GIT & GITHUB

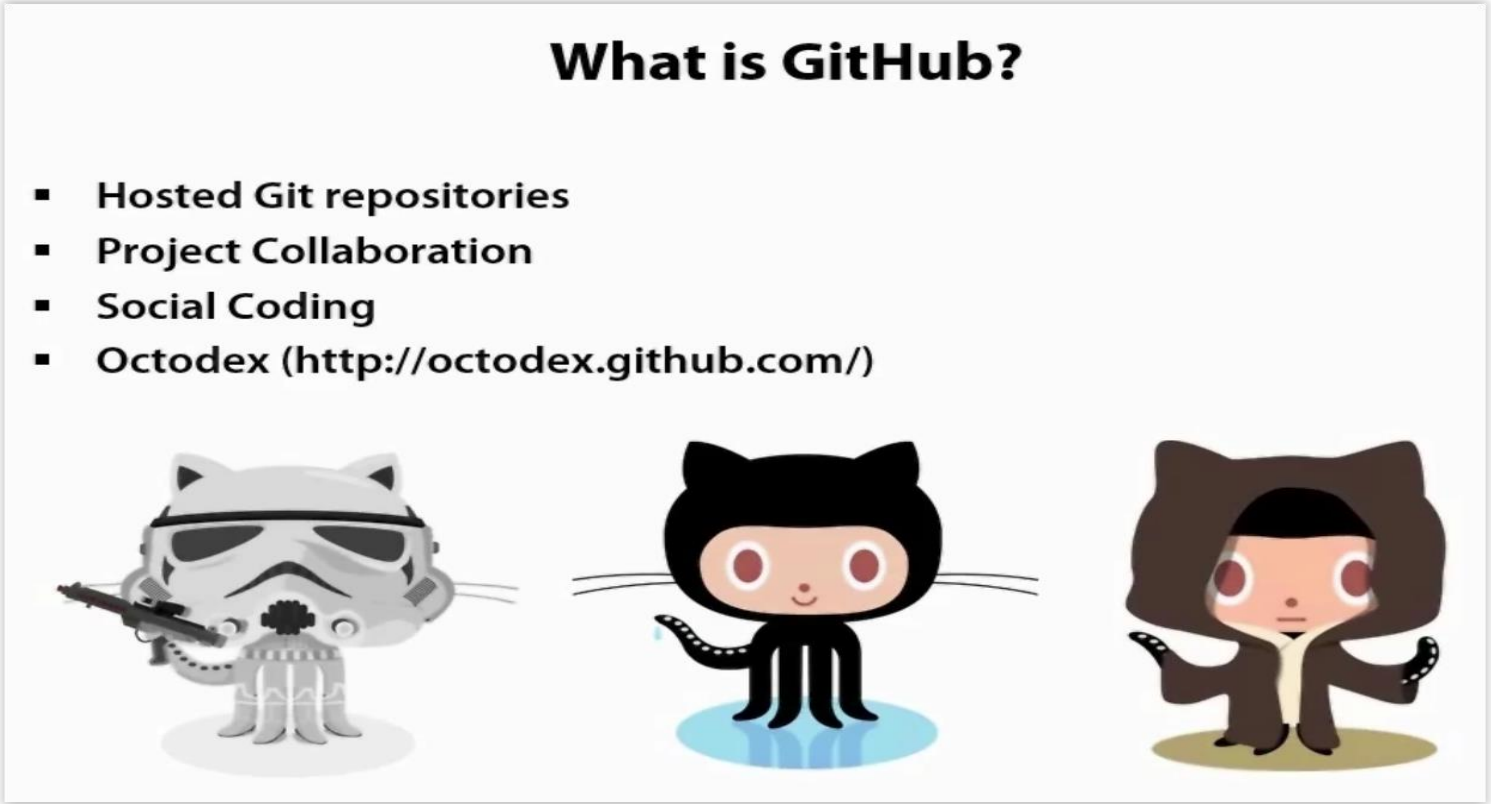
DevOps Course

JOHN BRYCE
Leading in IT Education
matrix company

Github What is It?

- GitHub is a code hosting platform for version control and collaboration.
- It lets you and others work together on projects from anywhere.

Github Intro



What is GitHub?

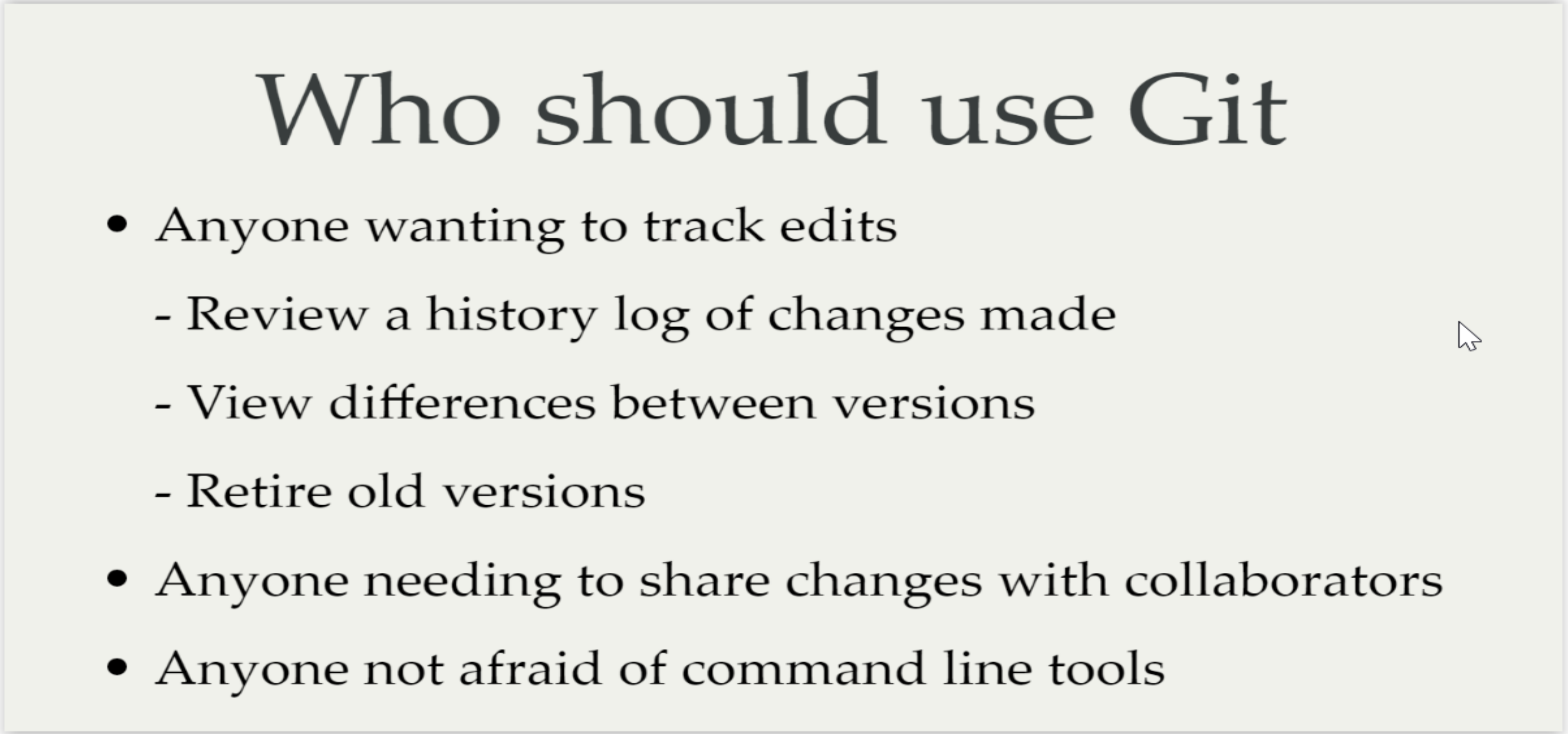
- Hosted Git repositories
- Project Collaboration
- Social Coding
- Octodex (http://octodex.github.com/)

What is Git(1)

What is git

Manage your source code versions

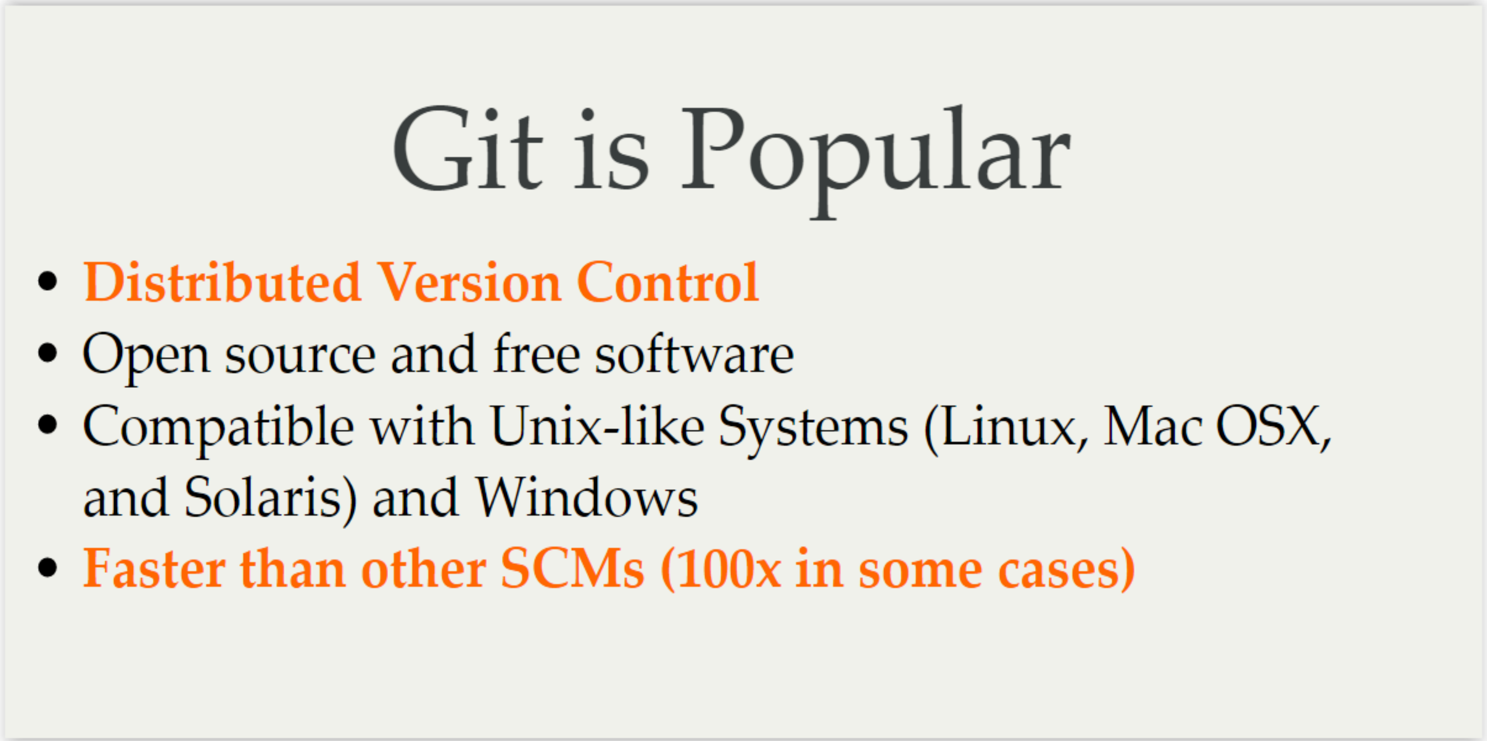
What is Git(2)



Who should use Git

- Anyone wanting to track edits
  - Review a history log of changes made
  - View differences between versions
  - Retire old versions
- Anyone needing to share changes with collaborators
- Anyone not afraid of command line tools

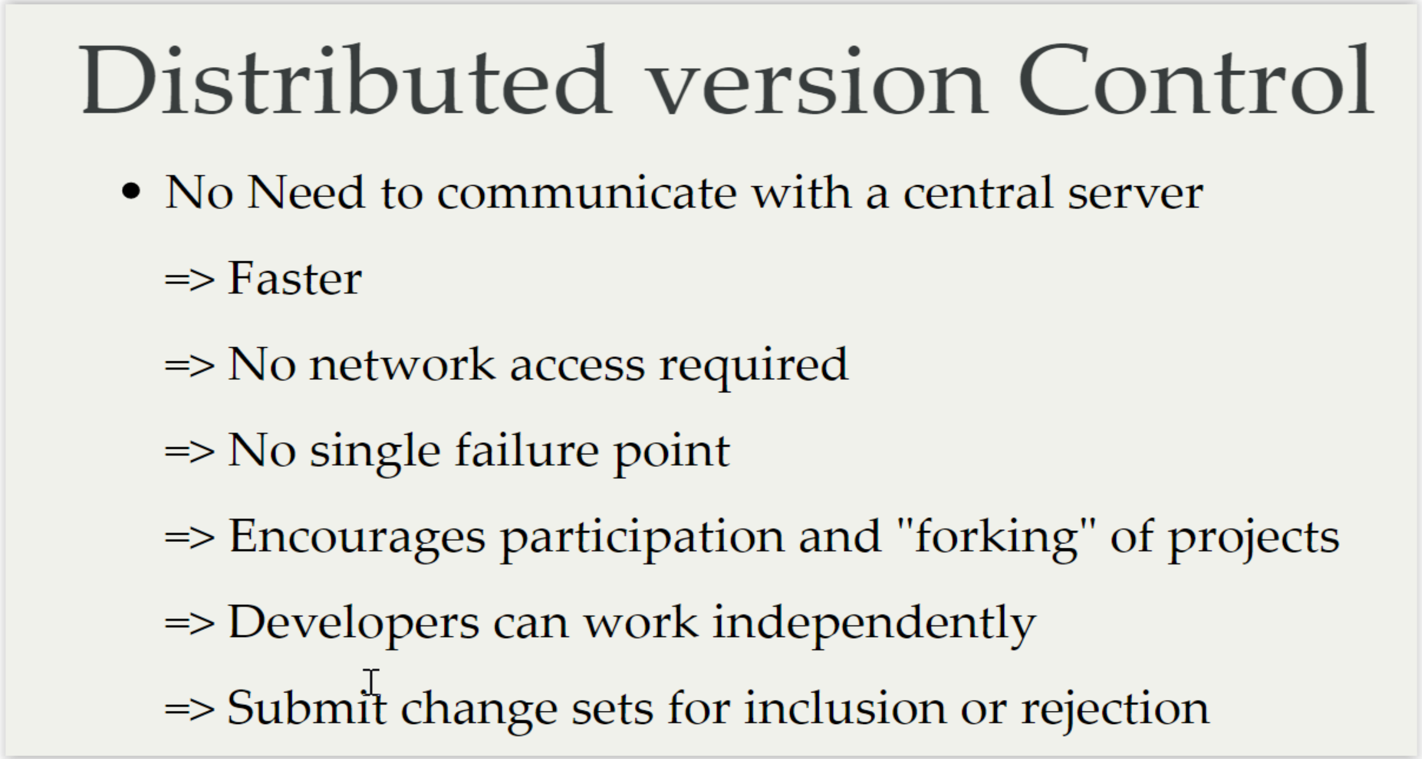
What is Git(3)



Git is Popular

- Distributed Version Control
- Open source and free software
- Compatible with Unix-like Systems (Linux, Mac OSX, and Solaris) and Windows
- Faster than other SCMs (100x in some cases)

What is Git(4)



Distributed version Control

- No Need to communicate with a central server

  => Faster

  => No network access required

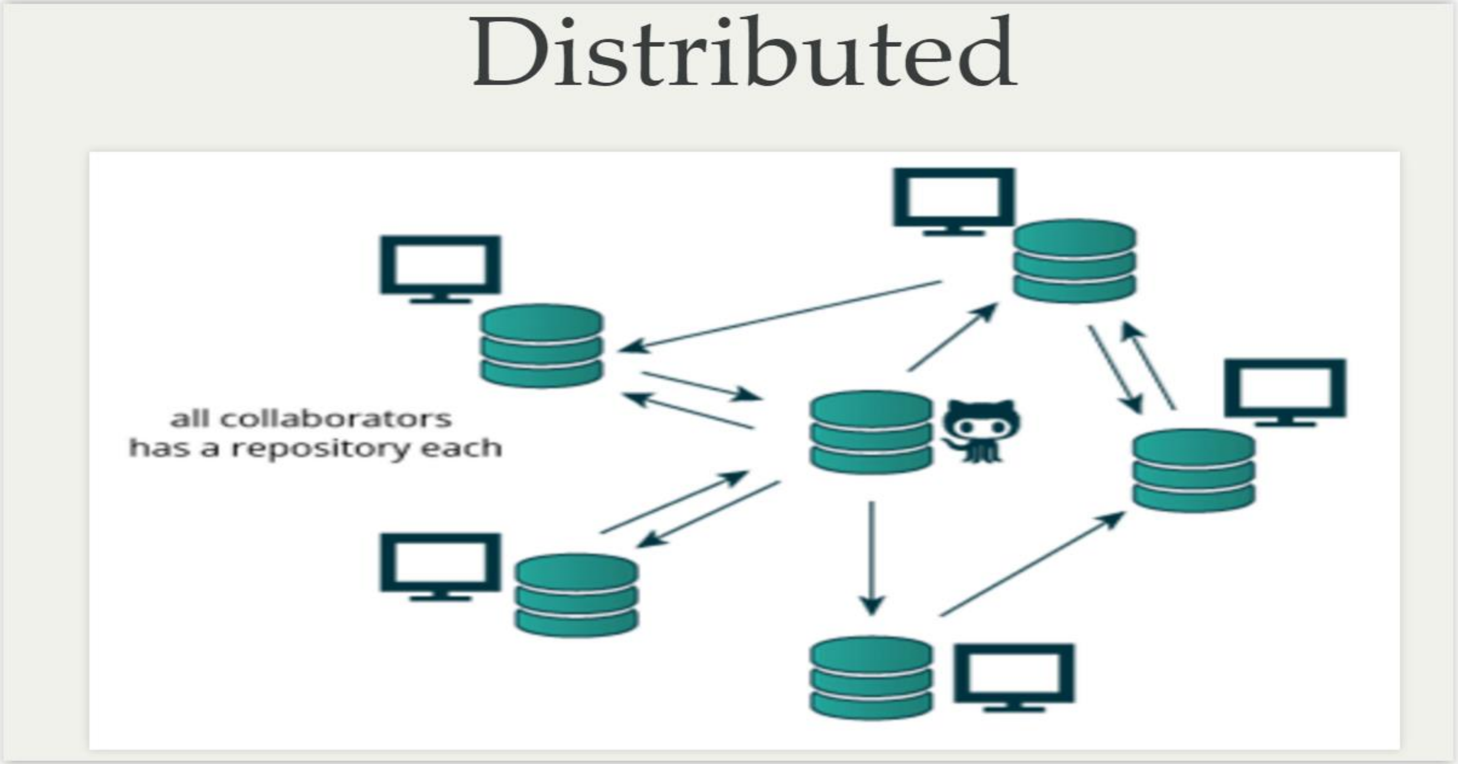
  => No single failure point

  => Encourages participation and "forking" of projects

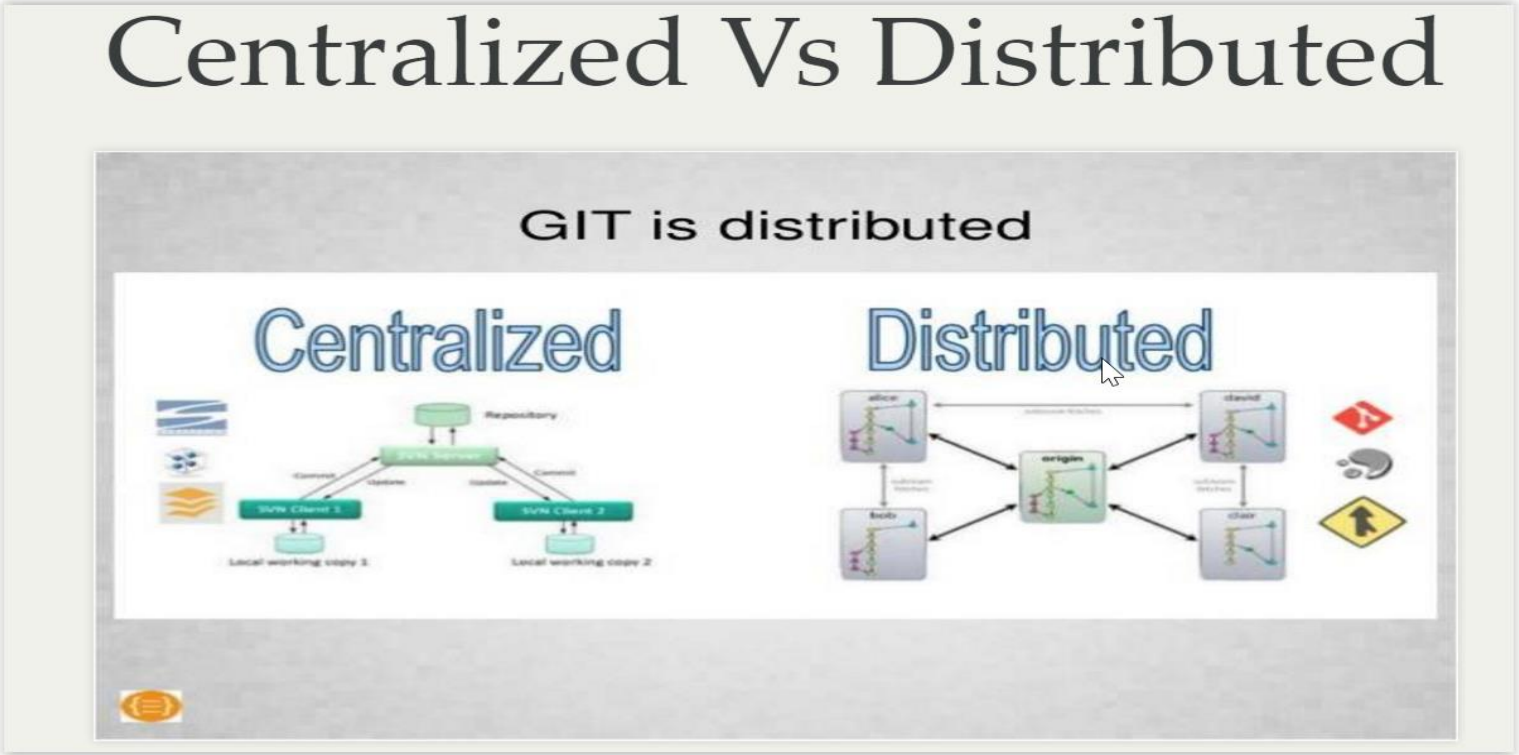
  => Developers can work independently

  => Submit change sets for inclusion or rejection

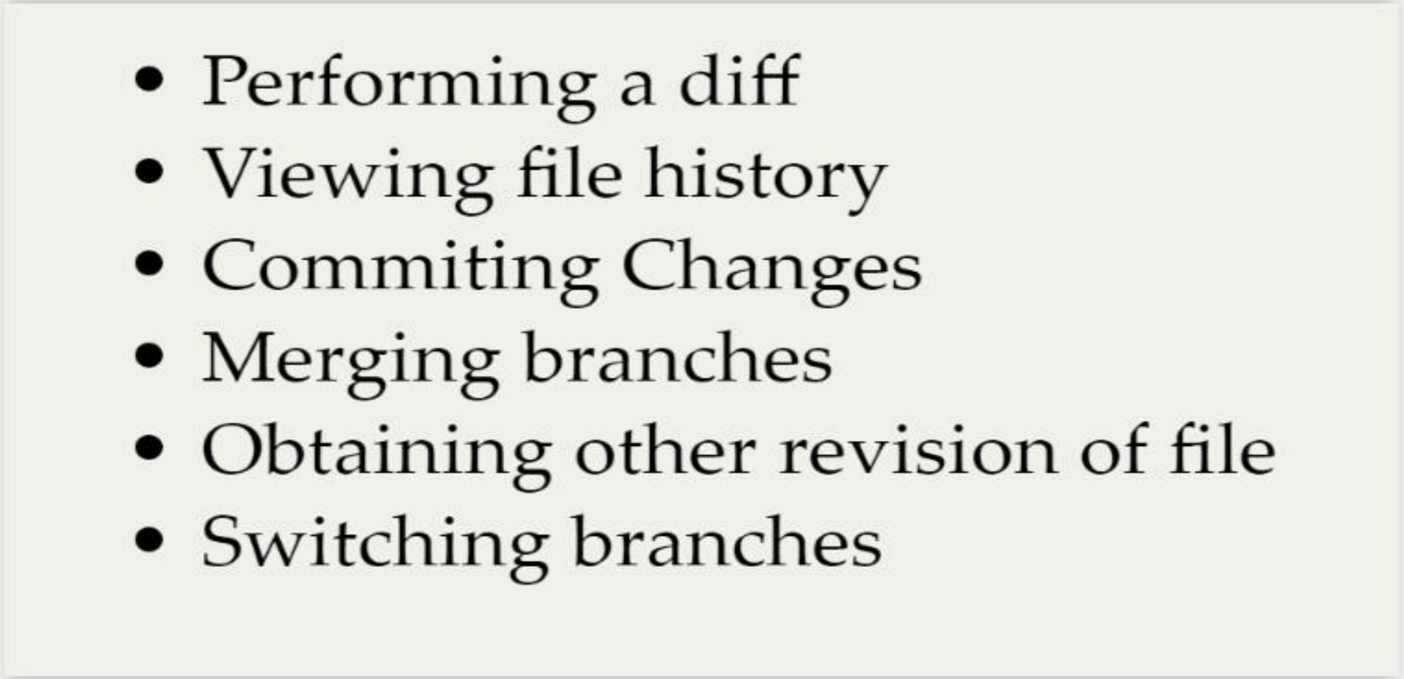
What is Distributed



What is Distributed(2)

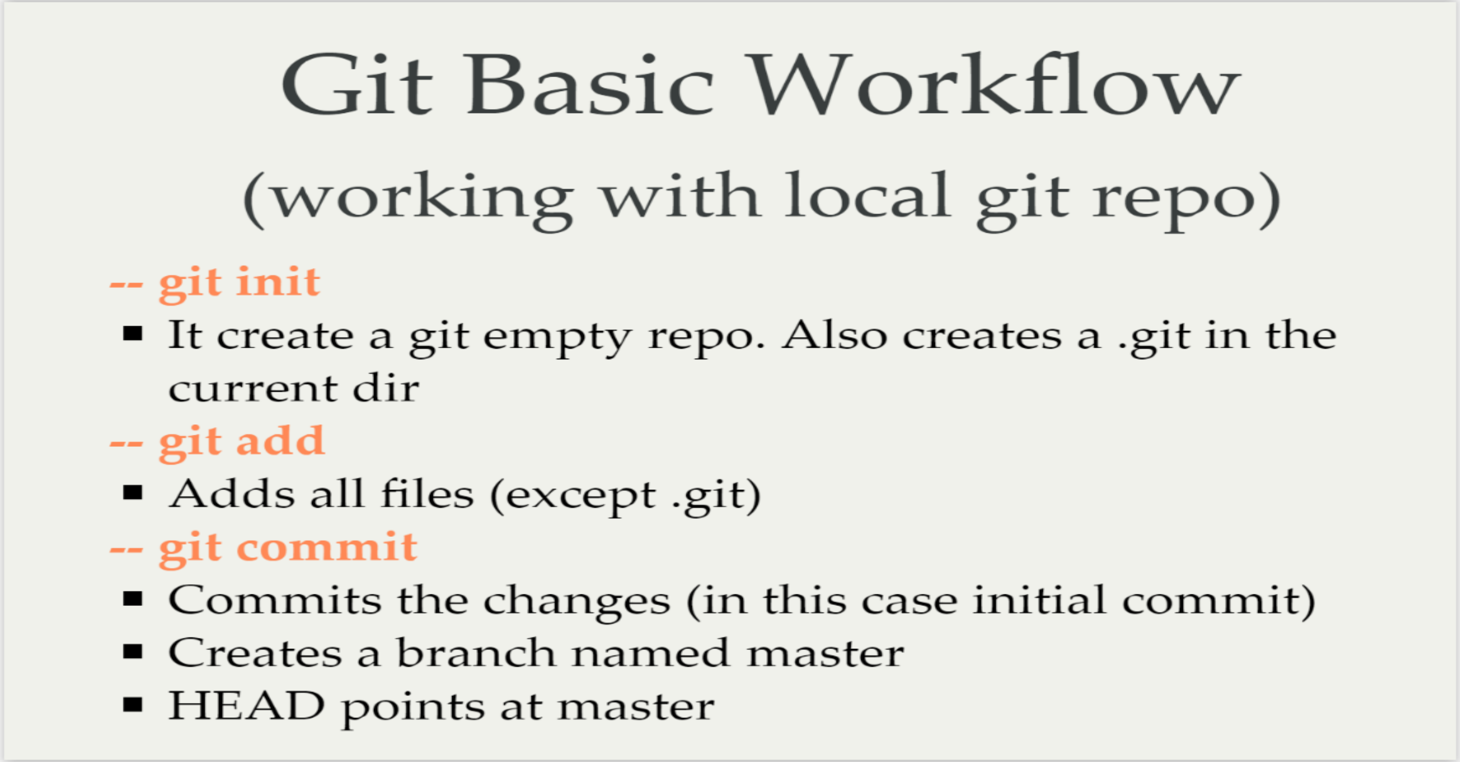


GIT Capablities



- Performing a diff
- Viewing file history
- Commiting Changes
- Merging branches
- Obtaining other revision of file
- Switching branches

GIT Getting started



Git Basic Workflow

(working with local git repo)

-- git init

- It create a git empty repo. Also creates a .git in the current dir

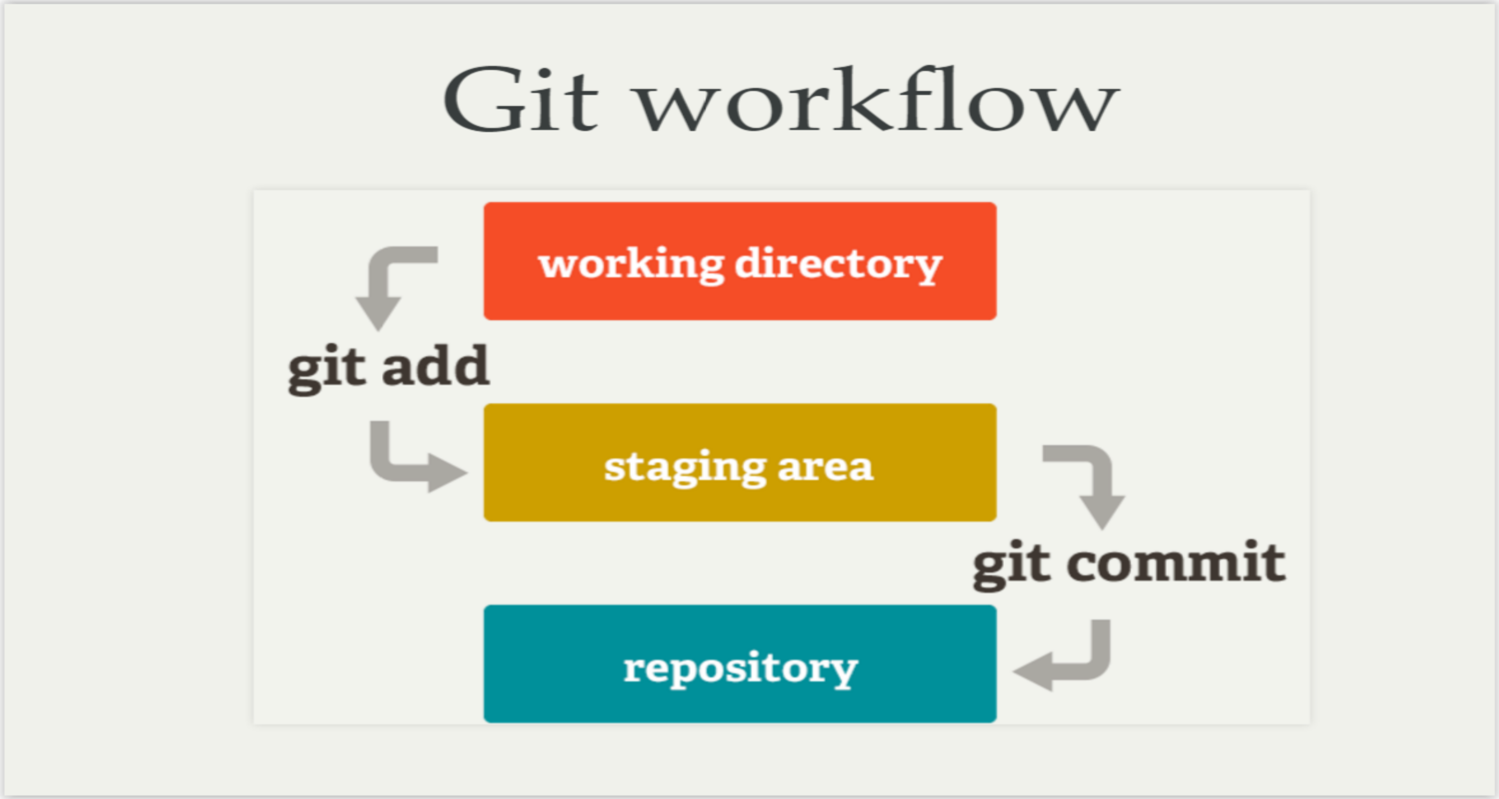
-- git add

- Adds all files (except .git)

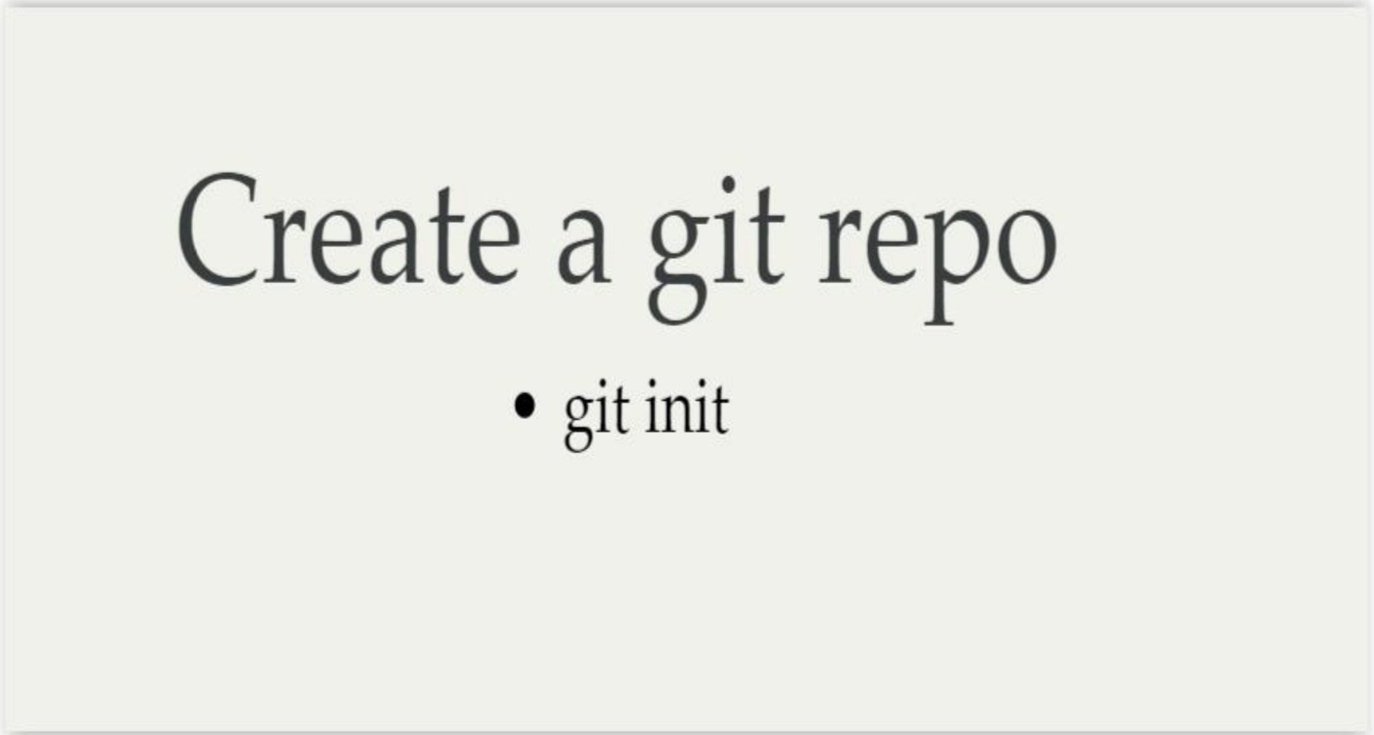
-- git commit

- Commits the changes (in this case initial commit)
- Creates a branch named master
- HEAD points at master

GIT WorkFlow

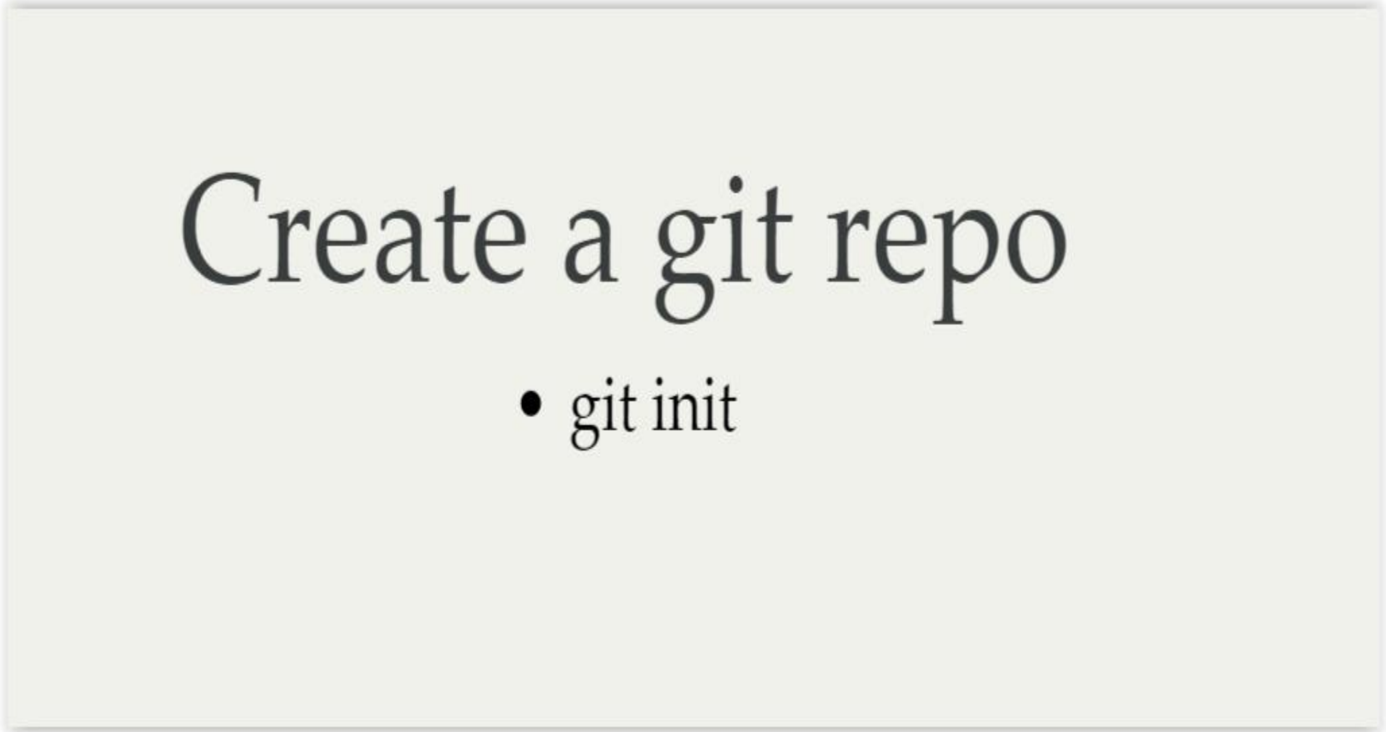


Using Git from shell


Create a git repo

- git init

Using Git from shell

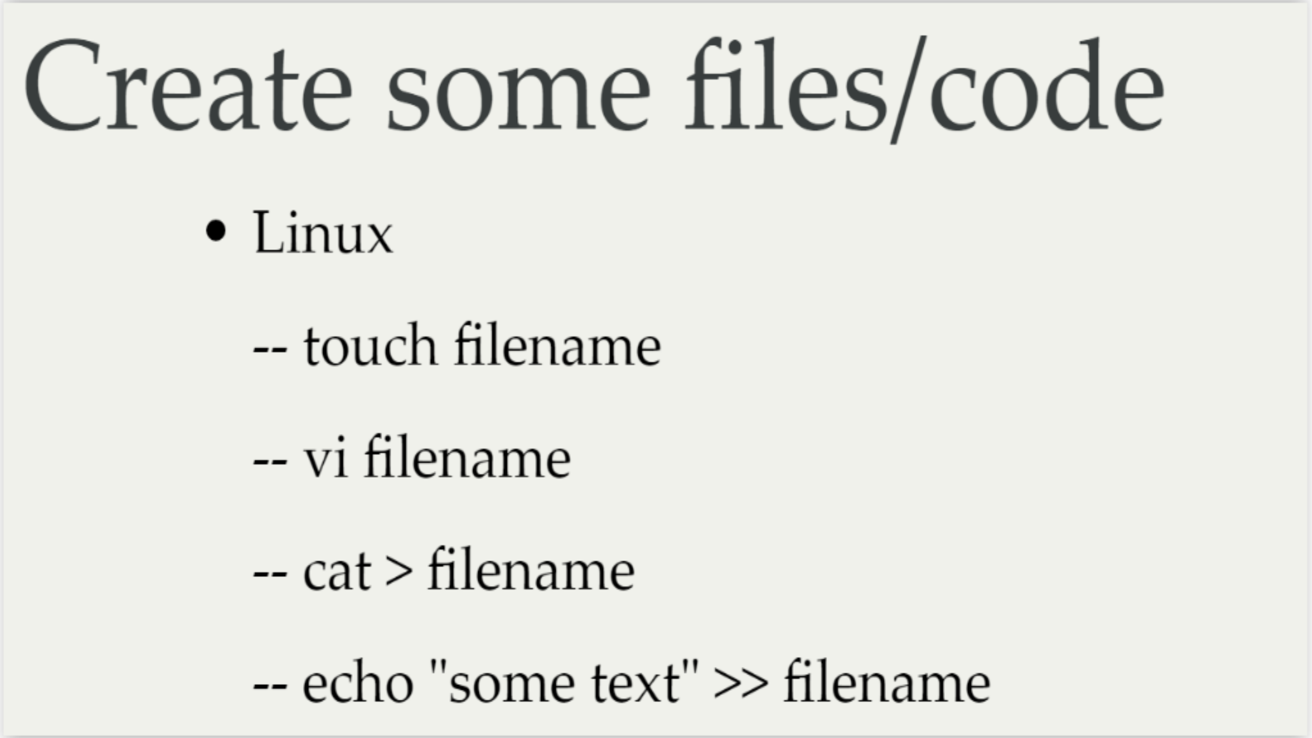

Create a git repo

- git init

Configuring Git(1)

- git config --global user.name “<yourname>”
- git config --global user.email someone@email.com
- git config --list

Configuring Git(2)



Create some files/code

- Linux

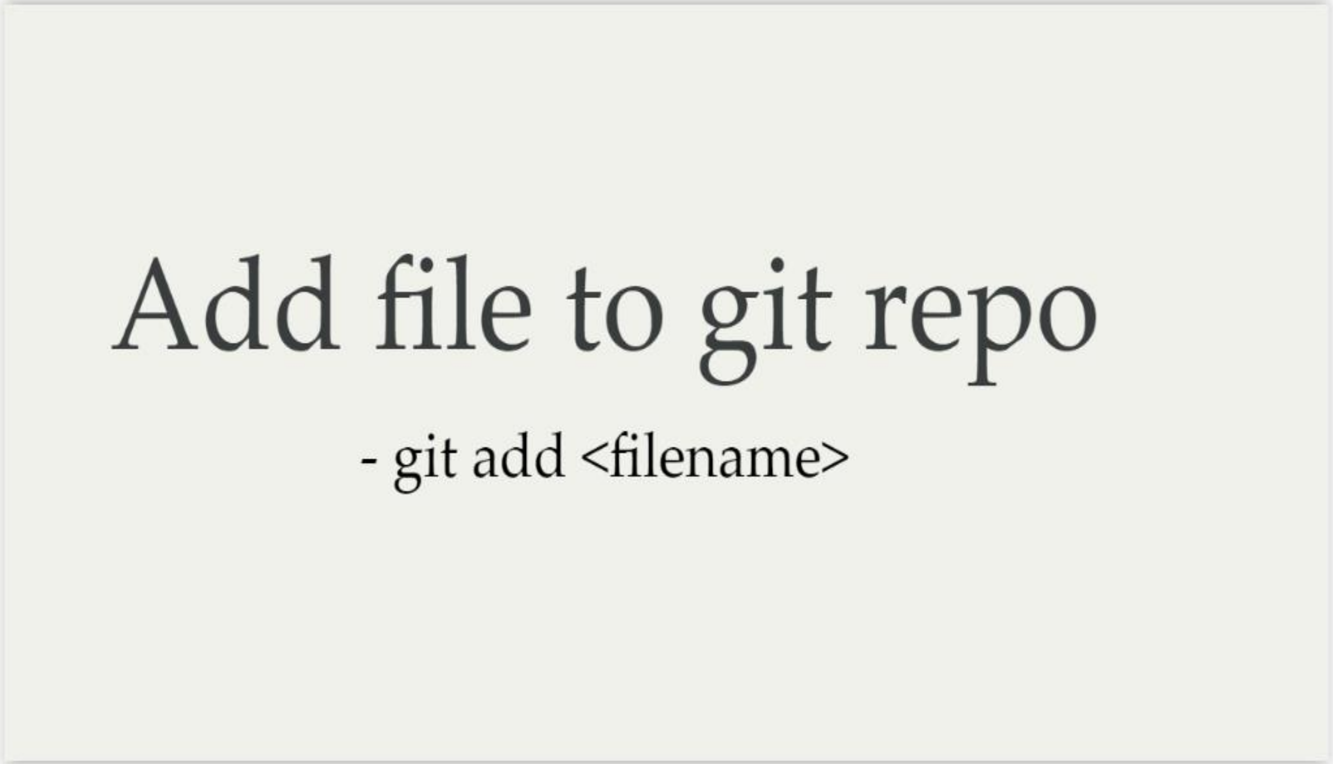
  -- touch filename

  -- vi filename

  -- cat > filename

  -- echo "some text" >> filename

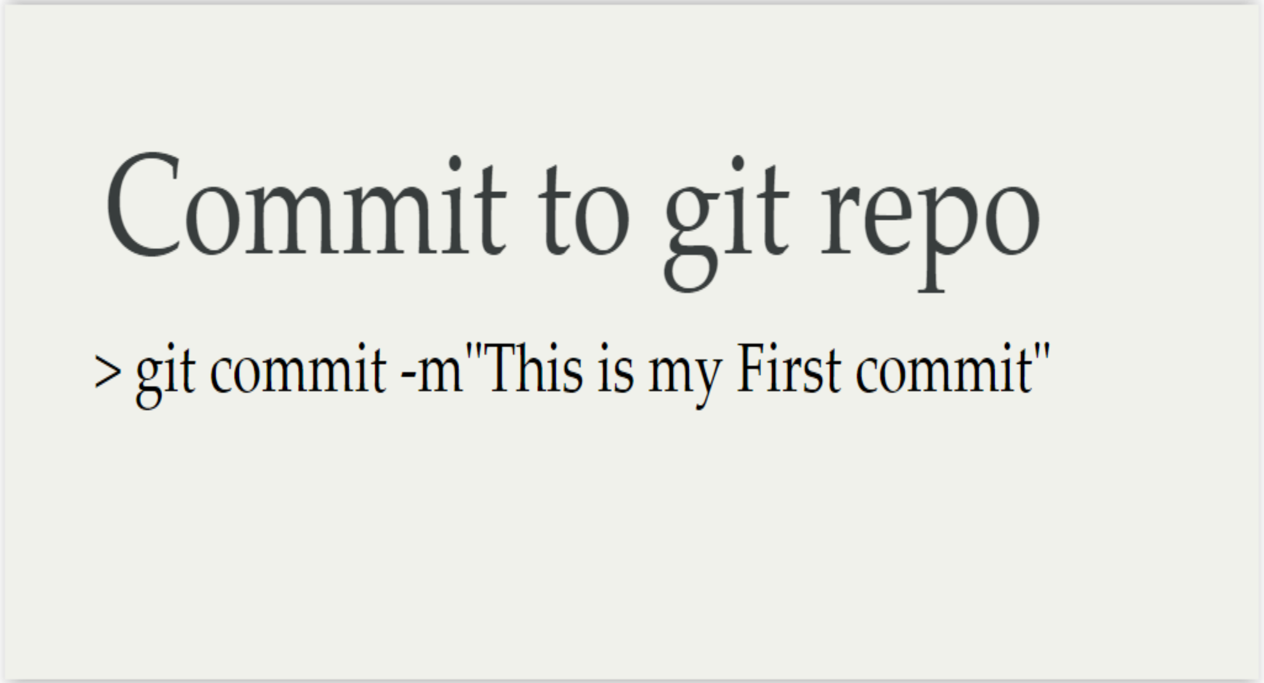
Configuring Git(3)



Add file to git repo

- git add <filename>

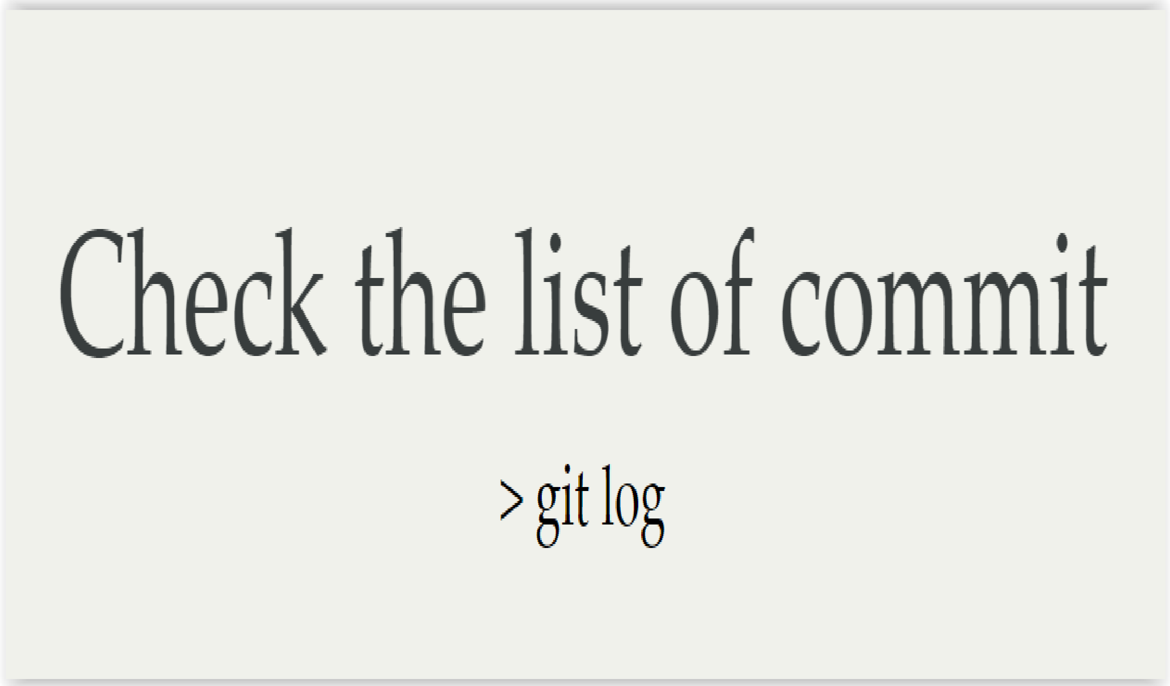
Configuring Git(4)



Commit to git repo

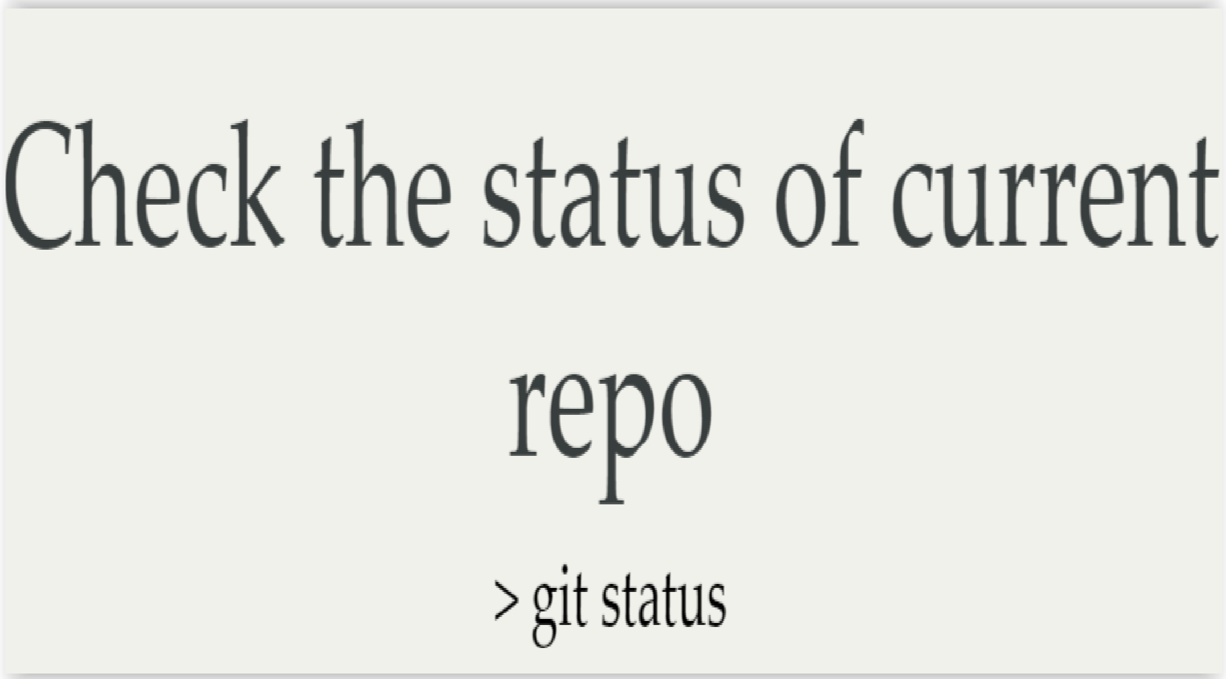
```
> git commit -m"This is my First commit"
```

Configuring Git(5)


Check the list of commit

```
> git log
```

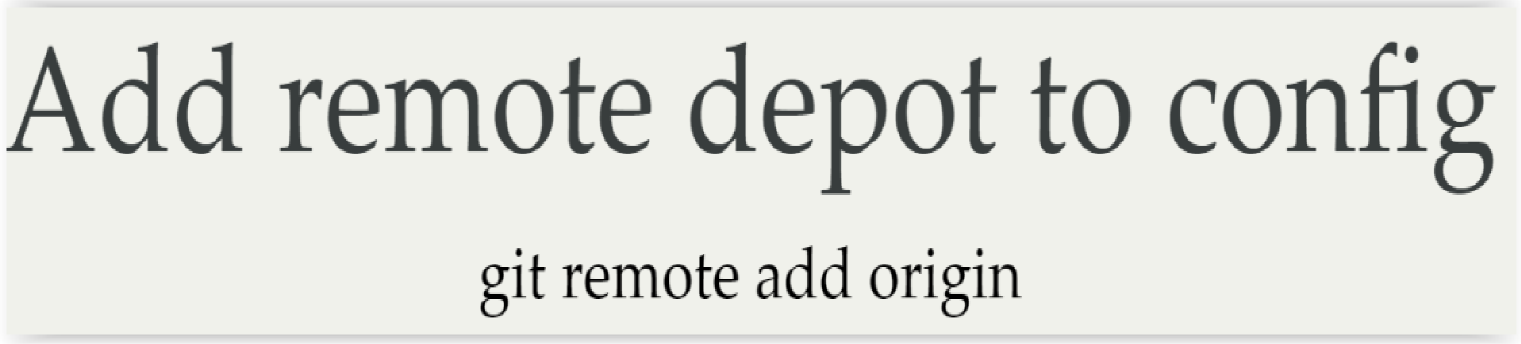
Configuring Git(6)



Check the status of current repo

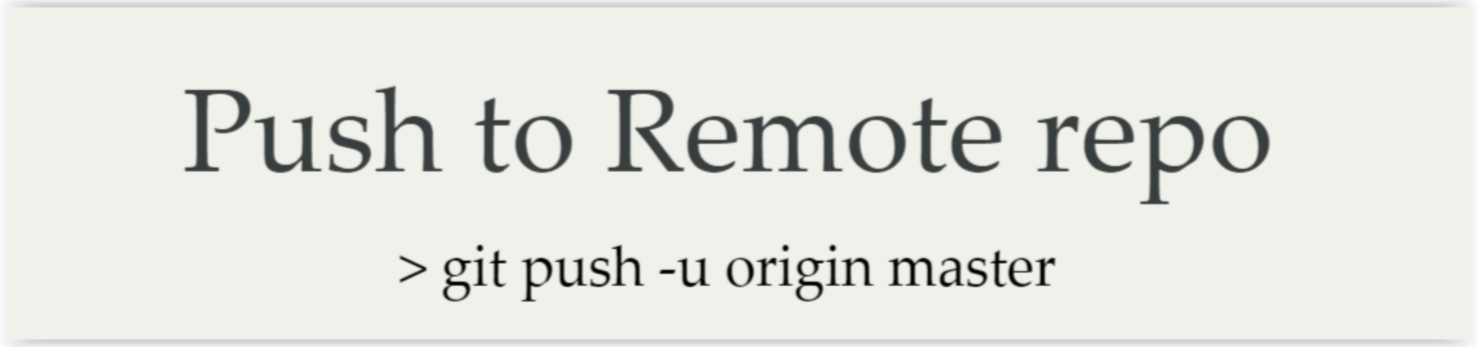
```
> git status
```

Github Remote Repo



Add remote depot to config

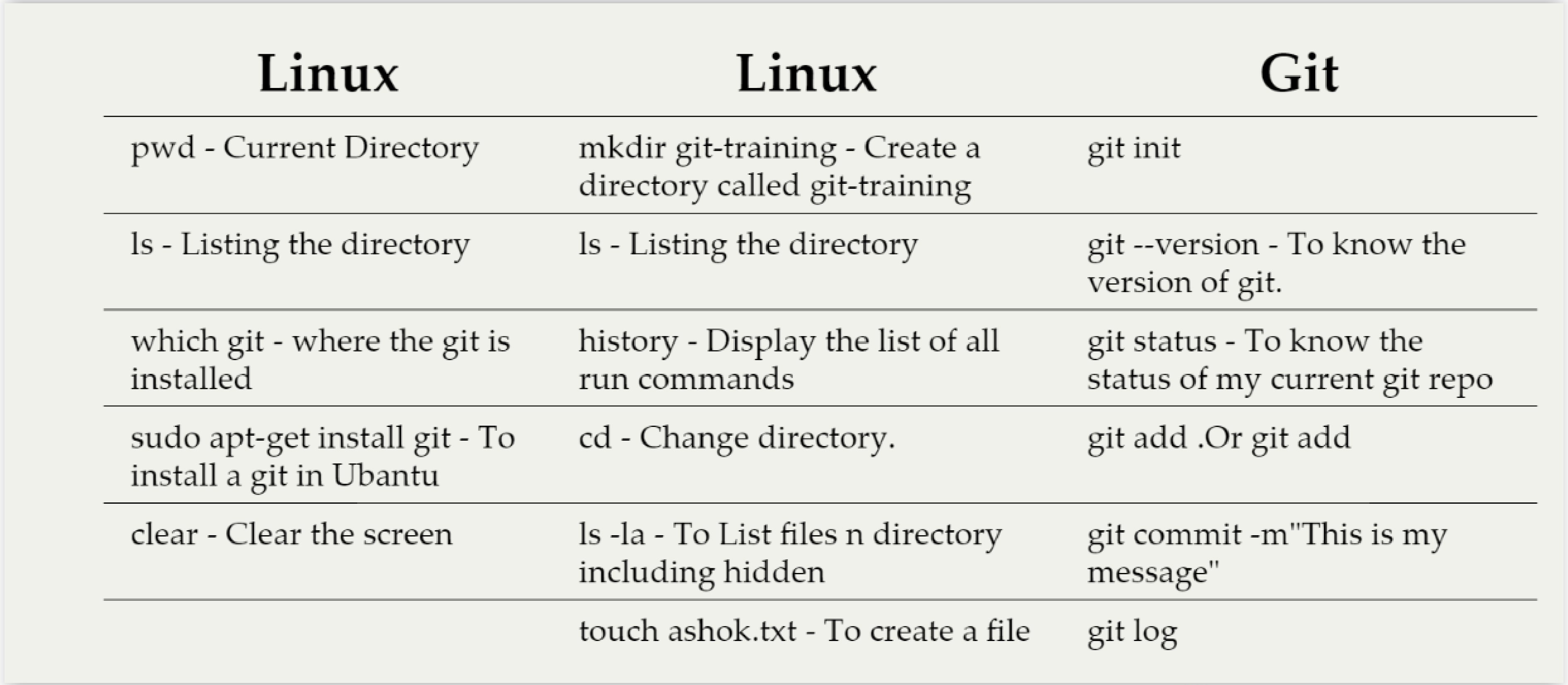
```
git remote add origin
```



Push to Remote repo

```
> git push -u origin master
```

github to local GIT workflow



Linux	Linux	Git
pwd - Current Directory	mkdir git-training - Create a directory called git-training	git init
ls - Listing the directory	ls - Listing the directory	git --version - To know the version of git.
which git - where the git is installed	history - Display the list of all run commands	git status - To know the status of my current git repo
sudo apt-get install git - To install a git in Ubantu	cd - Change directory.	git add .Or git add
clear - Clear the screen	ls -la - To List files n directory including hidden	git commit -m"This is my message"
	touch ashok.txt - To create a file	git log

JOHN BRYCE
Leading in IT Education
matrix company

GIT KEY CONCEPTS

GIT Key Concepts

SnapShots

Are the way git keeps track of your code history.

Essentially what git does is to record what all your files look like at a given point in time. You as the user have the ability decide when to take a snapshot by committing your changes , and of what files.

Once done, you'll have the ability to go back to visit any snapshot,
Review, delete or rollback to them

GIT Key Concepts

Commit

Is the act itself of creating a snapshot.

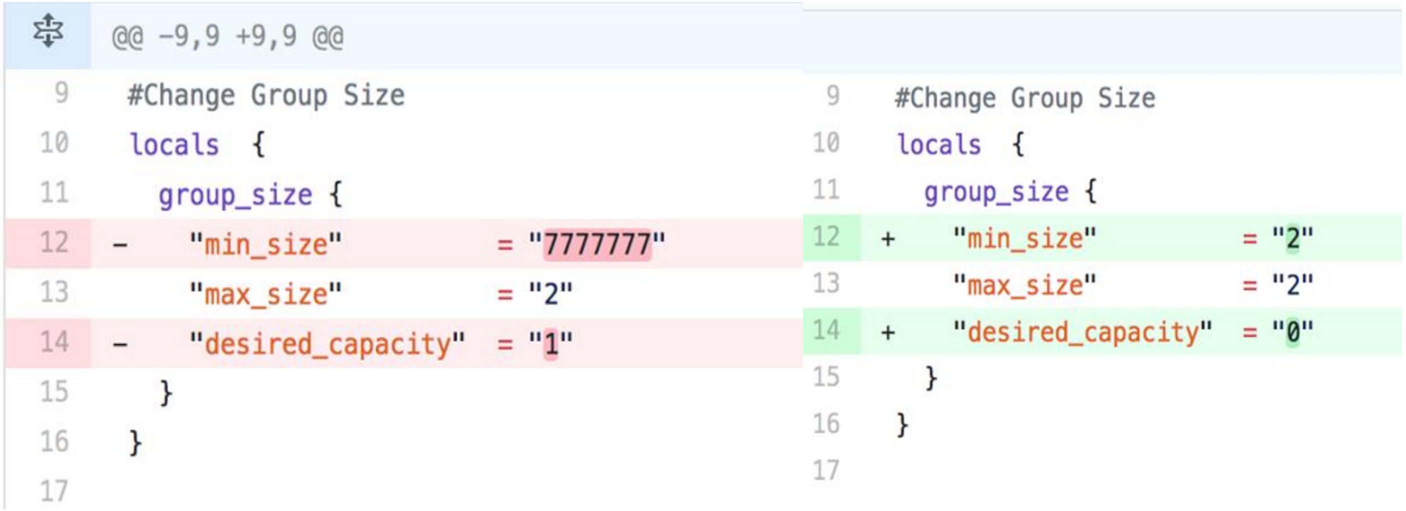
Eventually your project will be made up of a bunch of commits you perform along the lifetime of your project.

GIT Key Concepts

Commits

Contains three layers of information:

1. Information about how the files changed from previous commit[s]



```
@@ -9,9 +9,9 @@
 #Change Group Size                                 #Change Group Size
 locals  {                                          locals  {
   group_size {                                       group_size {
-    "min_size"          = "7777777"             +    "min_size"          = "2"
     "max_size"          = "2"                        "max_size"          = "2"
-    "desired_capacity"  = "1"                   +    "desired_capacity"  = "0"
   }                                                  }
 }                                                  }
```

GIT Key Concepts

Commits

2. A reference to the commit that came before it - Called the “parent commit”

3. As Git is a content-addressable filesystem. meaning that at the core of Git is a simple key-value data store. Git Stores everything as a file and uses Hashcode name as the KEY and your changes as the VALUE.

Once we COMMIT - Git will hand us back a unique key we can use later to retrieve that content. Git Hash looks something like this:

fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e

GIT Key Concepts

Repositories aka Repo

Are a collection of all the files and the history of those files you committed. The Repo consists with all of your versions, branches and history of your code / files.

Git Repo can live on a local (on premises)machine or as a SaaS on a remote server (such as GitHub!).

GIT Key Concepts

Repositories aka Repo

Common Tasks on a Repo:

- Copying a repository from a remote server/repo is called **cloning**. Cloning allow allow teams to work together in their local system.

- **Pulling Changes** is The process of downloading commits that don't exist on your local machine from a remote repository.

- **Pusing** is the process of adding your local changes to the remote repository.

GIT Key Concepts

Branching

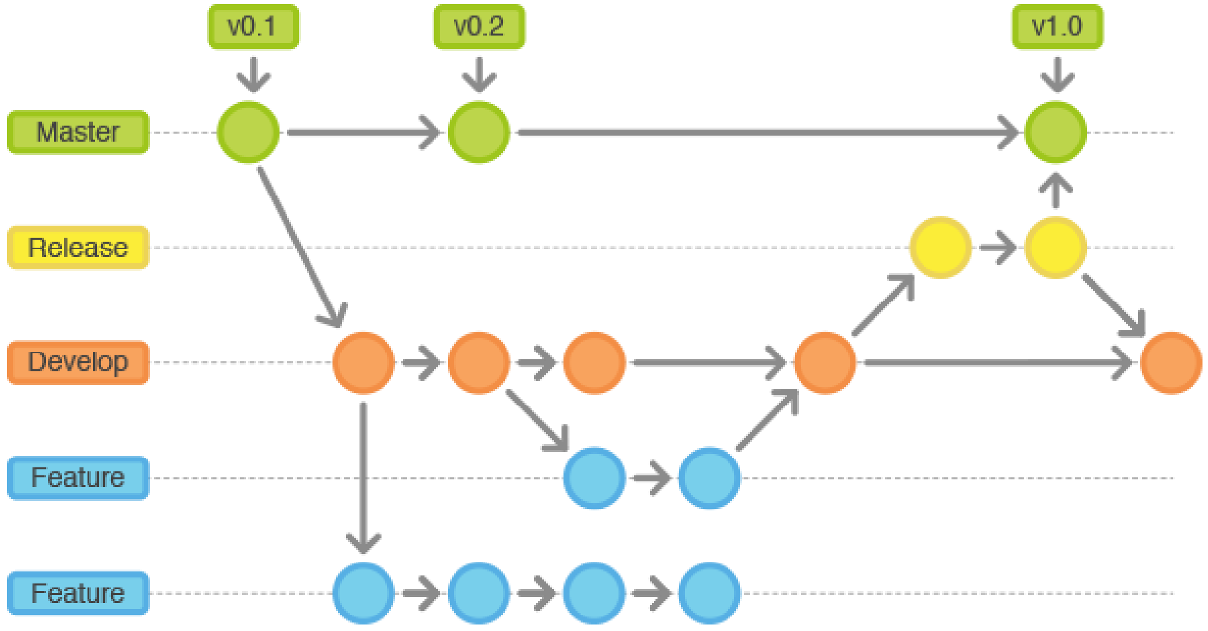
Branches are the main part in Git and SC systems,

Branches are used to develop **features isolated** from each other.
Meaning that we can develop different features of our product[s]
While we dont “mess” with current working branch.

The **master** branch is the "default" branch when you create a repository. We Use other branches for development and merge them back to the master branch upon completion

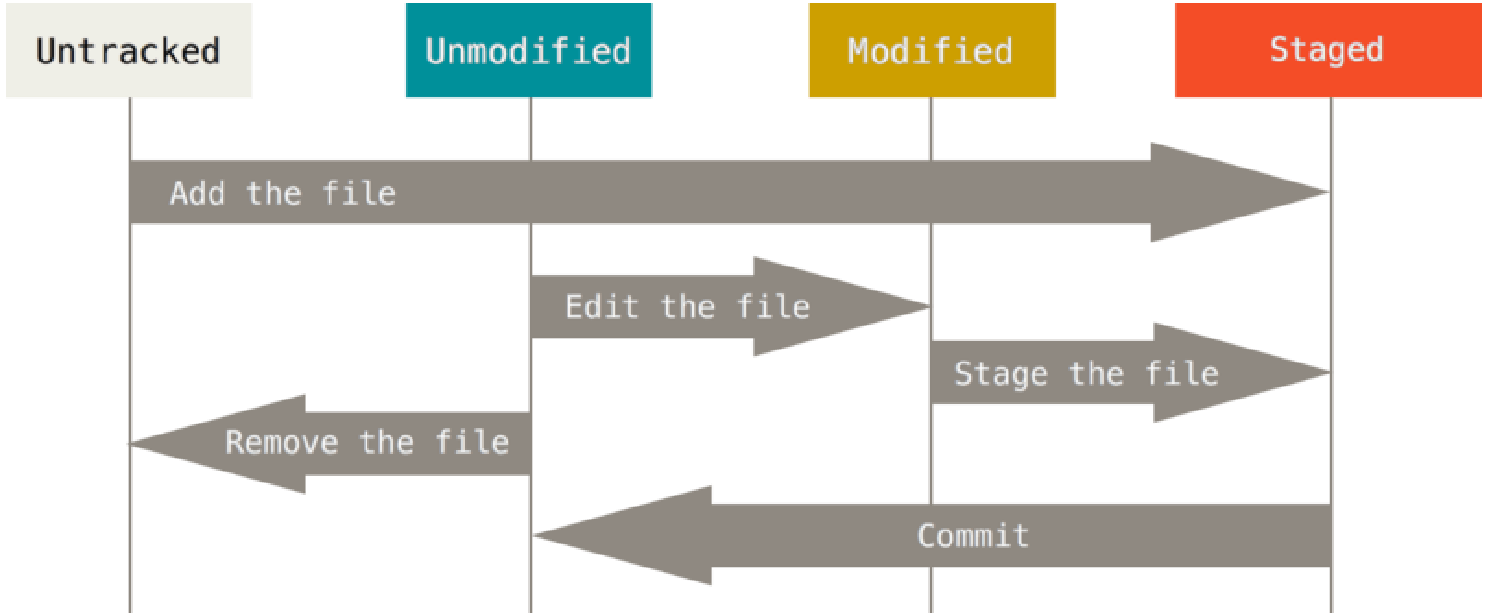
GIT Key Concepts

Branching



3 STAGES FLOW

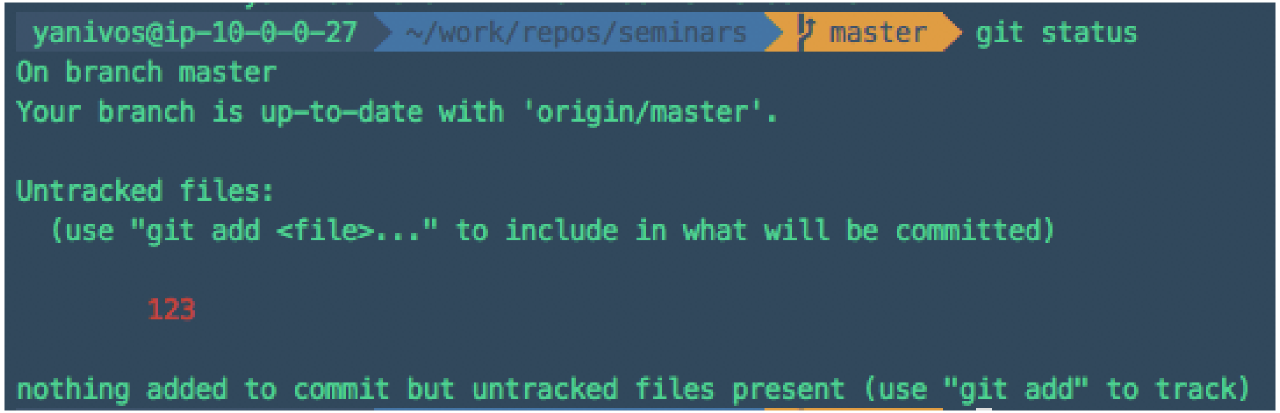
GIT FLOW


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

GIT FLOW

Untracked

Are any files in your working directory that were not in your last snapshot and are not in your staging area and were not committed

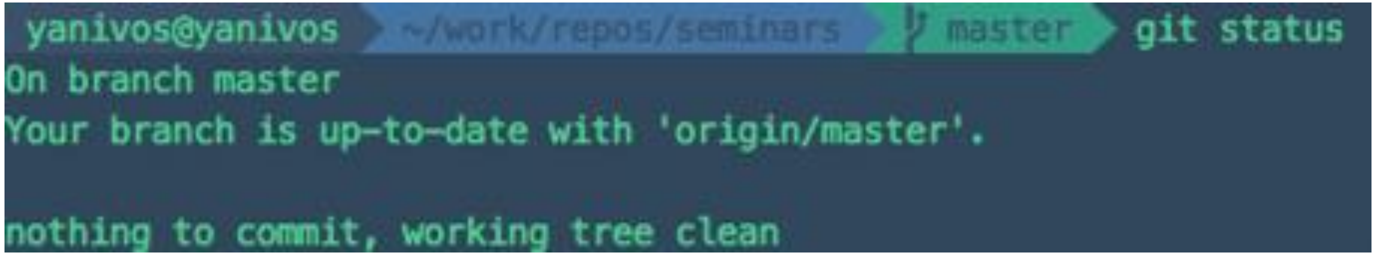


```
 yanivos@ip-10-0-0-27  ~/work/repos/seminars  master  git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        123

nothing added to commit but untracked files present (use "git add" to track)
```

GIT FLOW

Unmodified

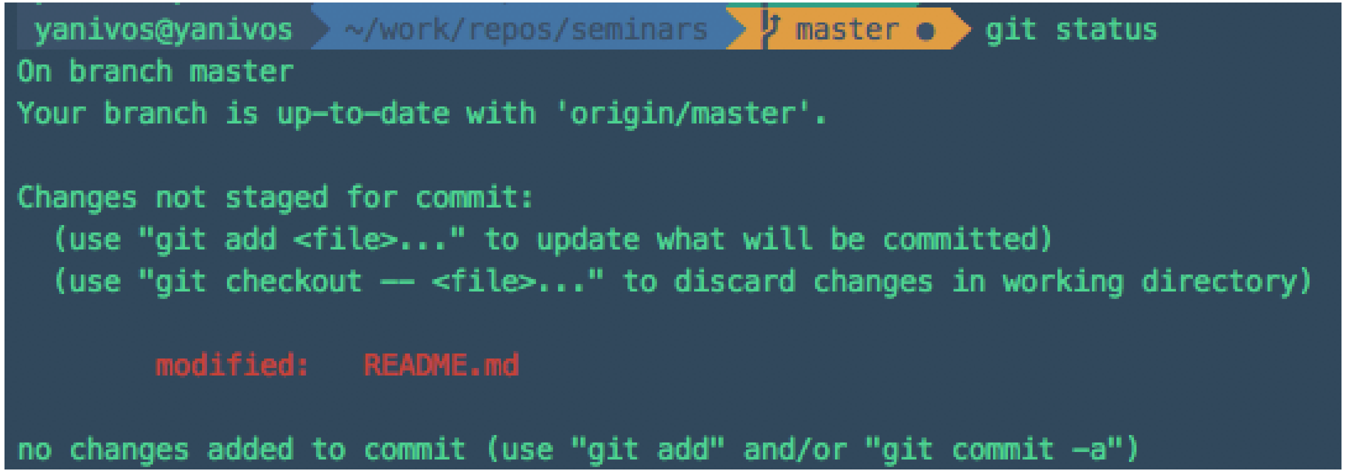
Are any files that you just checked out or cloned from a remote repository and you haven't edited anything yet.



```
yanivos@yanivos  ~/work/repos/seminars  master  git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean
```

GIT FLOW

`Modified` Are any files in your working directory that were modified since your last snapshot / commit and are not staged

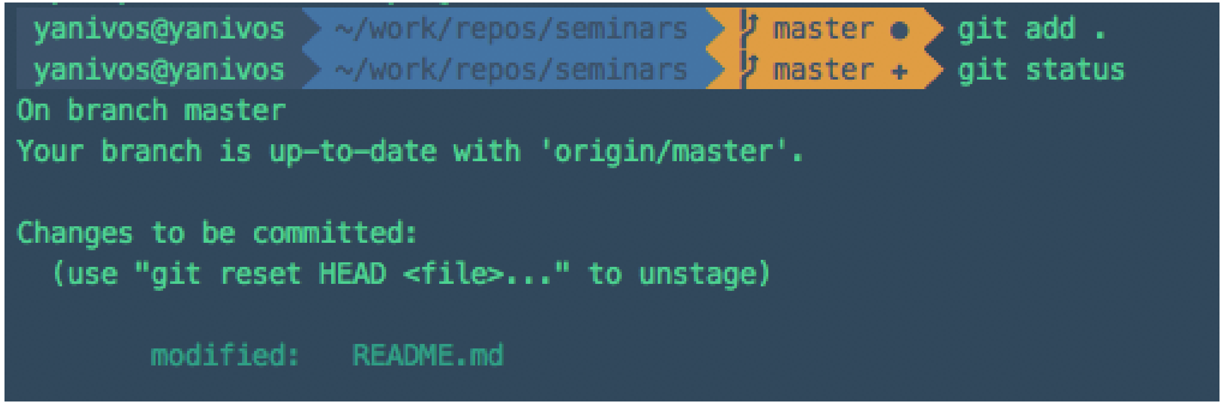


```
yanivos@yanivos  ~/work/repos/seminars  master ●  git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

GIT FLOW

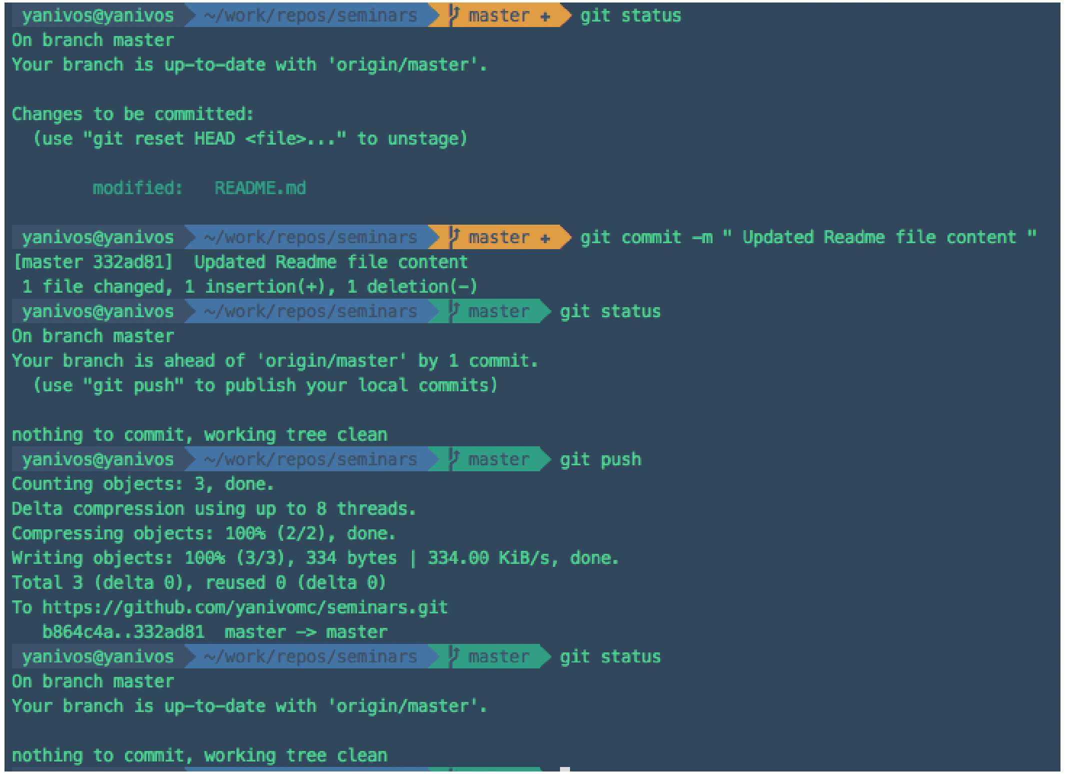
Staged

Are any files in your working directory that were modified and added since your last snapshot / commit and are **NOW** staged and ready to be **committed**



```
yanivos@yanivos  ~/work/repos/seminars  master ●  git add .
yanivos@yanivos  ~/work/repos/seminars  master +  git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

GIT FLOW



```
 yanivos@yanivos  ~/work/repos/seminars  master +  git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

 yanivos@yanivos  ~/work/repos/seminars  master +  git commit -m " Updated Readme file content "
[master 332ad81]  Updated Readme file content
 1 file changed, 1 insertion(+), 1 deletion(-)
 yanivos@yanivos  ~/work/repos/seminars  master  git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
 yanivos@yanivos  ~/work/repos/seminars  master  git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 334 bytes | 334.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/yanivomc/seminars.git
   b864c4a..332ad81  master -> master
 yanivos@yanivos  ~/work/repos/seminars  master  git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean
```

Once we added the file[s], we can commit them to our local repo and then push the changes to the remote repository

COMMIT

PUSH

UNCHANGED

GIT LAB 1

HANDS ON

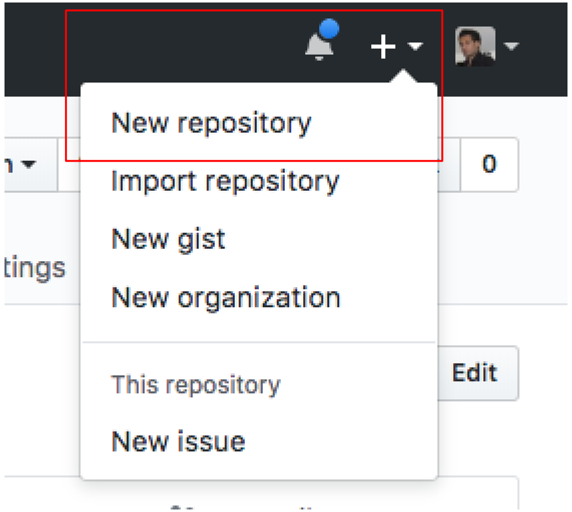
GIT LAB 1

Instruction

1. Create new Directory
   a. Open it and then perform git init which will create a new **local git repository**
2. Clone a remote repo
   a. git clone https://github.com/yanivomc/seminars
3. Study the repo history
   a. git log --stat or git log --pretty=oneline for compressed log where each commit is one line
4. See last commits to the repo and the changes made in a specific commit
   a. git show [commitID]
   b. git diff [1stCommitID] [2ndCommitID]
5. Revert to previous commit state and return to current state
   a. git checkout [commitID]
   b. git checkout master

Instruction

1. Create a new GITHUB account and a Repo at https://github.com/
   a. If you already have an account, login to it and click the “+” and select “New repository”

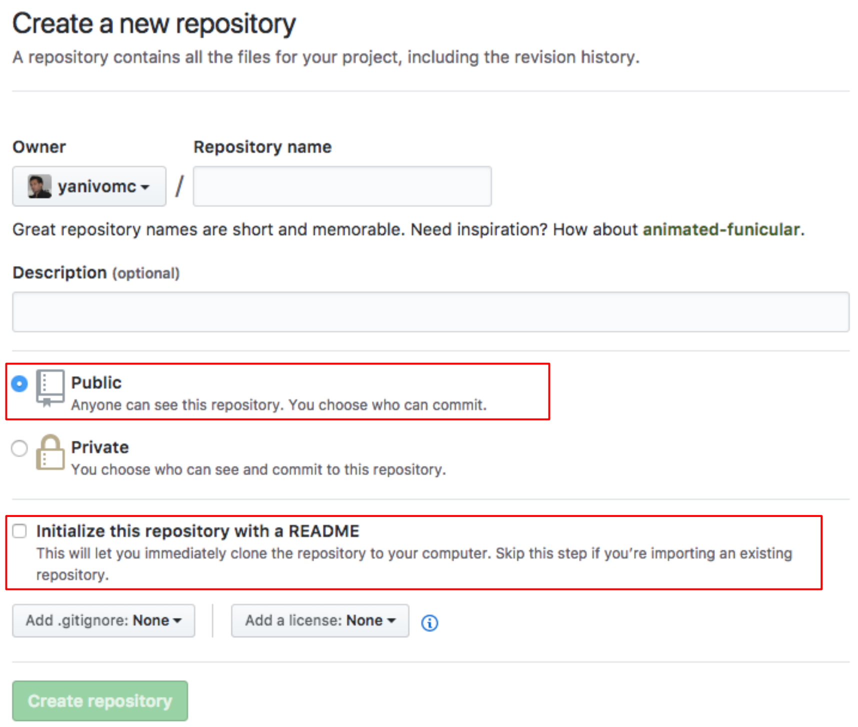


GIT LAB 2

Instruction

2. Name Your repo as you wish but -
   a. Mark it public
   b. Don't check the "initialize this repository with a README."

DONT



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: yanivomc / Repository name:

Great repository names are short and memorable. Need inspiration? How about **animated-funicular**.

Description (optional)

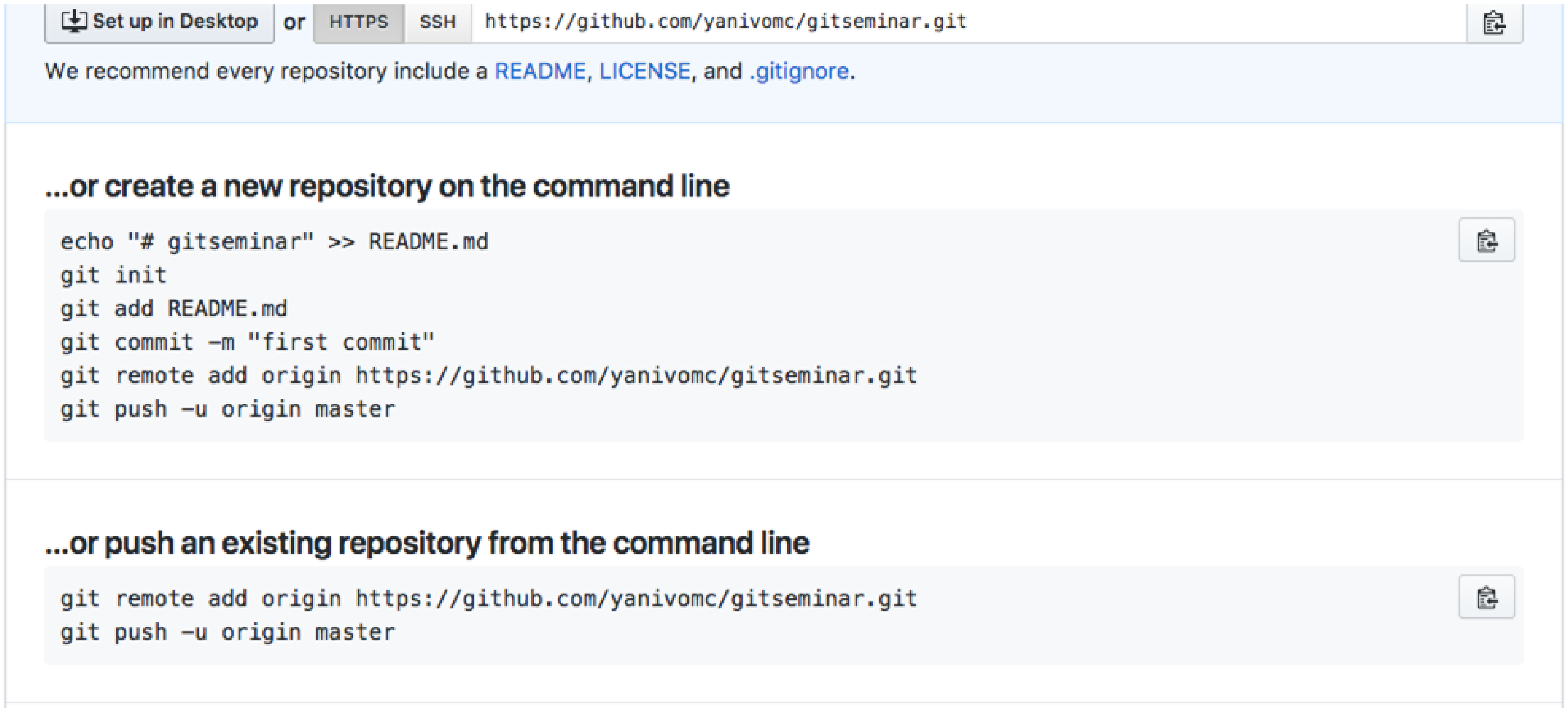
[✓ Public] Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

GIT LAB 2



Set up in Desktop or HTTPS SSH https://github.com/yanivomc/gitseminar.git

We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# gitseminar" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/yanivomc/gitseminar.git
git push -u origin master
```

…or push an existing repository from the command line

```
git remote add origin https://github.com/yanivomc/gitseminar.git
git push -u origin master
```

GIT LAB 2

Instruction

3. Create a new Directory
4. Add a new file called “git101.txt”
5. Write any 3 git commands inside the file
6. Save the file
7. Move the file to Staging Area
8. Commit the file with a comment

GIT LAB 2

JOHN BRYCE
Leading in IT Education
matrix company

Let’s go over it

Adding files:

1. Add new file[s] to staging area - `git add <filename> / git add *`
2. Commit changes - `git commit -m “Commit message”`

GIT LAB 2 Continue

Push changes to remote Origin

Pushing changes:

1. Push the file to your Github repository master (default) branch
2. Create new branch and push your changes to that branch also

But Wait..

Remote & Origin ??

Origin

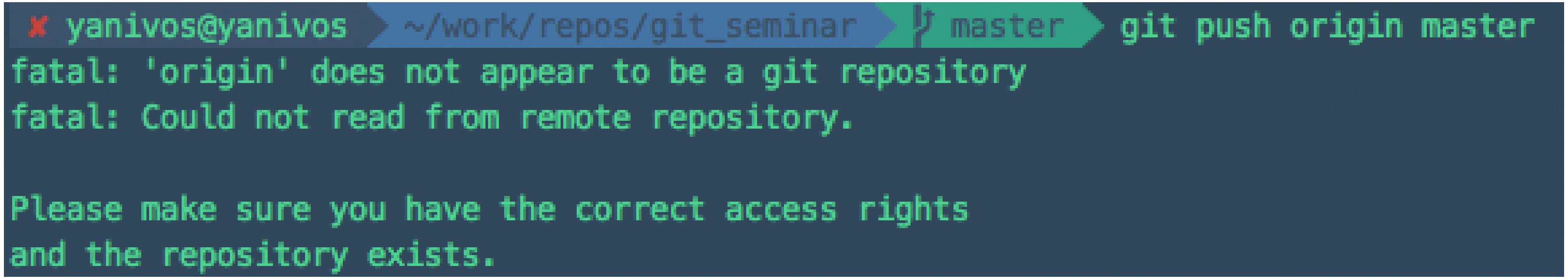
When we clone a repository for the first time,

Origin is the default name that GIT set for the remote repository we cloned, from where we pull and push changes.

`git push origin master` means that we are pushing to the **origin** repository and to the default master branch .

Let's try running push command

GIT LAB 2



```
✘ yanivos@yanivos  ~/work/repos/git_seminar  master  git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Why is that?

This error appears because we haven't configured the **remote** repository...

Remote & Origin

There are two options to set the remote origin

1. By cloning a remote repository - Git will automatically set the remote origin.

```
Example: git clone url / path to repository
```

Or By Setting Remote Origin

By convention we name new Github (**remote**) repository as Origin and once we run the following
We will resolve the error and git will know what is **remote & origin.**

1. If it's a new local repo that we created - we first run:
   ```
   git  init
   ```
2. Than we can configure the remote origin as followed:
   ```
   git remote add origin url/path
   ```

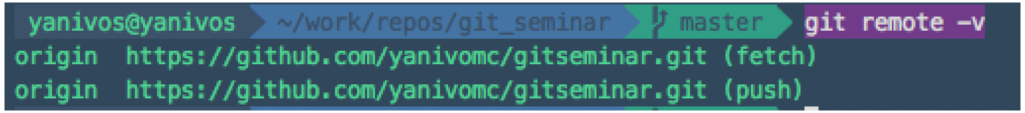
Note: there is no need to call the remote repository origin,
We can call it in whatever name we wish.

GIT LAB 2

So now:

Choose one of the two options to set the remote origin and push your changes

1. Add your remote origin repo
   a. Run `git remote -v` and expect to see



```
 yanivos@yanivos  ~/work/repos/git_seminar  master  git remote -v
origin  https://github.com/yanivomc/gitseminar.git (fetch)
origin  https://github.com/yanivomc/gitseminar.git (push)
```

1. Send your changes to your remote Repo
   `git push origin master`
2. Extra: Create a new branch and Push your changes to it
   `Tip: git push origin branch_name`

GIT LAB 2 CONTINUE

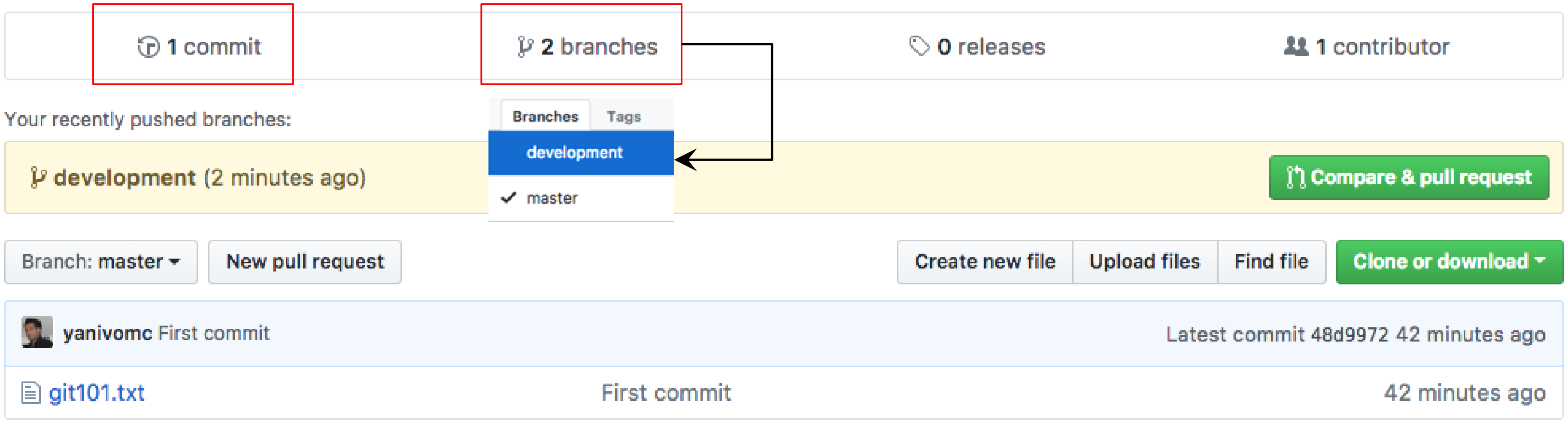
Pushing to a new Branch

1. Creating new branch
   a. git branch branchName - Lets call it "**development**"
   b. git branch - to show the branches we got
   c. git checkout development - to switch branch
2. Update the file we created
   a. Stage , commit and push it
      git add *
      ii. git commit -m "Commit message"
      iii. git push origin development

GIT LAB 2 CONTINUE

Expected results:

At least...



1 commit | 2 branches | 0 releases | 1 contributor

Your recently pushed branches:

development (2 minutes ago)

yanivomc First commit — Latest commit 48d9972 42 minutes ago

git101.txt	First commit	42 minutes ago

MERGING BRANCHES

MERGING BRANCHES

Now that we got two branches -
One for developing a **feature** and the second is our **master** production branch.
In most products life cycle, eventually we would like to release and merge the new feature developed in our **development** branch into our master branch which is used (in most cases) as our production deployment branch.

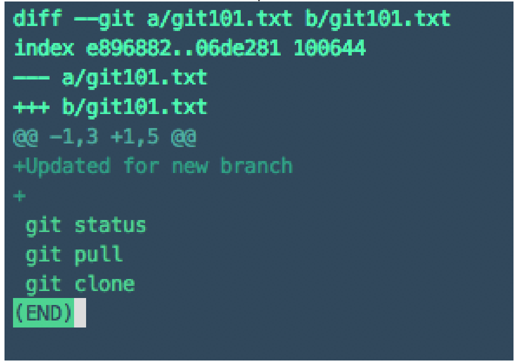
This operation is called - **Merging branches**.

HANDS ON

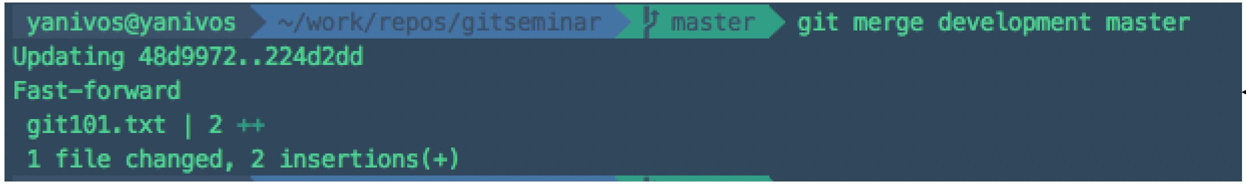
GIT MERGING BRANCHES

Instruction

1. Check out the target branch (Master in our case)
   a. git checkout master
2. Diff the two files in master and in **development** branch
   a. git diff “target branch that we want to merge into” “source branch”
3. Run git merge command
   a. git merge “source branch” “target branch to merge into”



```
diff --git a/git101.txt b/git101.txt
index e896882..06de281 100644
--- a/git101.txt
+++ b/git101.txt
@@ -1,3 +1,5 @@
+Updated for new branch
+
 git status
 git pull
 git clone
(END)
```



```
 yanivos@yanivos  ~/work/repos/gitseminar   master  git merge development master
Updating 48d9972..224d2dd
Fast-forward
 git101.txt | 2 ++
 1 file changed, 2 insertions(+)
```

MERGING CONFLICTS

GIT BRANCH CONFLICTS

Intro

Git sole purpose is to allow multiple team members to work at the same time on same features, branches etc…

But what will happen if we try to merge our development branch into our master branch when one of our team members updated our master branch and someone else changed/updated the development branch?

GIT BRANCH CONFLICTS

Let's simulate it

GIT BRANCH CONFLICTS

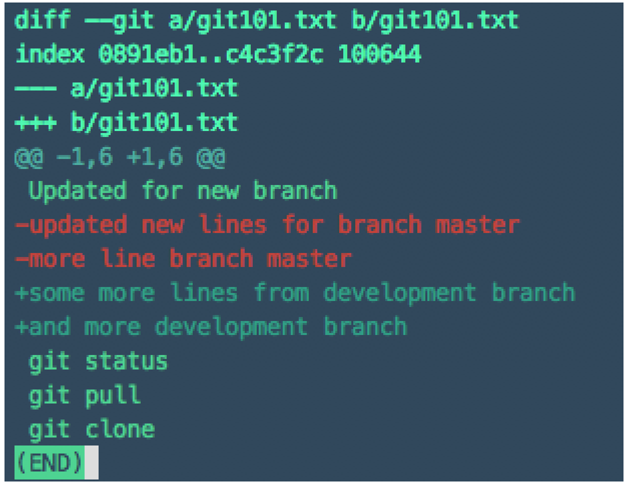
Instruction

1. Make sure both master and development branches are aligned
   a. git diff master development
2. Check out development branch
   a. git checkout development
   b. Add new lines into git101.txt file
   c. Stage, commit and push it
3. Now checkout master branch and update git101.txt file with some other newlines
   a. Stage, commit and push it
4. Diff the two files in master and in development branch
5. Run git merge command
   a. git merge development master

GIT BRANCH CONFLICTS

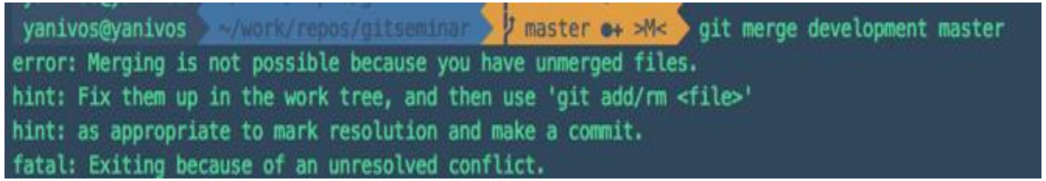
Expected outcome

DIFF



```
diff --git a/git101.txt b/git101.txt
index 0891eb1..c4c3f2c 100644
--- a/git101.txt
+++ b/git101.txt
@@ -1,6 +1,6 @@
 Updated for new branch
-updated new lines for branch master
-more line branch master
+some more lines from development branch
+and more development branch
 git status
 git pull
 git clone
(END)
```

MERGE



```
yanivos@yanivos  ~/work/repos/gitseminar  master ●+ >M<  git merge development master
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.
```

RESOLVING CONFLICTS

To resolve a conflict - All collaborators working on the new branch (development) has to switch to master branch and then pull all the latest code from repository master branch

1. `git checkout master`
2. `git pull origin master`

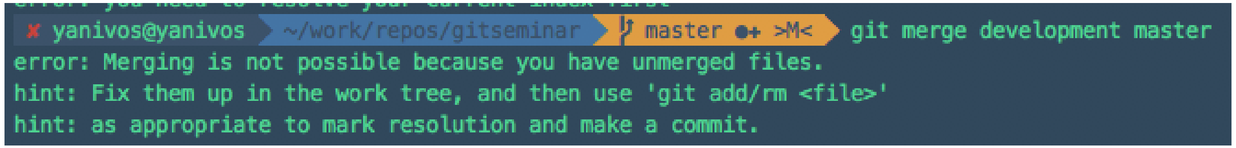
Than we switch back to development branch and **merge** master into our development branch

1. `git checkout development`
2. `git merge development master`

GIT BRANCH CONFLICTS

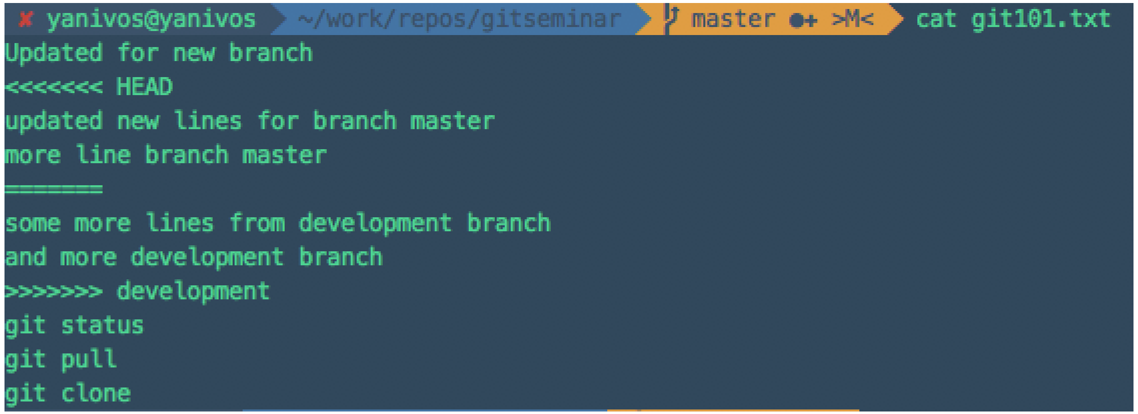
Expected outcome

MERGE



```
error: you need to resolve your current index first
✘ yanivos@yanivos  ~/work/repos/gitseminar  master ●+ >M<  git merge development master
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
```

FILE

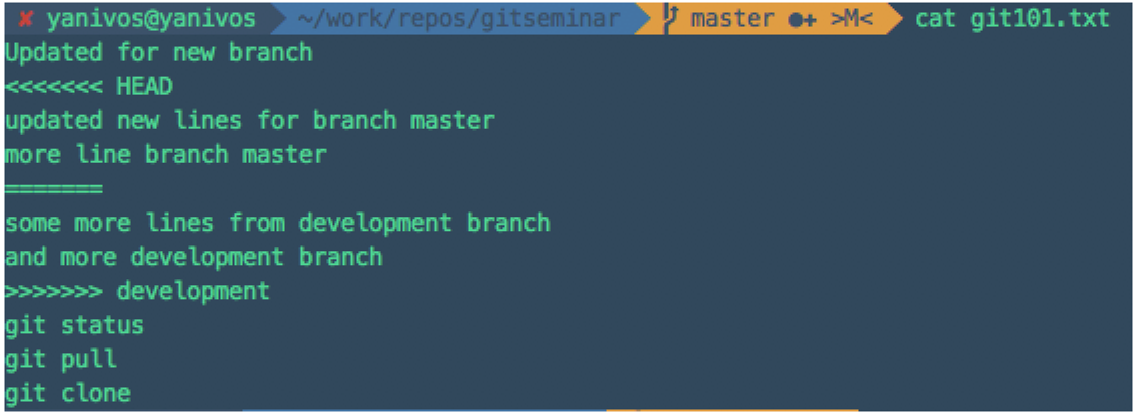


```
✘ yanivos@yanivos  ~/work/repos/gitseminar  master ●+ >M<  cat git101.txt
Updated for new branch
<<<<<<< HEAD
updated new lines for branch master
more line branch master
=======
some more lines from development branch
and more development branch
>>>>>>> development
git status
git pull
git clone
```

RESOLVING CONFLICTS

Now we manually edit the file - notice “HEAD”, “DEVELOPMENT”

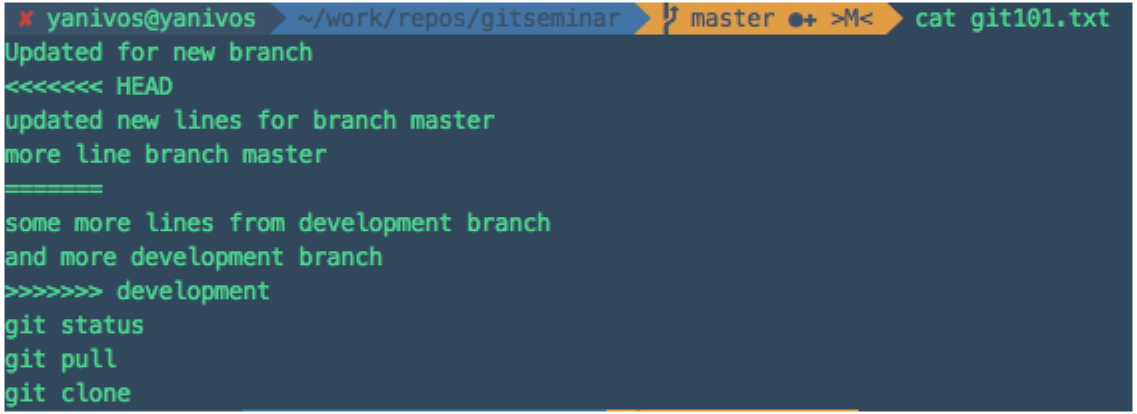
That shows us what lines cause the conflict.



```
✘ yanivos@yanivos  ~/work/repos/gitseminar  master ●+ >M<  cat git101.txt
Updated for new branch
<<<<<<< HEAD
updated new lines for branch master
more line branch master
=======
some more lines from development branch
and more development branch
>>>>>>> development
git status
git pull
git clone
```

RESOLVING CONFLICTS

Once we resolved the conflict by deleting those lines or leaving them intact.
We can again stg , commit and push them and than merge it to master branch



```
✘ yanivos@yanivos  ~/work/repos/gitseminar   master ●+ >M<  cat git101.txt
Updated for new branch
<<<<<<< HEAD
updated new lines for branch master
more line branch master
=======
some more lines from development branch
and more development branch
>>>>>>> development
git status
git pull
git clone
```

PULL REQUEST / MERGE PULL REQUEST

GIT PULL REQUEST

EXPLAINED

Merge a pull request is when we wish to merge into the upstream branch or a fork that a work on it is completed (Recall that branches are usually short live) we are doing pull request to allow others know about the changes you've pushed to a GitHub repository (even for code review). Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary before a merge take place.

Anyone with push access to the repository can complete the merge.

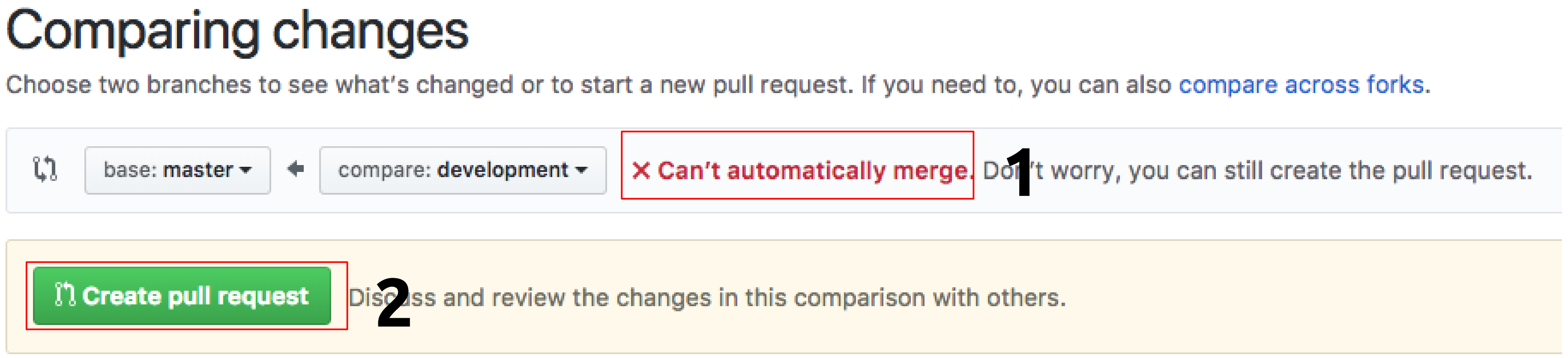
Pull Request is being done in Github

GIT PULL REQUEST

JOHN BRYCE
Leading in IT Education
matrix company

EXPLAINED

There are scenarios where merge due to conflicts as mentioned before is not possible and clicking Pull request for merging automatically won't be possible



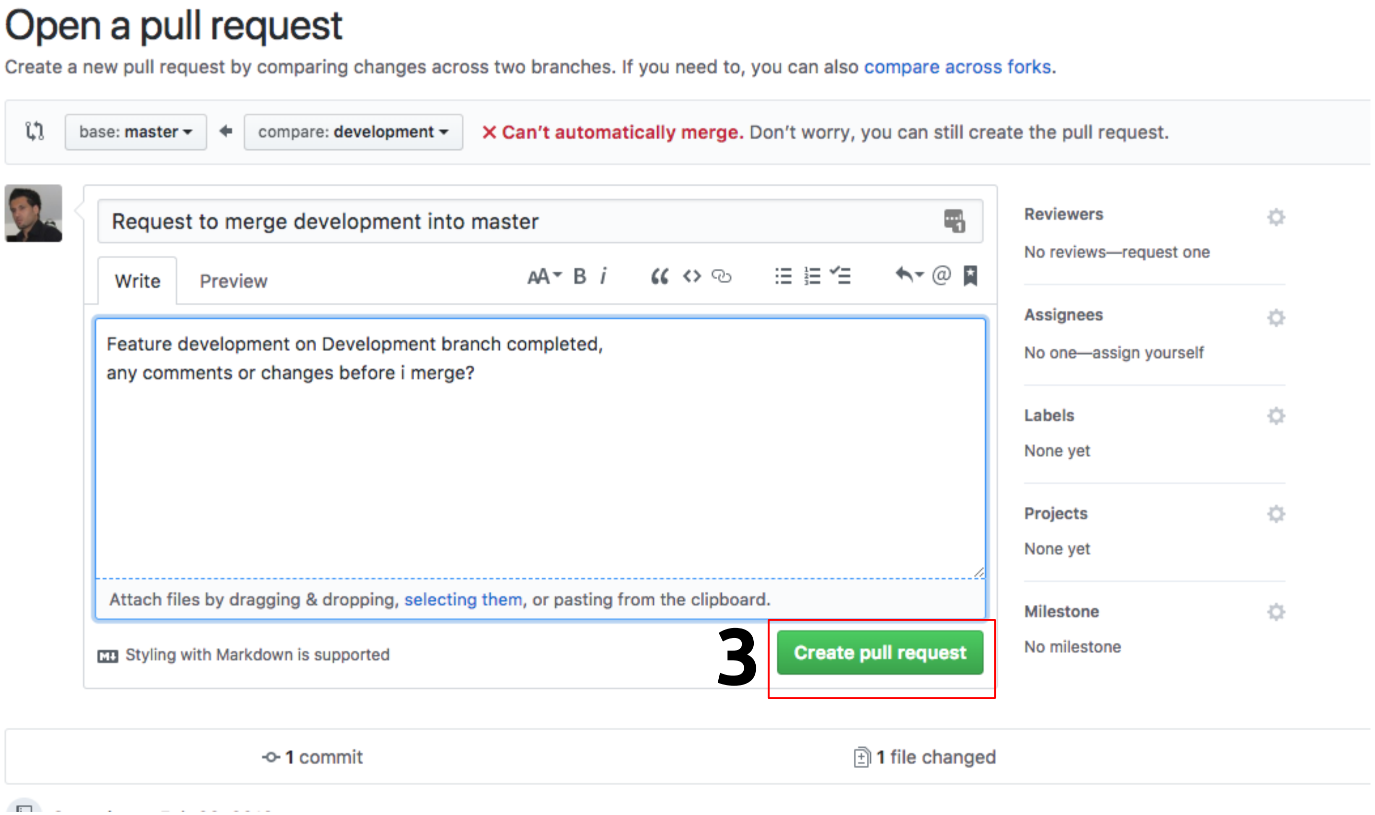
Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

× Can't automatically merge. Don't worry, you can still create the pull request.

Discuss and review the changes in this comparison with others.

GIT PULL REQUEST



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: master ← compare: development ✕ Can't automatically merge. Don't worry, you can still create the pull request.

Request to merge development into master

Feature development on Development branch completed,
any comments or changes before i merge?

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

3

Reviewers
No reviews—request one

Assignees
No one—assign yourself

Labels
None yet

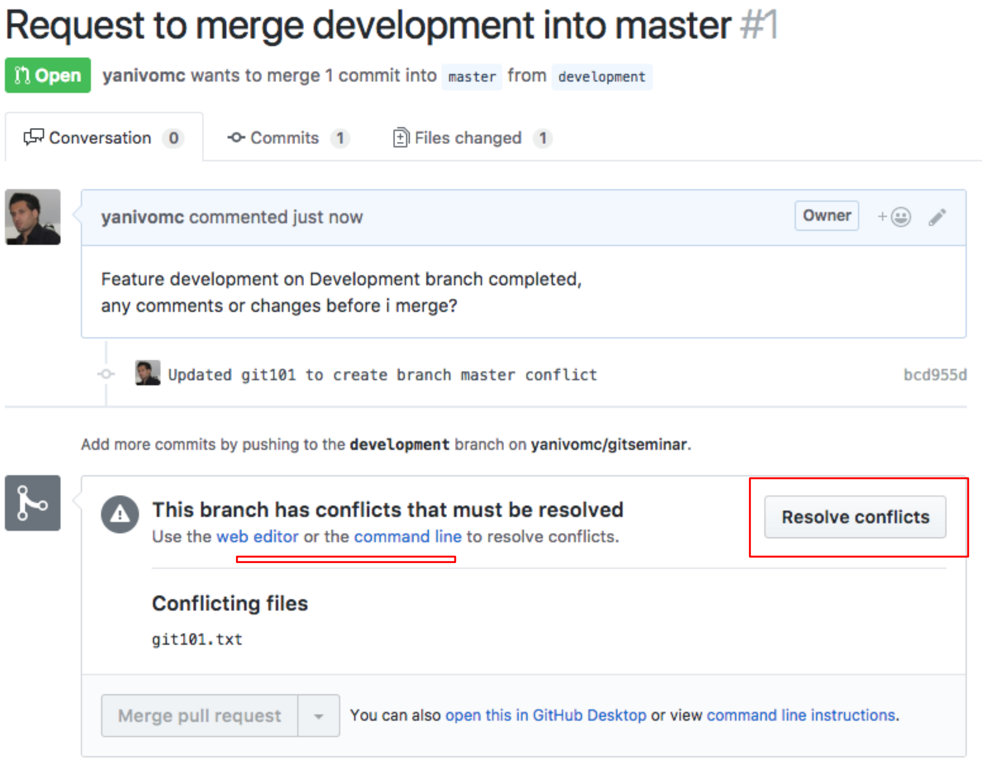
Projects
None yet

Milestone
No milestone

1 commit

1 file changed

GIT PULL REQUEST



Request to merge development into master #1

Open yanivomc wants to merge 1 commit into `master` from `development`

yanivomc commented just now

Feature development on Development branch completed,
any comments or changes before i merge?

```
Updated git101 to create branch master conflict    bcd955d
```

Add more commits by pushing to the **development** branch on **yanivomc/gitseminar**.

This branch has conflicts that must be resolved
Use the web editor or the command line to resolve conflicts.

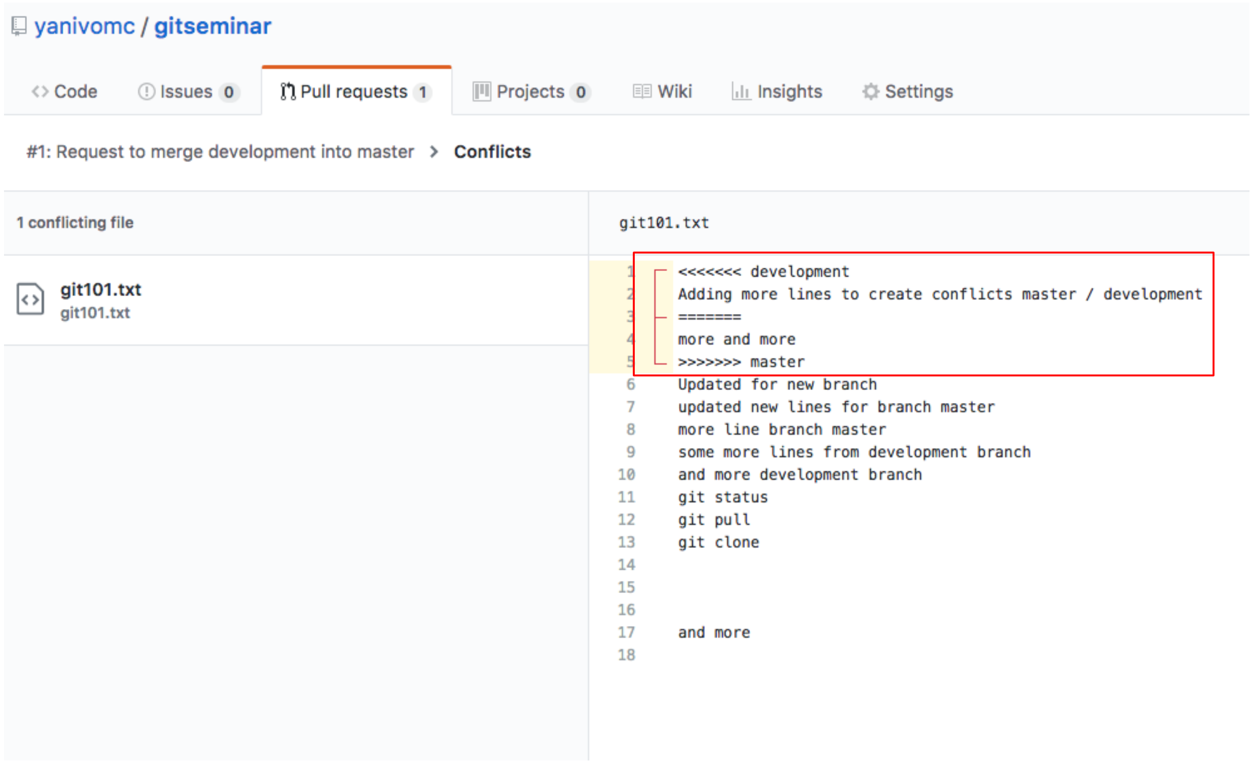
Conflicting files

```
git101.txt
```

You can also open this in GitHub Desktop or view command line instructions.

4

GIT PULL REQUEST


yanivomc / gitseminar

#1: Request to merge development into master > **Conflicts**

1 conflicting file

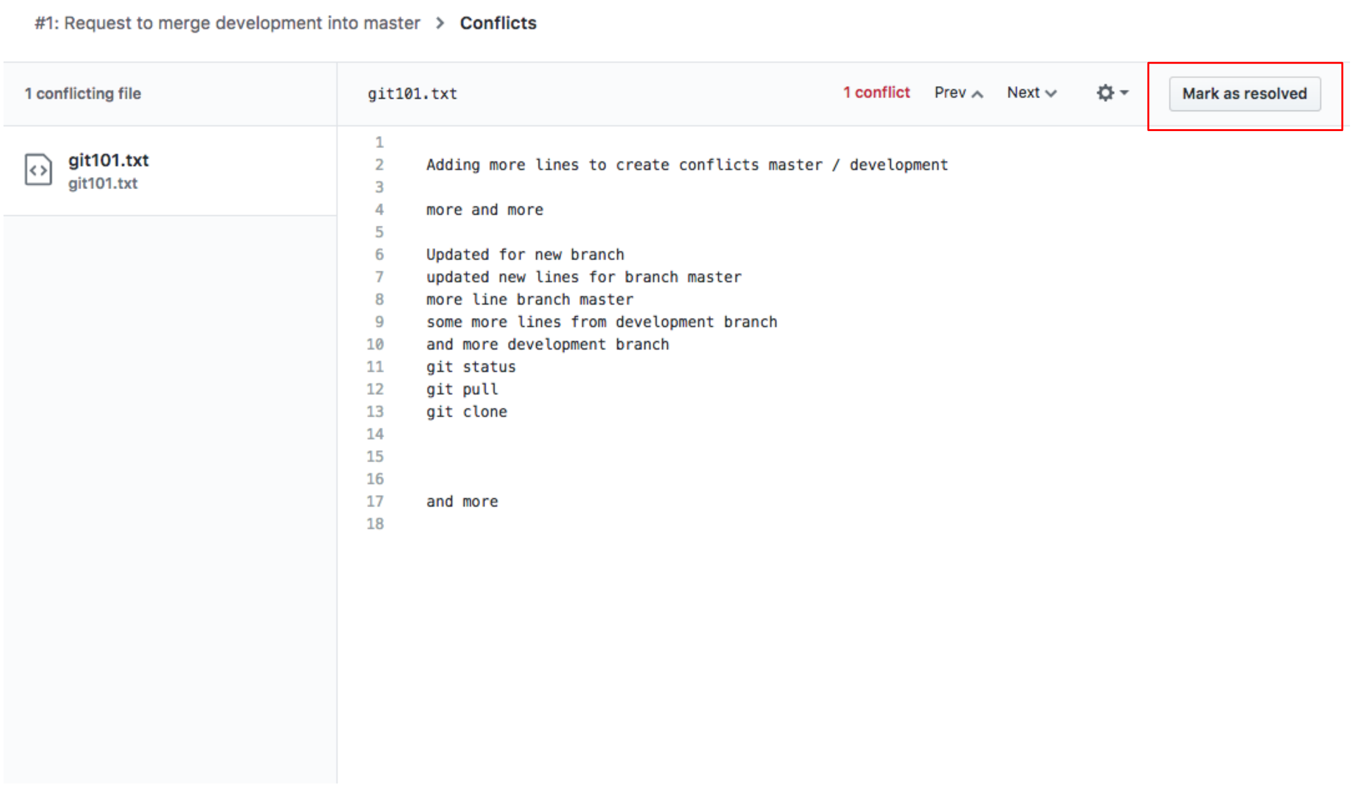
git101.txt
git101.txt

git101.txt

```
<<<<<<< development
Adding more lines to create conflicts master / development
=======
more and more
>>>>>>> master
Updated for new branch
updated new lines for branch master
more line branch master
some more lines from development branch
and more development branch
git status
git pull
git clone



and more

```

5

GIT PULL REQUEST



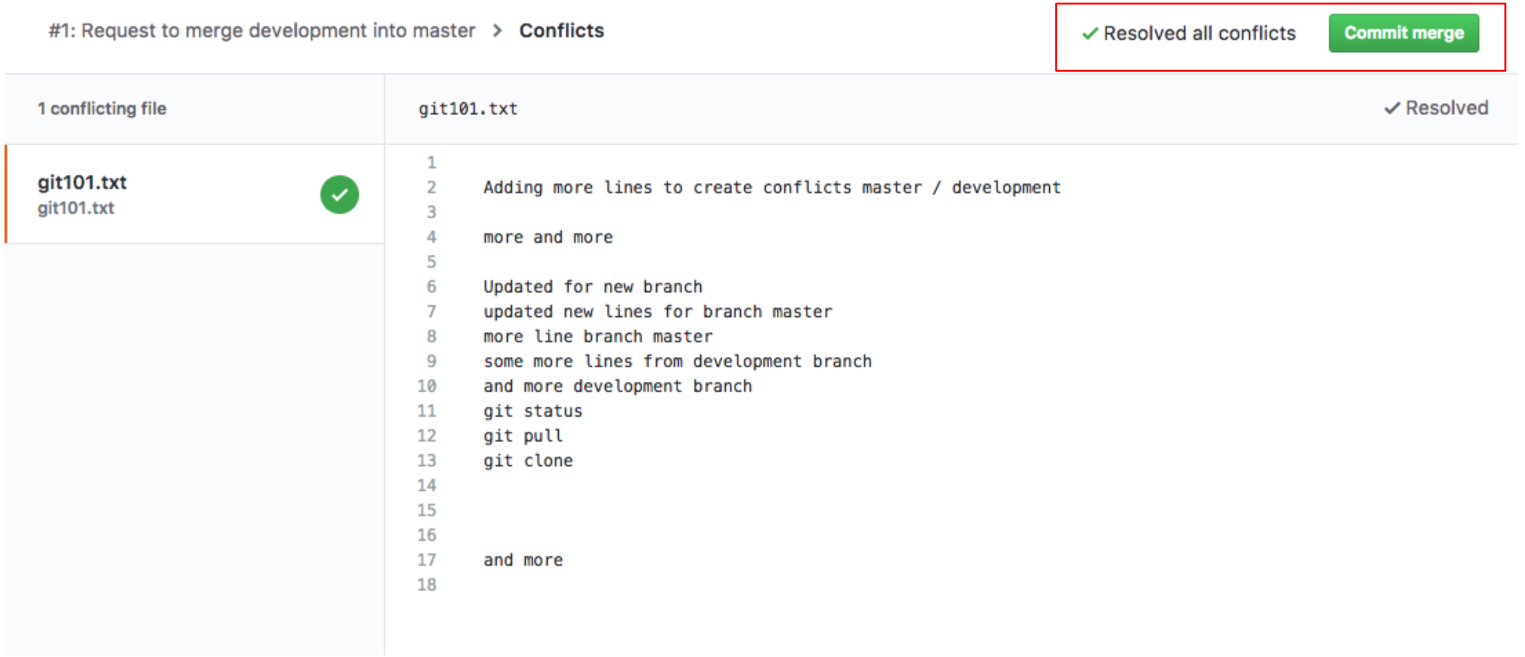
#1: Request to merge development into master > **Conflicts**

git101.txt

```

Adding more lines to create conflicts master / development

more and more

Updated for new branch
updated new lines for branch master
more line branch master
some more lines from development branch
and more development branch
git status
git pull
git clone



and more

```

6

GIT PULL REQUEST

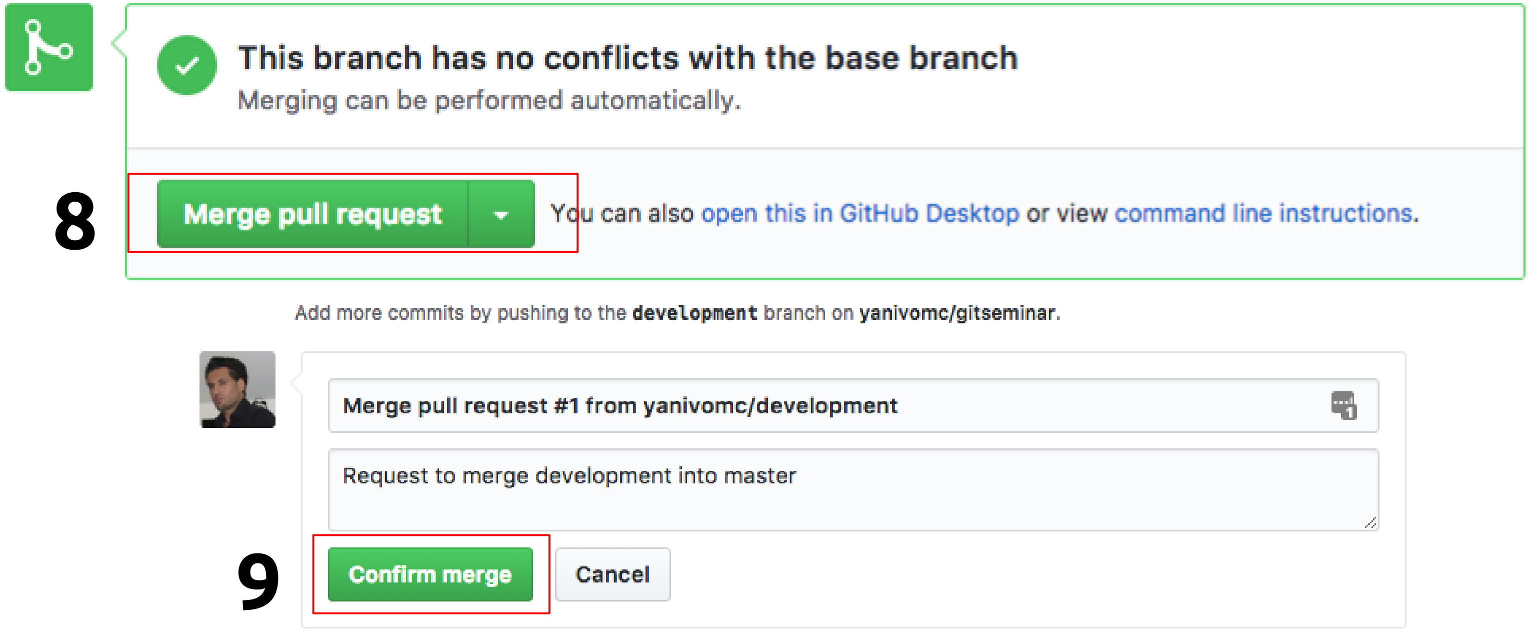

#1: Request to merge development into master > **Conflicts**

git101.txt

```

Adding more lines to create conflicts master / development

more and more

Updated for new branch
updated new lines for branch master
more line branch master
some more lines from development branch
and more development branch
git status
git pull
git clone



and more

```

7

GIT PULL REQUEST


8

9

This branch has no conflicts with the base branch

Merging can be performed automatically.

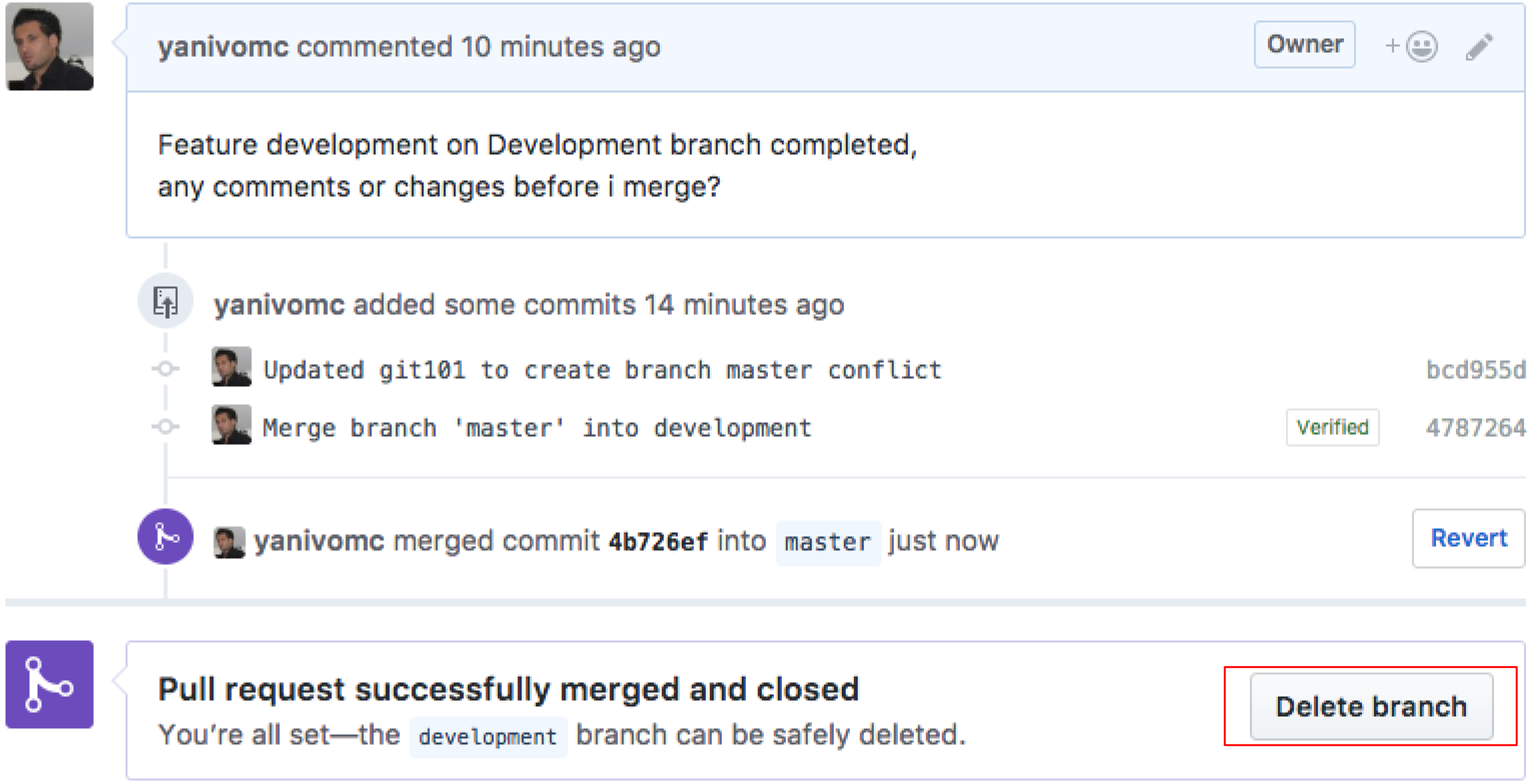
You can also open this in GitHub Desktop or view command line instructions.

Add more commits by pushing to the **development** branch on **yanivomc/gitseminar**.

Merge pull request #1 from yanivomc/development

Request to merge development into master

GIT PULL REQUEST



yanivomc commented 10 minutes ago

Feature development on Development branch completed,
any comments or changes before i merge?

yanivomc added some commits 14 minutes ago

```
Updated git101 to create branch master conflict          bcd955d
Merge branch 'master' into development      Verified     4787264
```

yanivomc merged commit **4b726ef** into `master` just now

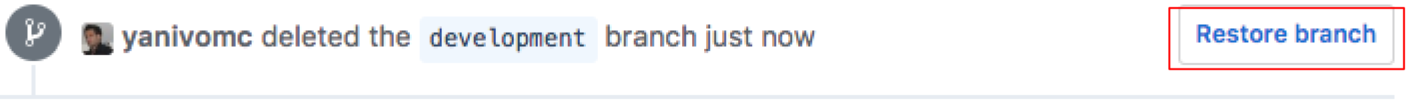
Pull request successfully merged and closed

You're all set—the `development` branch can be safely deleted.

10

GIT PULL REQUEST

DELETE BRANCH


yanivomc deleted the `development` branch just now

11