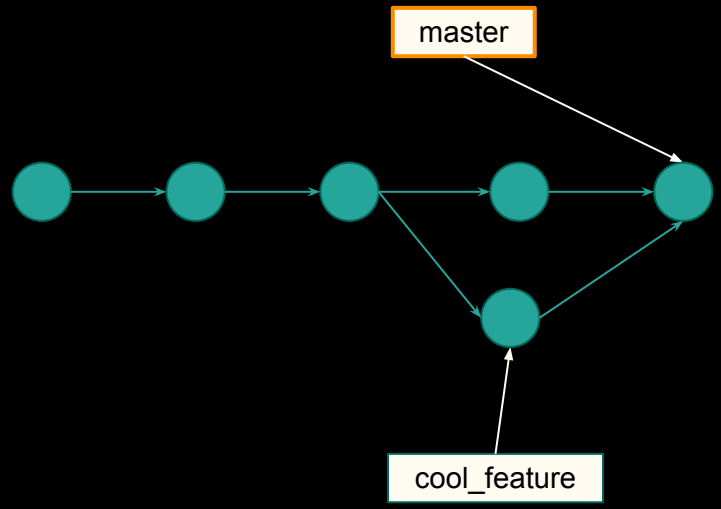
Merge



You've just made some changes but someone else also did. What to do?

```
git checkout master
```

Merge

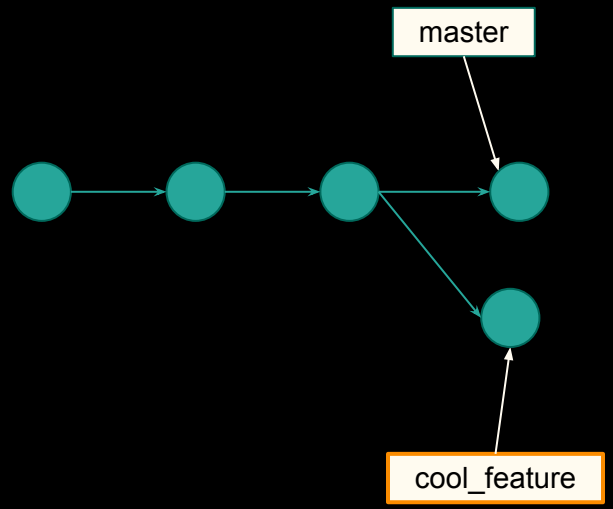


You've just made some changes but someone else also did. What to do?

```
git checkout master
```

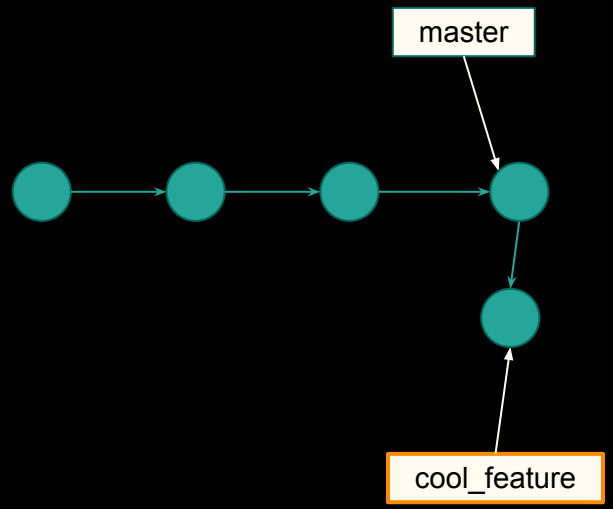
```
git merge cool_feature
```

Rebase



You've just made some changes but someone else also did. What to do?

Rebase



You've just made some changes but someone else also did. What to do?

```
git rebase master
```

How to Resolve Git Conflicts?

- Find which files have conflicts using:

```
git status
```

```
both modified:   doc/install_generic.inc
both modified:   doc/install_windows.txt
both modified:   doc/requirements.inc
```

How to Resolve Git Conflicts?

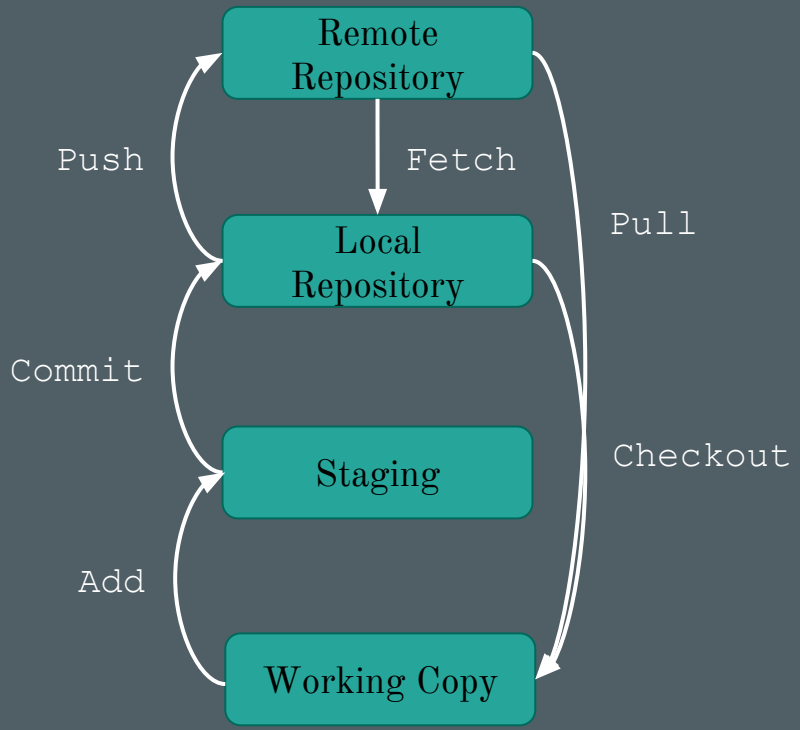
- Go to that file and find the conflict and choose the line that you want to keep.

```
<<<<<<< HEAD
        * **Recommended**: MKL, which is free through Conda with ``mkl
-service`` package.
=======
        * **Recommended**: MKL, which is free through Conda.
>>>>>>> 1b62e953a... Merge pull request #5433 from notoraptor/windoc
```

How to Resolve Git Conflicts?

- Resolve all the conflicts and add them:
  - `git add path/to/files/with/conflicts`
- Then commit your changes:
  - `git commit`
- Continue to rebase or merge:
  - `git rebase --continue`
  - `git merge --continue`
- Or abort the rebase/merge in case you want to stop applying the changes:
  - `git rebase --abort`
  - `git merge --abort`

Git Update



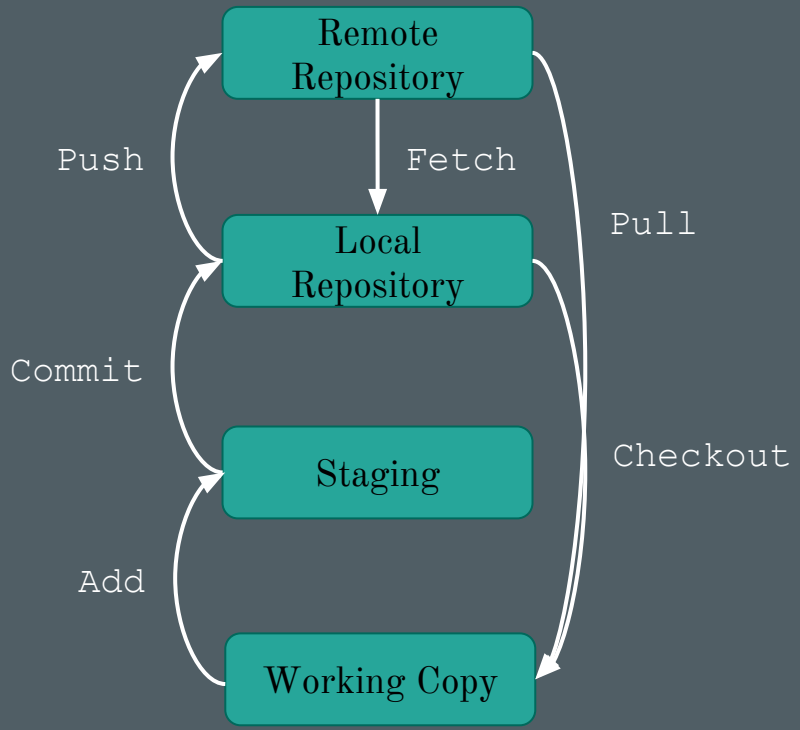
- Fetch the changes from a repository but do not merge them:

```
git fetch repo/branch
```

- Update local repository to the newest commit:

```
git pull repo/branch
```

Interaction with remote repos



- See your remote repos:

```
git remote -v
```

- Add a remote repository:

```
git remote add [name][url]
```

- Send the changes from a local repo to a remote repo:

```
git push repo/branch
```

- Push and force the local changes to the remote:

```
git push -f repo/branch
```

Workflows for contributing to other repos

- Fork the repo to contribute to
- Checkout your forked repo
- Add a remote pointing to the repo to contribute to
- Create a pull request

Git Stash

- If you have unstaged files, and you want to checkout to another branch, you need to save your changes:

```
git stash
```

- To retrieve the saved changes on that branch:

```
git stash pop
```

- You can see the current stash list:

```
git stash list
```

Some Useful Tips

- Never ever commit to the master, always create a new branch first.
- Make the commit messages informative not like: "*new changes*", give a short description like: "*changed make_node() to be more readable*"
- Before starting to implement a new feature, always fetch and pull the changes in the remote branch which is often called the upstream.
- If you want to revert uncommitted local changes use:

```
git reset --hard
```

- If you want to go to a specific commit:

```
git checkout {sha1-code}
```

- Select one commit and apply it to the active branch:

```
git cherry-pick {sha1-code}
```

fournova

GIT CHEAT SHEET

presented by **TOWER** › Version control with Git - made easy

CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit

...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```

References:

- https://sites.google.com/a/lisa.iro.umontreal.ca/lisainfo/getting-started/welcome/tutorials-kick-off-meeting/tutorials-2015/git_github_tutorial
- https://learngitbranching.js.org
- http://git-school.github.io/visualizing-git/
- https://git-scm.com/docs/
- http://rogerdudler.github.io/git-guide/
- https://www.slideshare.net/anuragupadhaya/git-101-for-beginners
- https://github.com/jlord/git-it-electron#what-to-install