

Introducing



Tiago Vale
21 February 2013

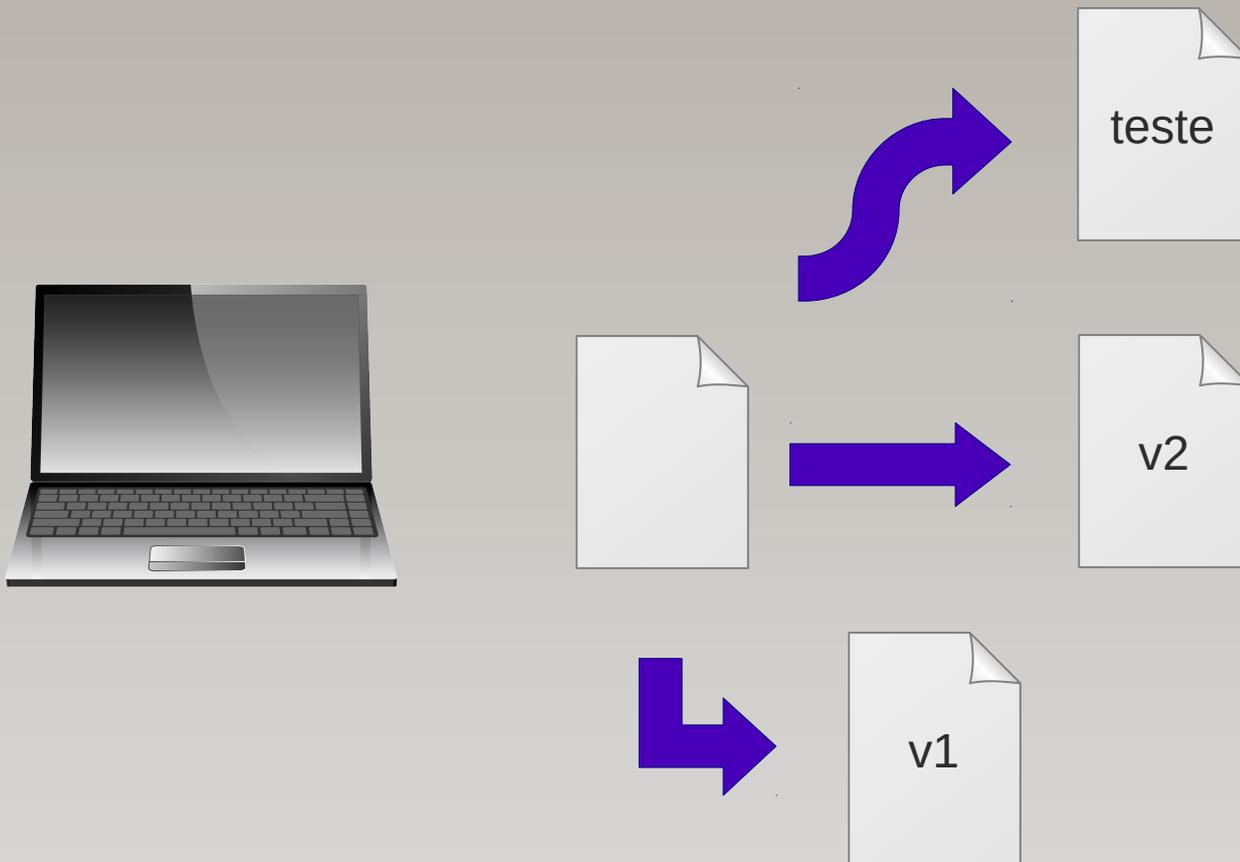
Outline

- **What** (is git)?
- **Why** (do we need git)?
- **Who** uses git?
- **Concepts and hands on**

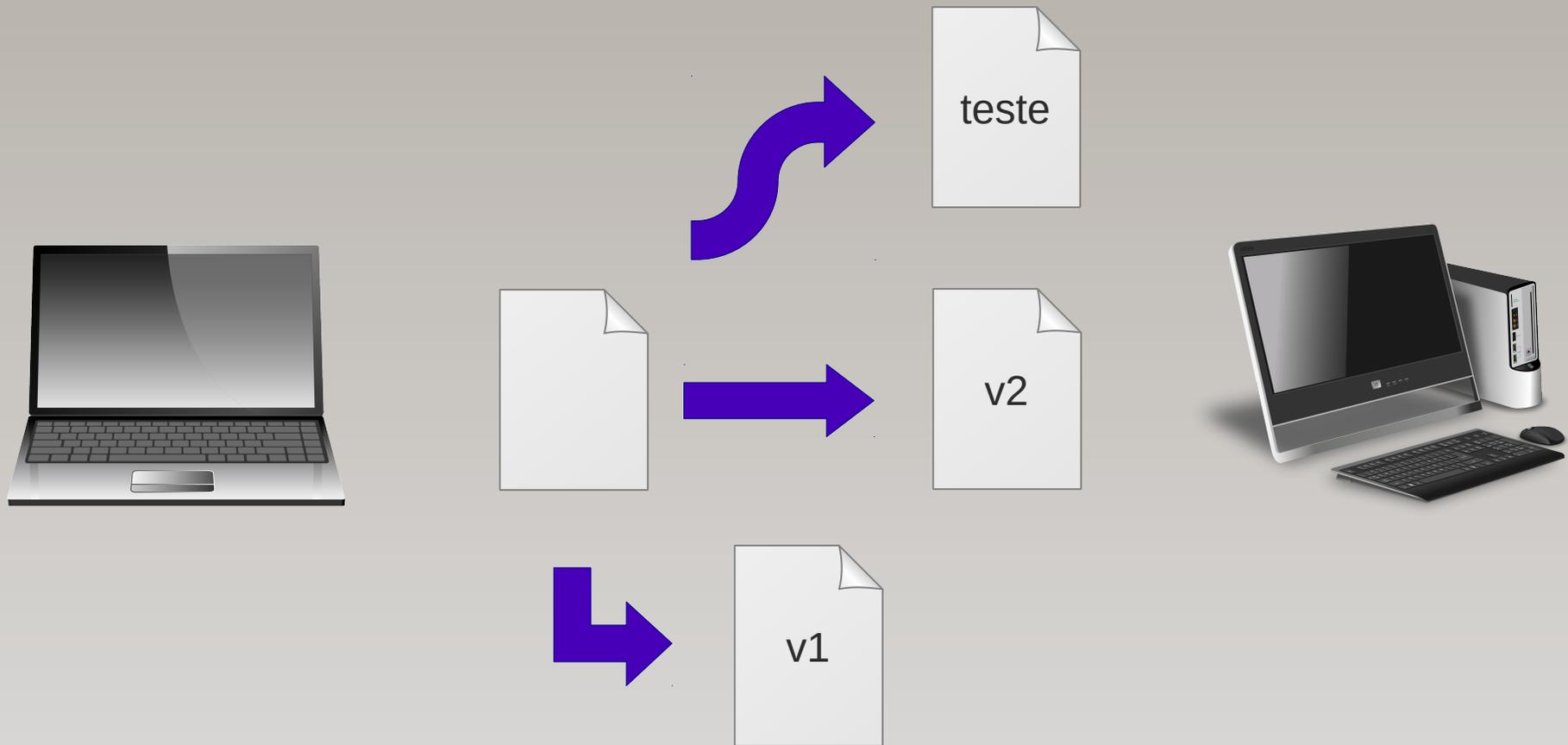
What (is git)?

- (Distributed) version control system;
- **Free** and open source;

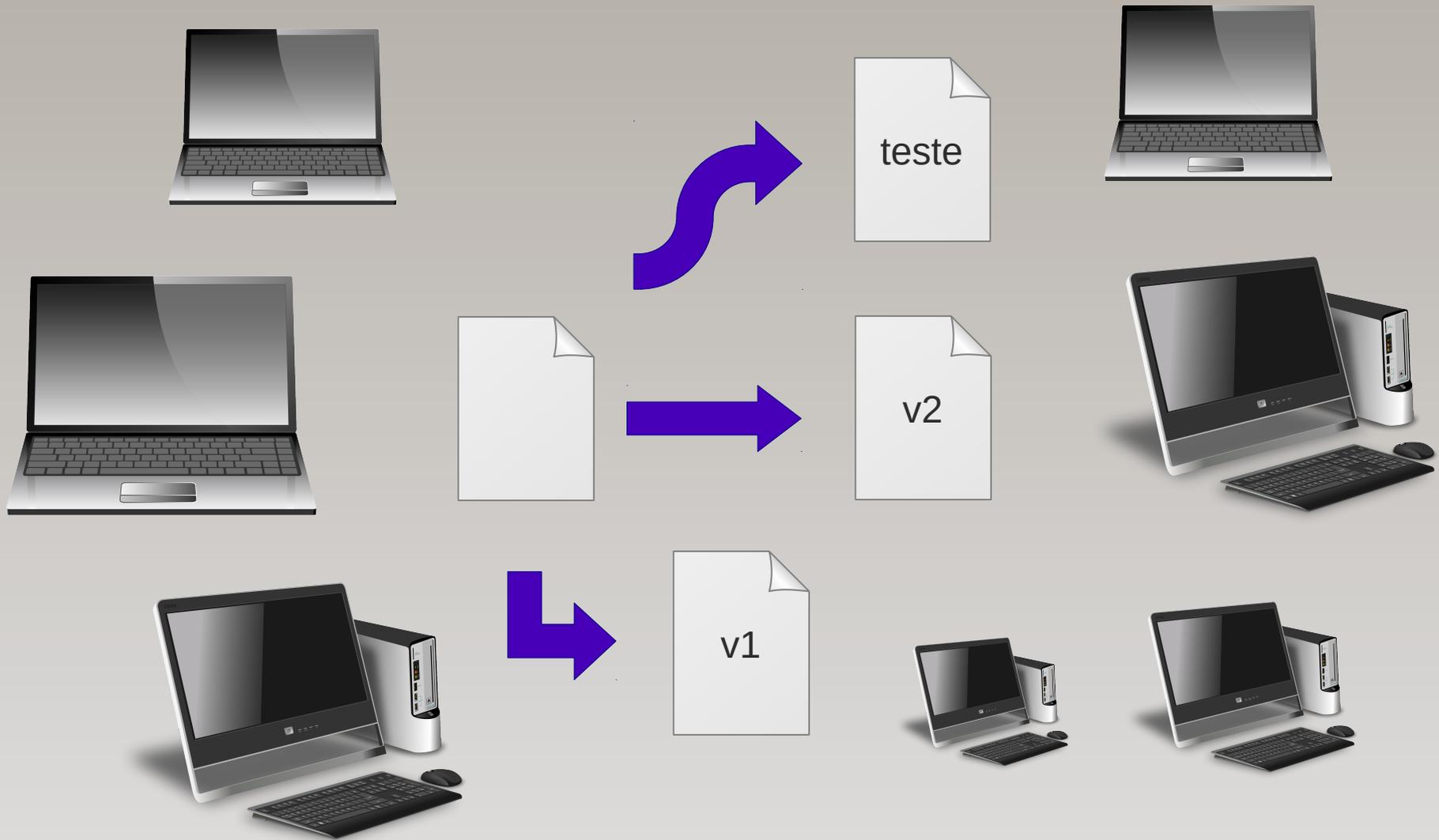
Why (do we need git)?



Why (do we need git)?

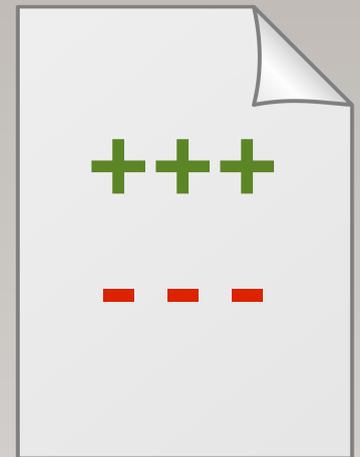


Why (do we need git)?



Why (do we need git)?

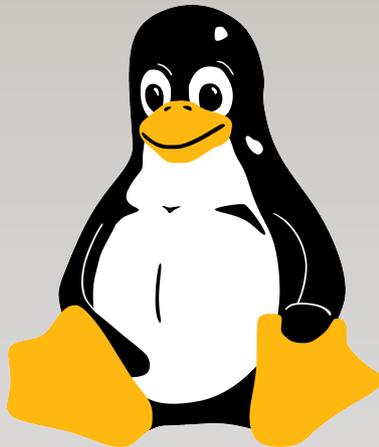
- What was changed?
- When?
- By whom?
- Why?



Who uses git?

facebook®

Google *Microsoft*®

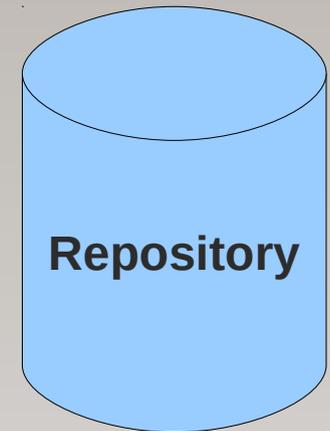


ANDROID

Linked **in**

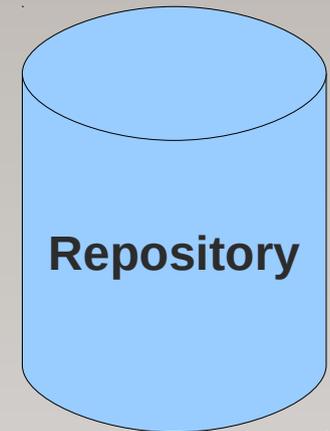
Concepts – Repository

- Where all our work is stored;
- Contains every version of our work;
- Can be shared.



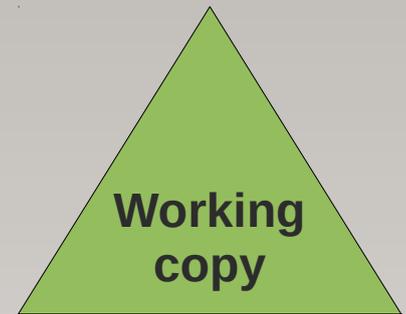
Hands On – Repository

- Okay, so let's create our repository;
- Create/change to some directory:
 - e.g., gitworkshop;
- `$ git init`
 - Command to create a repository!



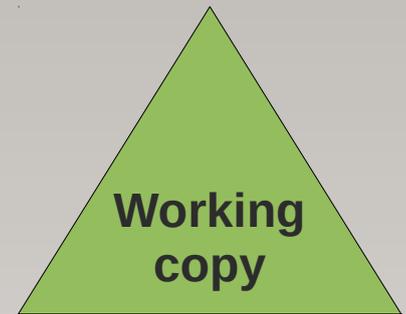
Concepts – Working Copy

- A snapshot of the repository;
- Where we work, i.e., change things;
- Private.



Hands On – Working Copy

- `$ git status`
 - Check the state of our working copy.

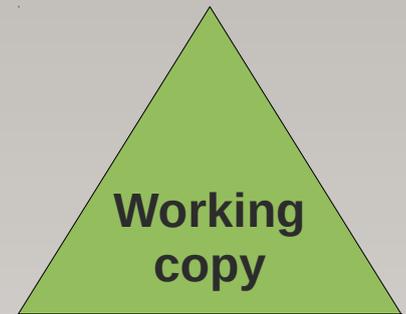


nothing to commit, working directory clean

Hands On – Working Copy

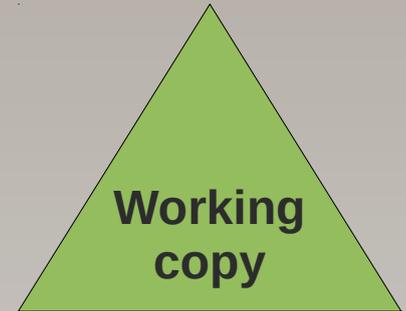
- Let's create our first file;
- Create the file `main.c`:

```
#include <stdio.h>
int main(int argc, char** argv)
{
    printf("Hello world\n");
    return 0;
}
```



Hands On – Working Copy

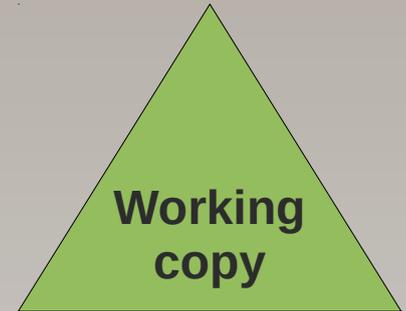
- `$ git status`
 - We created a new file.



```
# Untracked files:
#   (use "git add <file>..." to include in what will be
committed)
#
#   main.c
nothing added to commit but untracked files present
(use "git add" to track)
```

Hands On – Working Copy

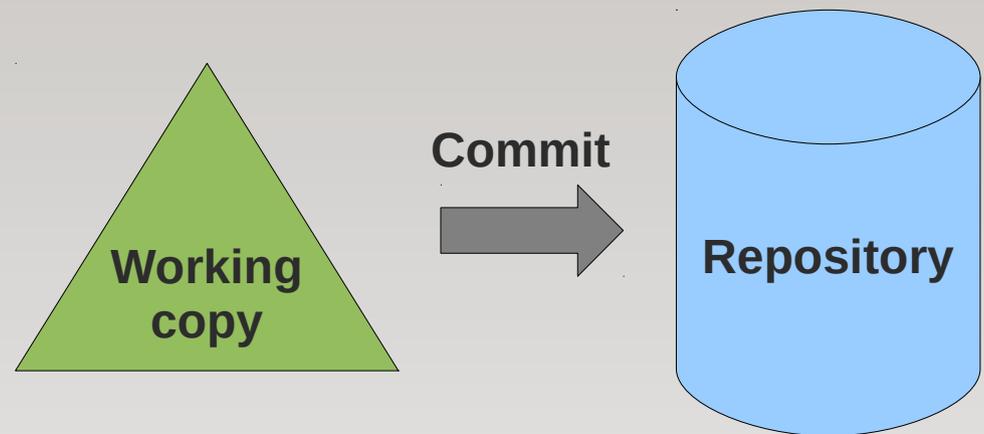
- `$ git add main.c`
 - We want git to maintain `main.c`.



```
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
# new file:   main.c
```

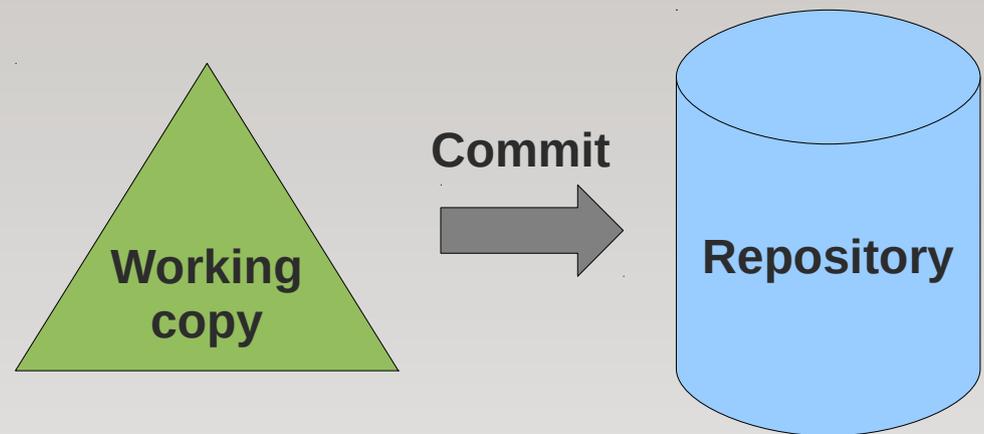
Concepts – Commit

- Operation that modifies the repository;
- Typically accompanied by a comment that:
 - explains the changes made;
 - becomes part of the repository's history.



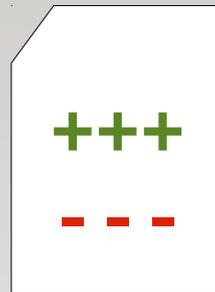
Hands On – Commit

- `$ git commit --message "My first commit!"`
- `--message/ -m` flag specifies the commit comment.



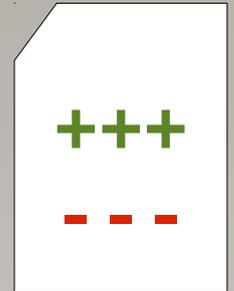
Concepts – Log

- History of the repository's evolution.
- Description of the modifications made, who made them, and when.



Hands On – Log

- `$ git log`
 - Summary containing the author, date and comment;
- `$ git show`
 - More detailed description, includes actual changes.

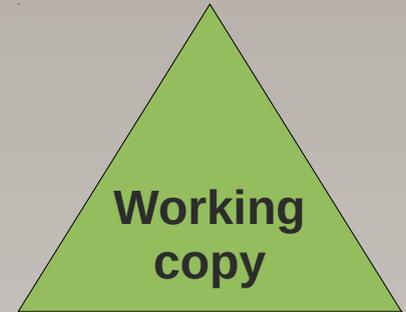


```
commit 9c5c46b64c9470bf0e87cc63421cff23a226023b
Author: Tiago Vale <tiagomarquesvale@gmail.com>
Date:   Wed Feb 20 21:22:57 2013 +0000
```

```
    My first commit
```

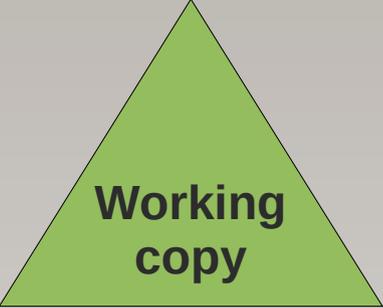
Hands On

- Let's modify `main.c`.
- And now, we want to commit.



```
#include <stdio.h>
int main(int argc, char** argv)
{
    printf("Hello world, how are you?\n");
    return 0;
}
```

Hands On

- `$ git commit -m "Modified string"`
- What happened? :-(


```
# 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:   main.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```

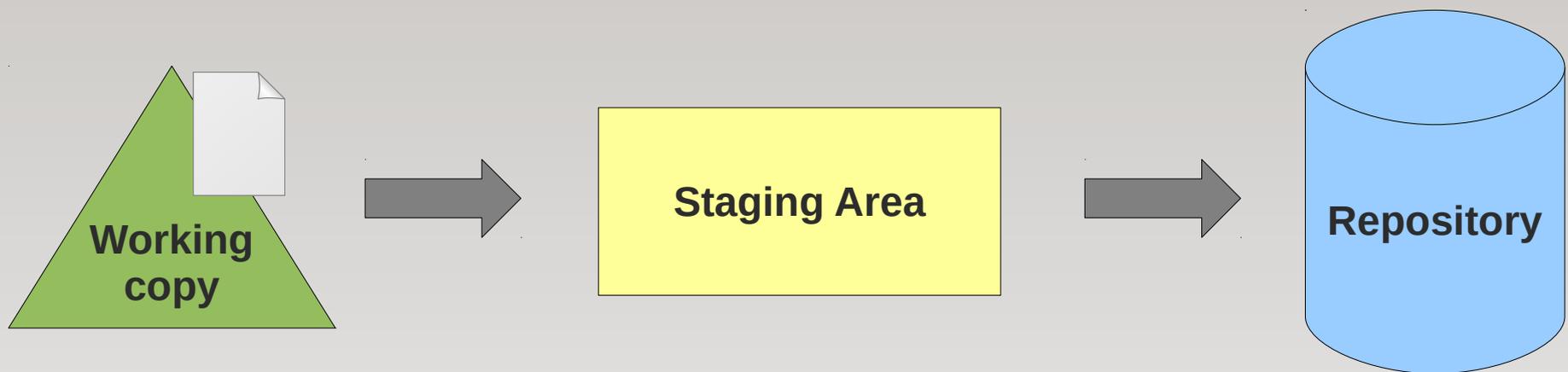
Concepts – Staging Area

- Sort of a loading dock;
- When we commit, we only apply the changes to files which are staged.



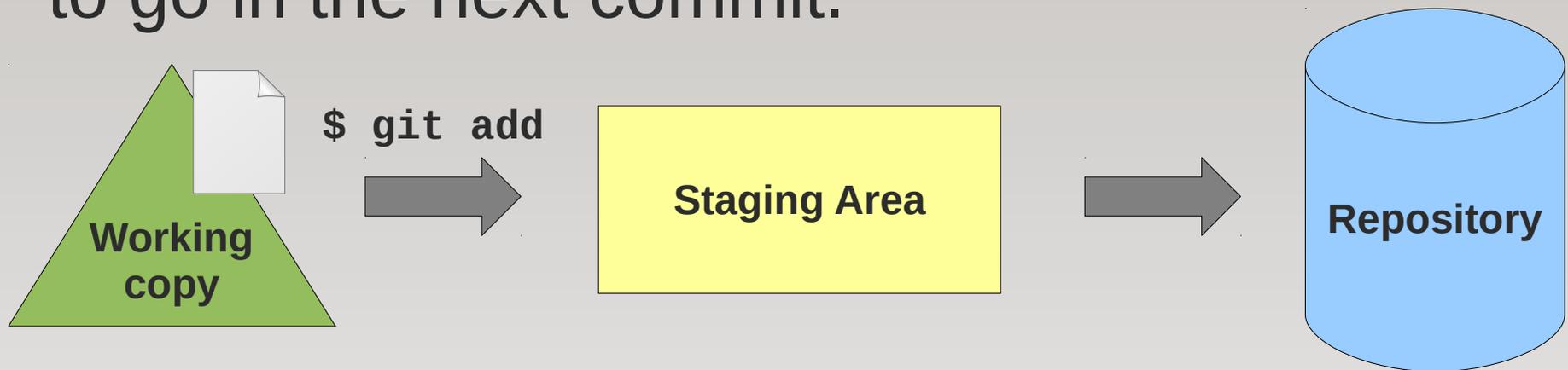
Concepts – Staging Area

- When we modify a file in the working copy, it is marked as “modified.”



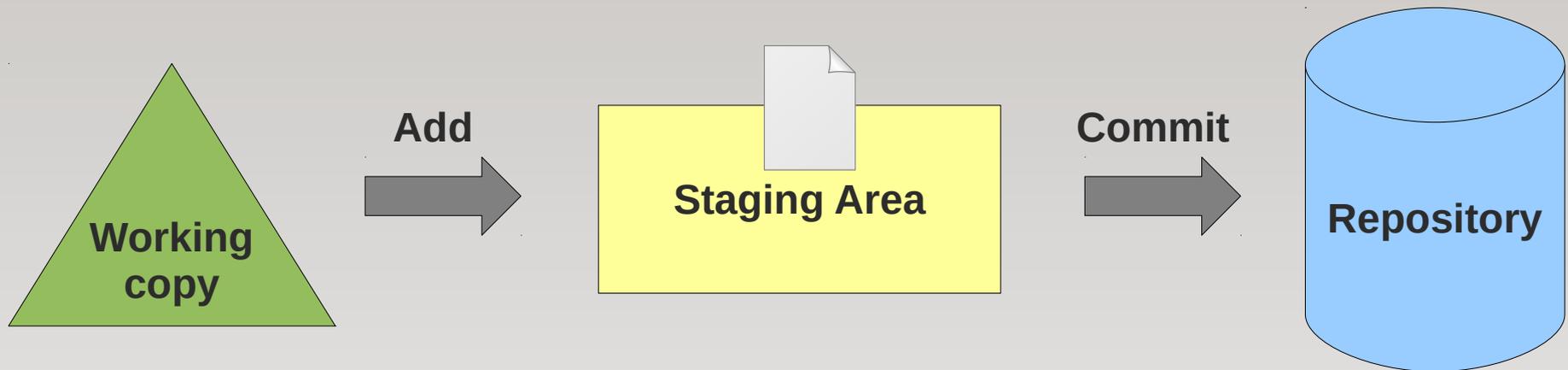
Concepts – Staging Area

- Before committing, the modified file needs to be “staged”
 - i.e., add a snapshot of it to the staging area;
- Modified data is marked in its current version to go in the next commit.



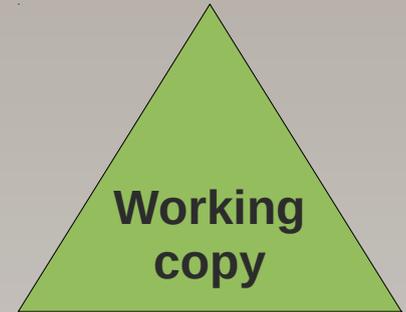
Concepts – Staging Area

- Staged changes can be committed!



Hands On – Let's Try Again

- `$ git add main.c`

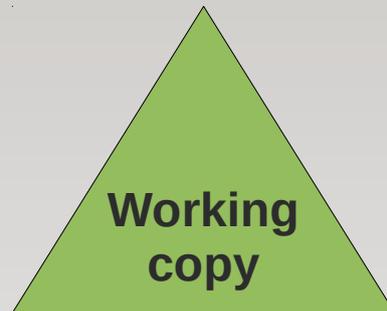


- `$ git commit -m "Modified string"`

```
[master cd9b797] Modified string
1 file changed, 1 insertion(+), 1 deletion(-)
```

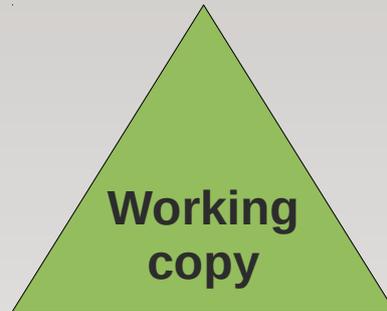
Hands On – Let's Try Again

- The staging area can be bypassed with the `--all/-a` commit flag;
 - Commits all changed files in the working copy.
- `$ git commit --all --message "..."`



Hands On

- What if I modify something, and change my mind? How do I discard the changes?
 - i.e., revert to the current repository version.
- `$ git checkout -- <file>`
 - e.g., `$ git checkout -- main.c`

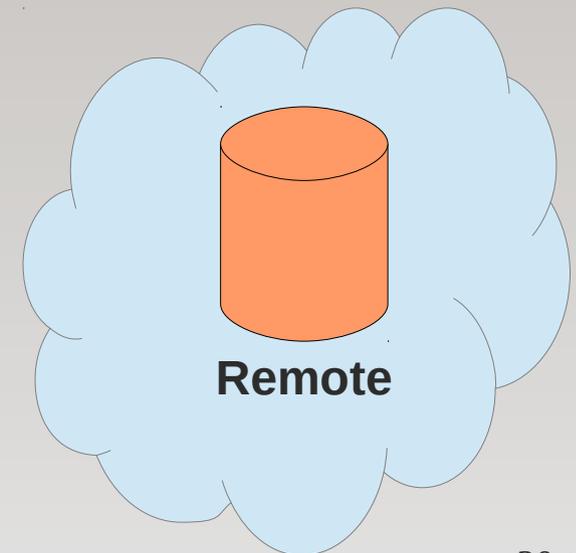
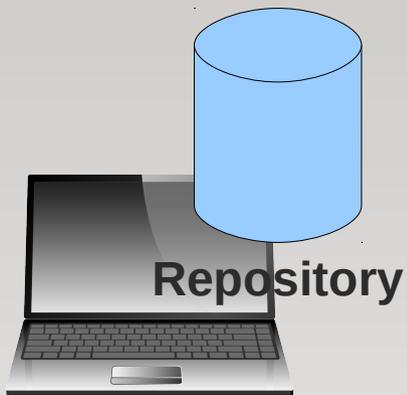


Commands, Revisited

- `$ git init`
- `$ git status`
- `$ git add <file>`
- `$ git diff <file>`
- `$ git commit [-a] -m <message>`
- `$ git checkout -- <file>`
- `$ git reset HEAD <file>`
- `$ git log/show`

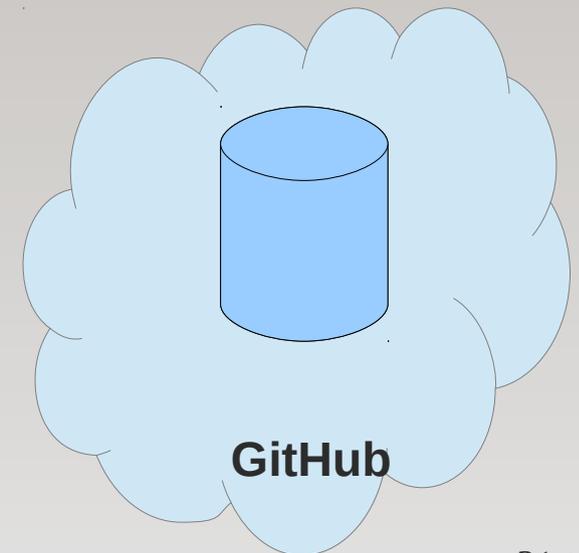
Concepts – Remote

- Other instance of this repository...
- ...on other computer!
 - e.g., on GitHub.



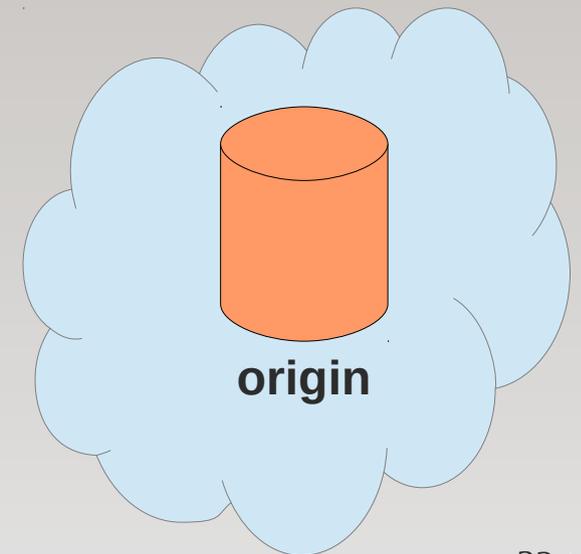
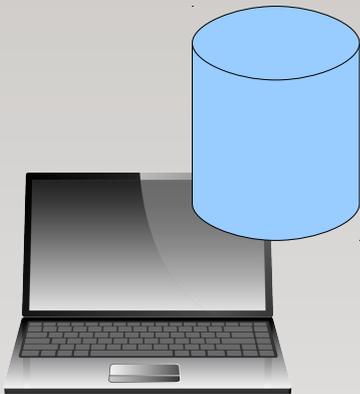
Hands On – Remote

- Create a repository on GitHub;
- Now, we *clone* GitHub's repository to our machine
 - `$ git clone https://github.com/<user>/<repo>.git`



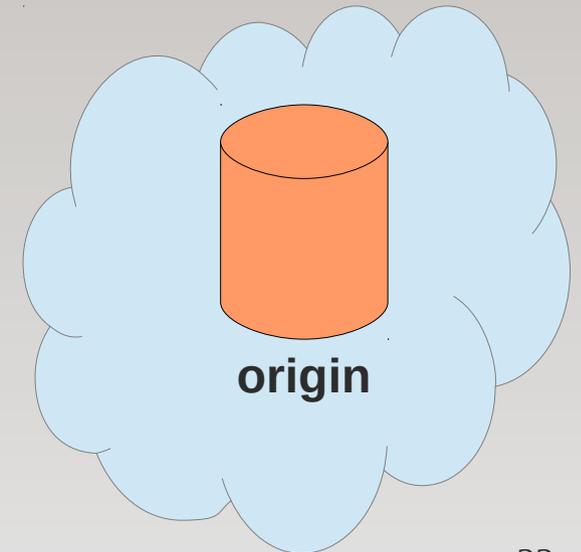
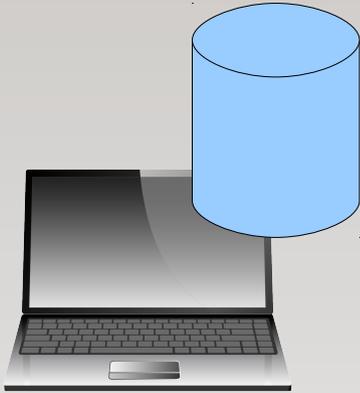
Hands On – Remote

- We now have our local instance of GitHub's repository;
- We always work on *our local instance*;
- The repository on GitHub is called *origin*.
- `$ git remote`



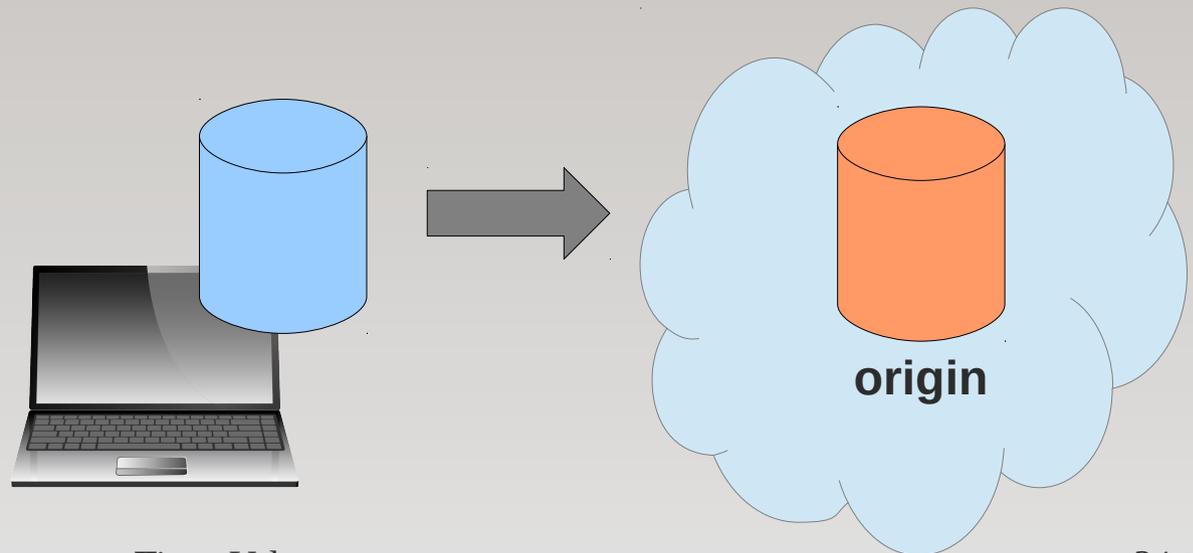
Hands On – Remote

- Let's put our `main.c` file in this repository;
- Check GitHub.
- Not there! :-)



Concepts – Push

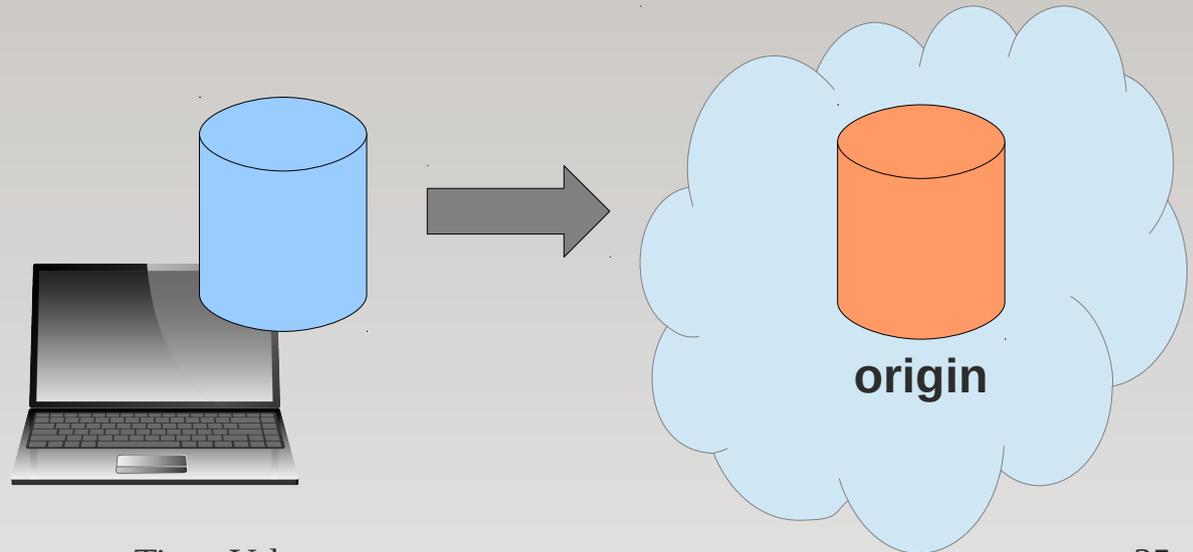
- Copy changes from the local repository instance to a remote one;
- Synchronization between two repository instances.



Hands On – Push

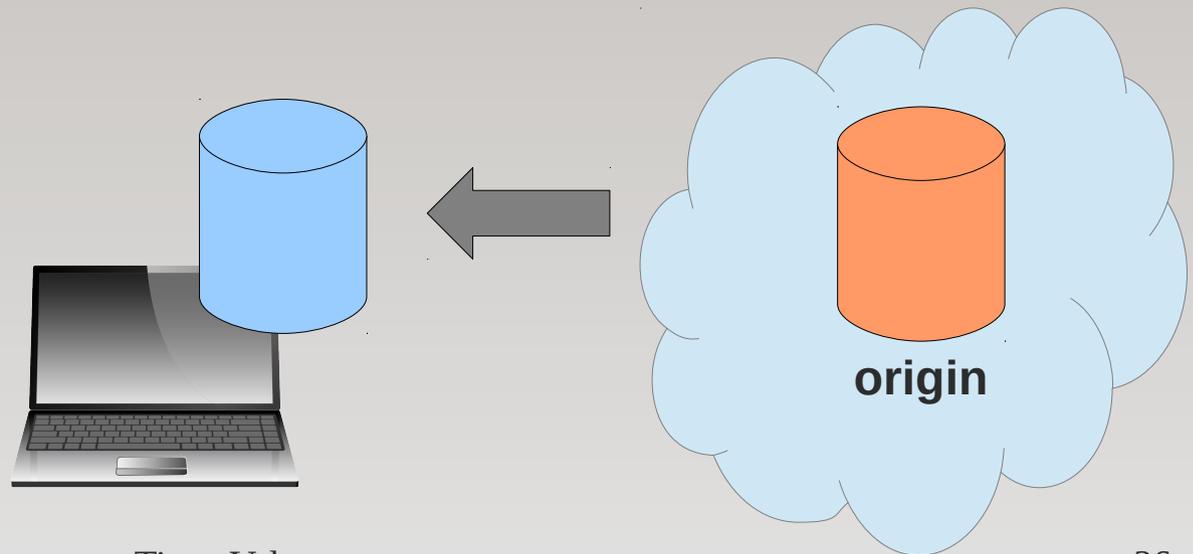
- We want to push our work to keep GitHub's repository up to date.
- `$ git push origin`

```
To https://github.com/tvale/git-workshop.git  
1c95c92..6284ab4  master -> master
```



Concepts – Pull

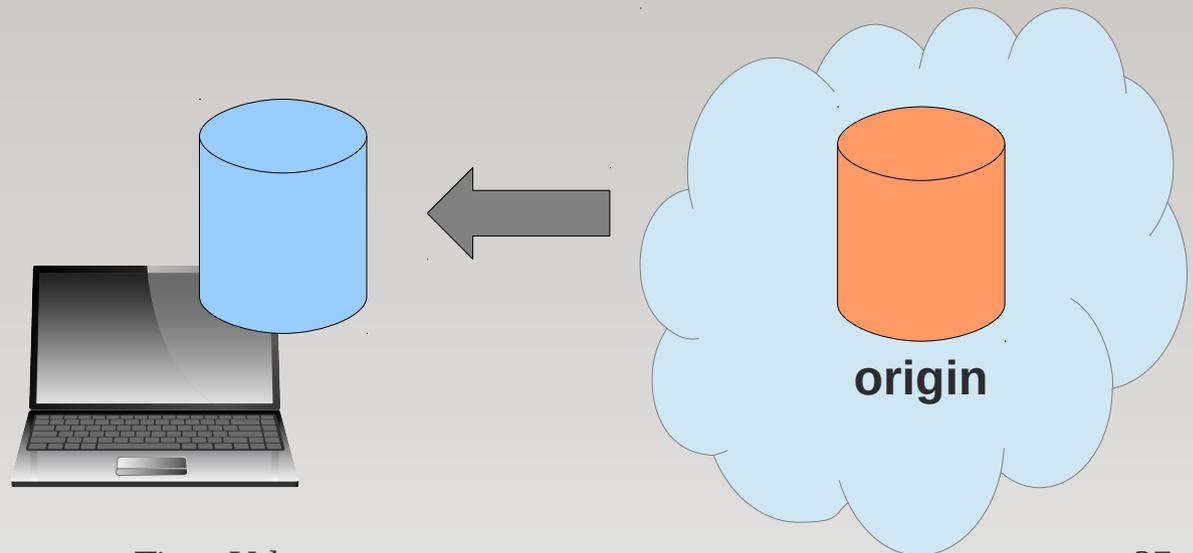
- Copy changes from a remote repository instance to the local one;
- The other way around!



Hands On – Pull

- Update your local repository instance.
- `$ git pull origin`

```
main.c | 2 +-  
1 file changed, 1 insertion(+), 1 deletion(-)
```



Hands On

- Now one of you changes the string in `main.c`, then commits and pushes.
- Meanwhile, I also modified my copy of `main.c` and will push now.

```
To https://github.com/tvale/git-workshop.git
! [rejected]          master -> master (non-fast-forward)
error: failed to push some refs to
'https://github.com/tvale/git-workshop.git'
```

- What happened?

Hands On

- Git is not allowing me to push my changes because someone has already pushed theirs first;
- I must pull the changes before pushing my own modifications;
- Two possible scenarios:
 - Everything goes okay; or
 - My modifications conflict with the pulled changes!

Hands On

- If everything goes okay:

```
Auto-merging main.c
Merge made by the 'recursive' strategy.
 main.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- `$ git push origin`

Hands On

- If there are conflicts:

```
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then
commit the result.
```

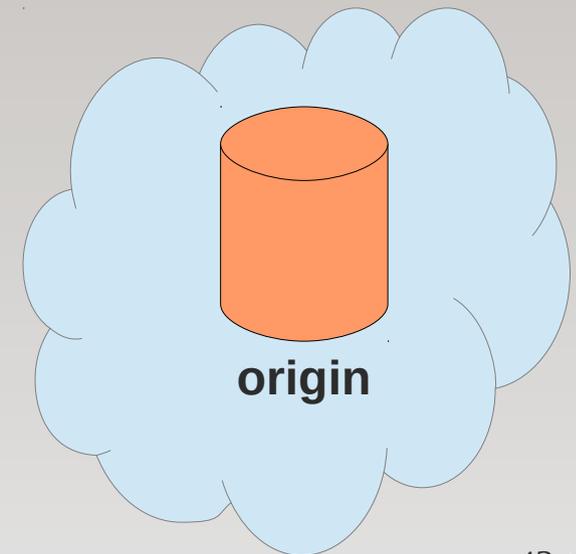
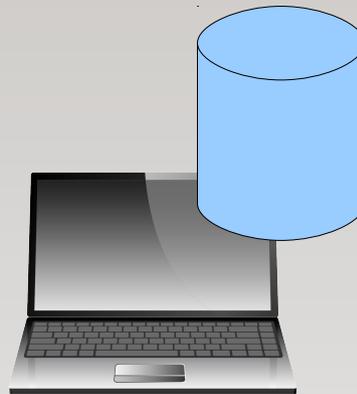
```
<<<<<<< HEAD
  My modifications
=====
  Changes in GitHub
>>>>>>> ...
```

Hands On

- When we resolve the conflicts in a file, we mark as solved:
 - `$ git add <file>`
- After fixing all conflicts, we commit...
 - `$ git commit`
- ...and can push now.
 - `$ git push origin`

Commands, Revisited

- `$ git clone`
- `$ git push`
- `$ git pull`



Conclusion

- Learned how to use git to manage the evolution of your projects, on your own;
- Learned how to use git and GitHub to work as a team on the same project.

References

- <http://git-scm.com/>
- <https://github.com/>
- <http://git-scm.com/book>
- <http://www.ericssink.com/vcbe/>

Thank you.