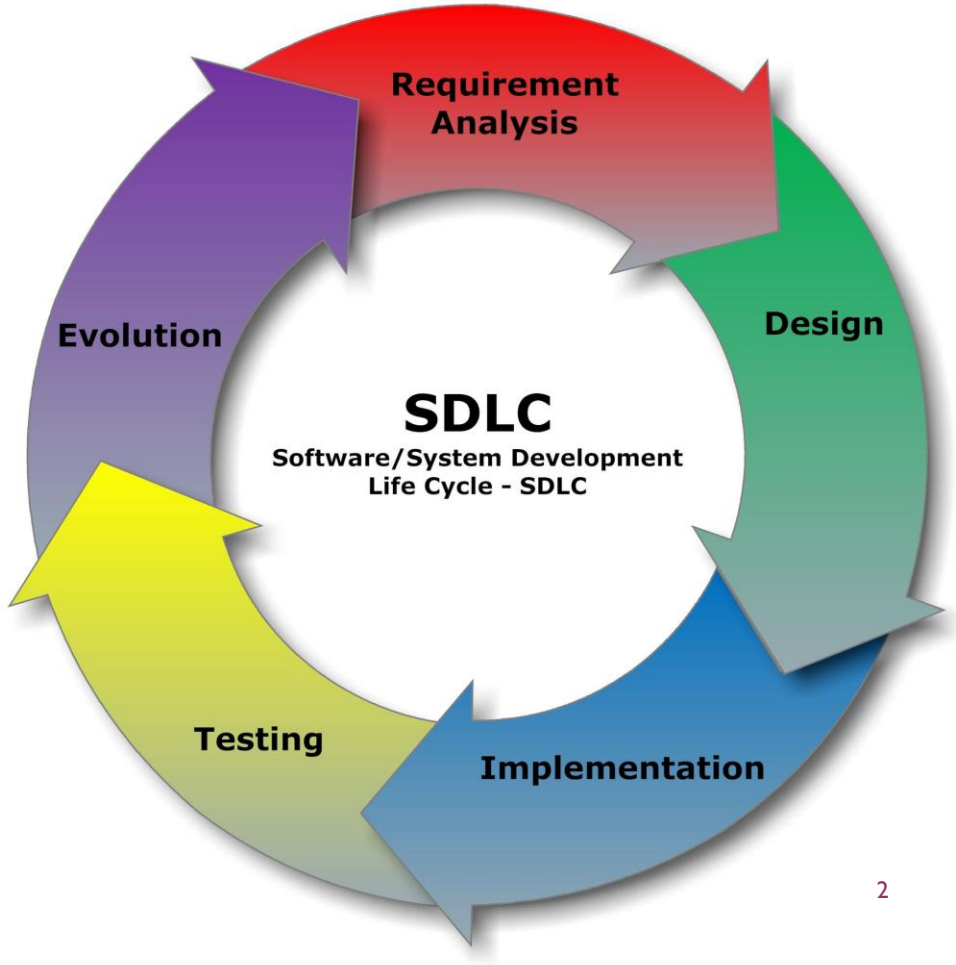
VERSION CONTROL

EMBEDDED SYSTEM PROGRAMMING

STAGES IN SOFTWARE DEVELOPMENT

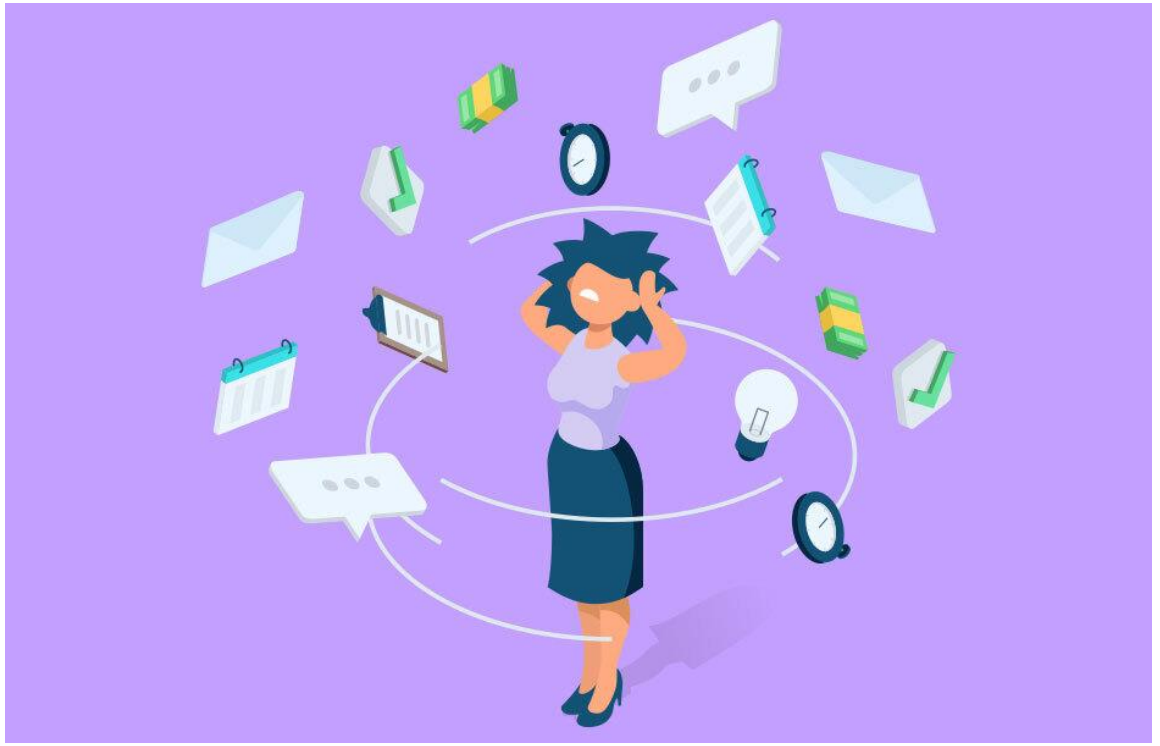
- Requirements Analysis
- High-level Design
  - [Plan (SE2800)]
- Low-level Design
- Implementation

- Unit Test (SE2832)
- Integration
- System Test (SE3800)
- Deploy (SDL)
- Maintain (SDL)



MULTIPLE PROJECTS RUN CONCURRENTLY

- In many cases, a person may be handling multiple projects at the same time
- New features are typically being added to the next version
- While at the same time, defects need to be corrected in existing releases
- This happens more in SDL

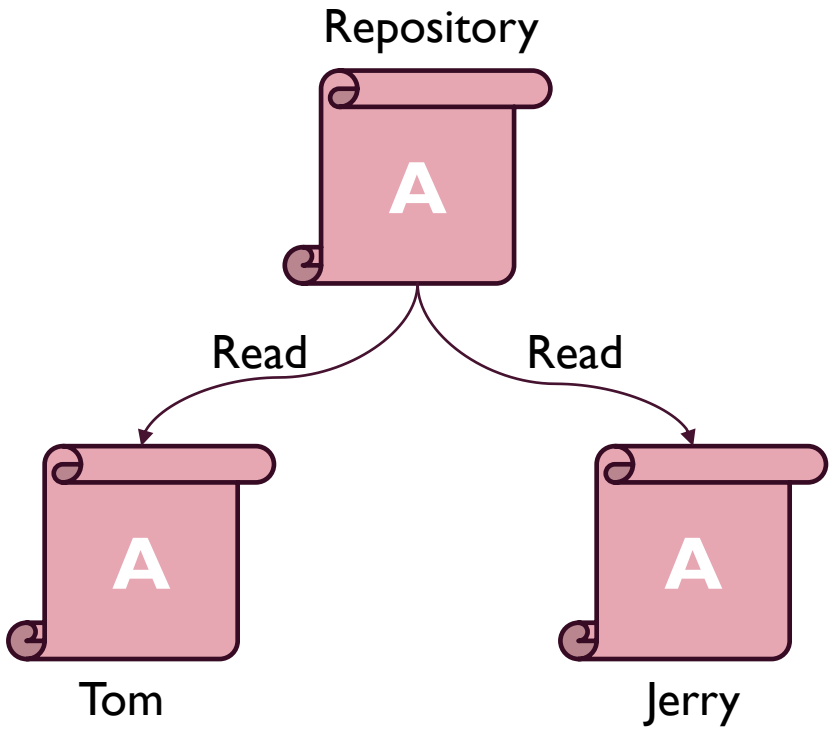


A TEAM WORKING FOR A PRODUCT RELEASE

- All team members work on the same code and develop their respective parts
- An engineer may be responsible for an entire class
- Or just a few methods within a particular class

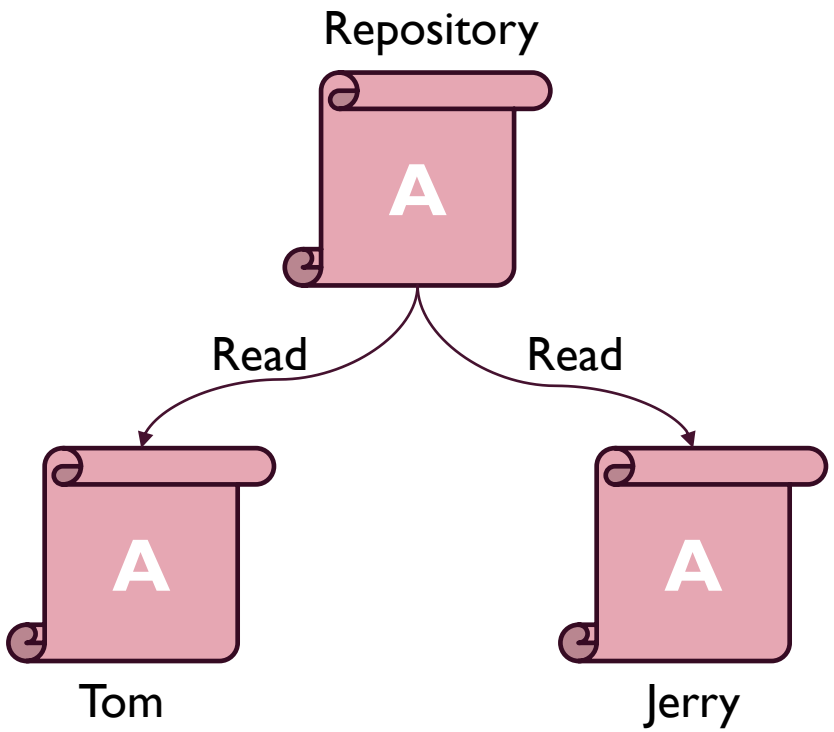
FILES MUST BE SHARED AMONG DEVELOPERS ON A TEAM PROJECT

- Shared files can be kept in some type of Repository where they can be accessed and worked on by multiple individuals
- Tom and Jerry get their own respective local Working Copies of the Master File in the Repository

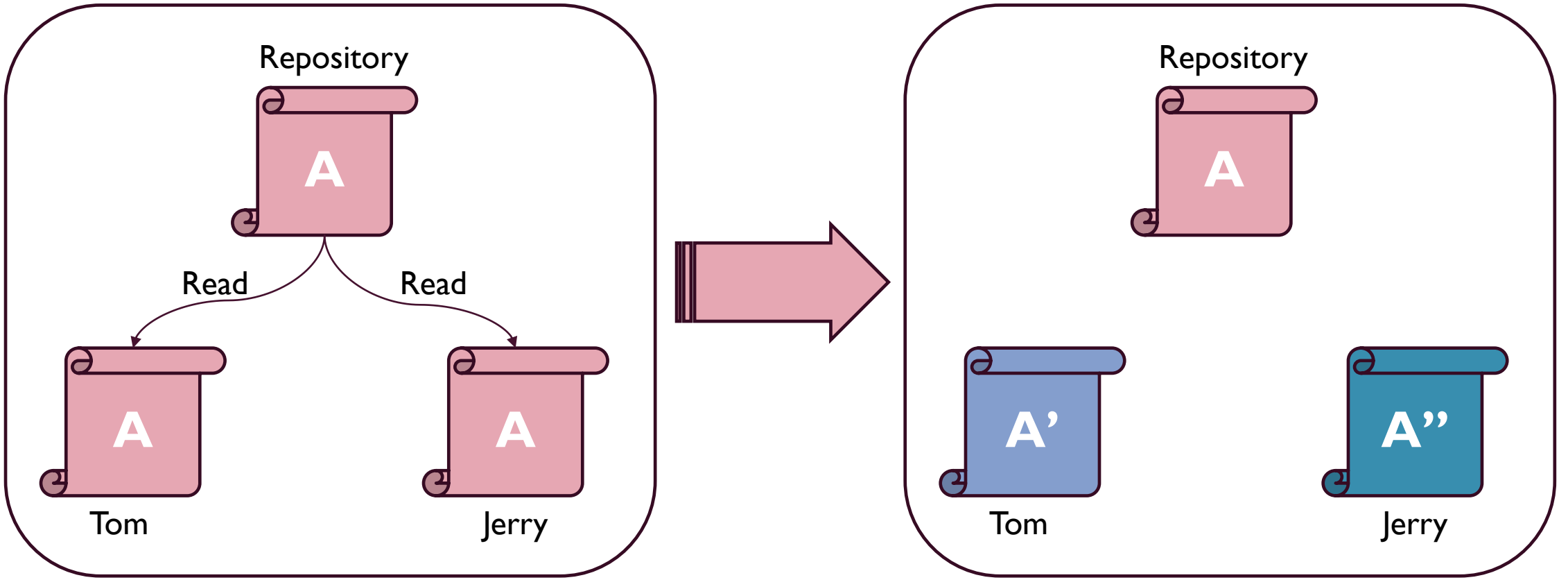


STORAGE MEDIUM FOR REPOSITORY?

- USB/SD drive
- Private network drive
  - E.g. shared "X:" drive
- Cloud drive
  - Google drive
  - Dropbox
  - iCloud



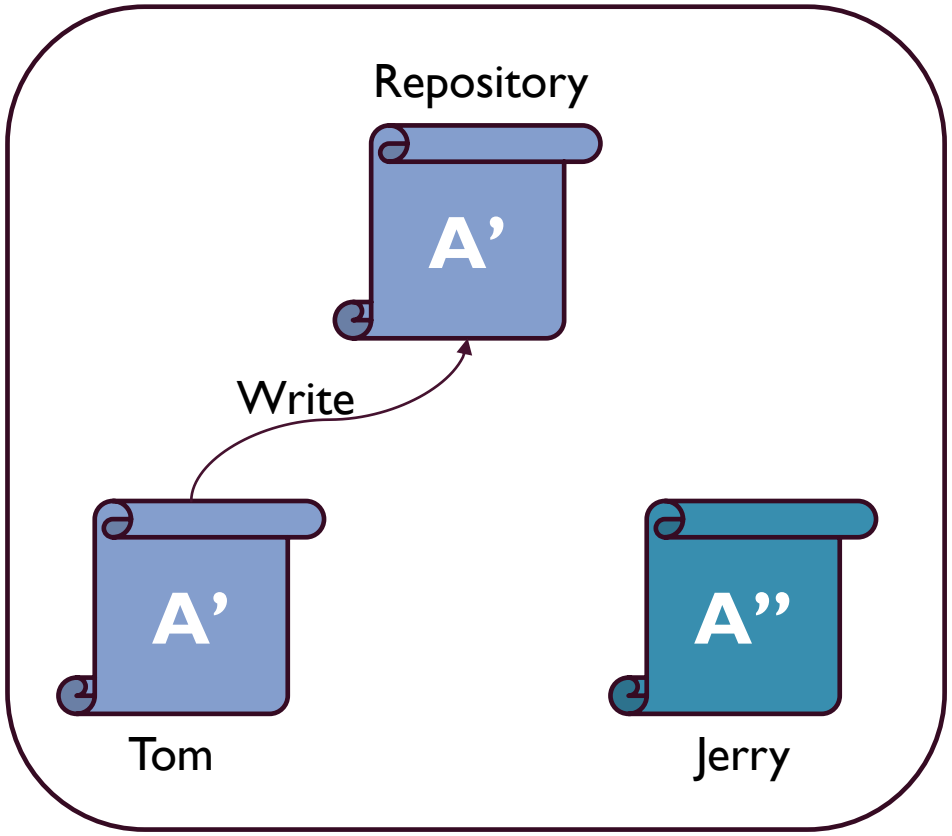
PROBLEMS WITH FILE SHARING



Two users read the same file

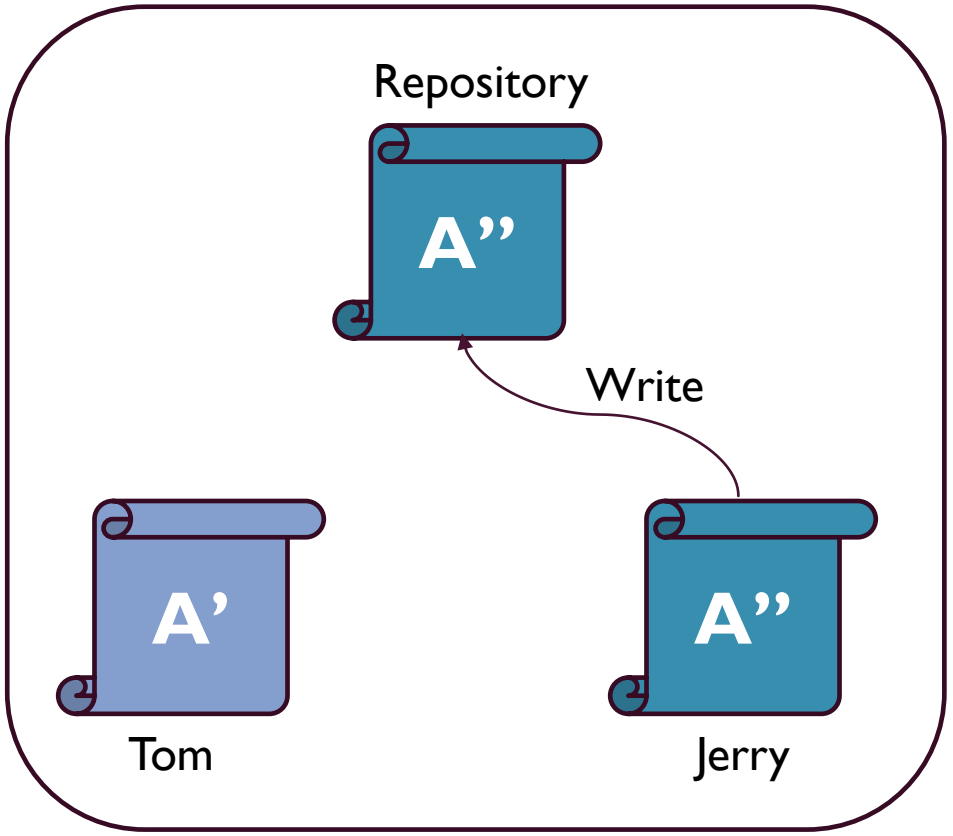
After editing, both have different versions of the same file

PROBLEM TO AVOID



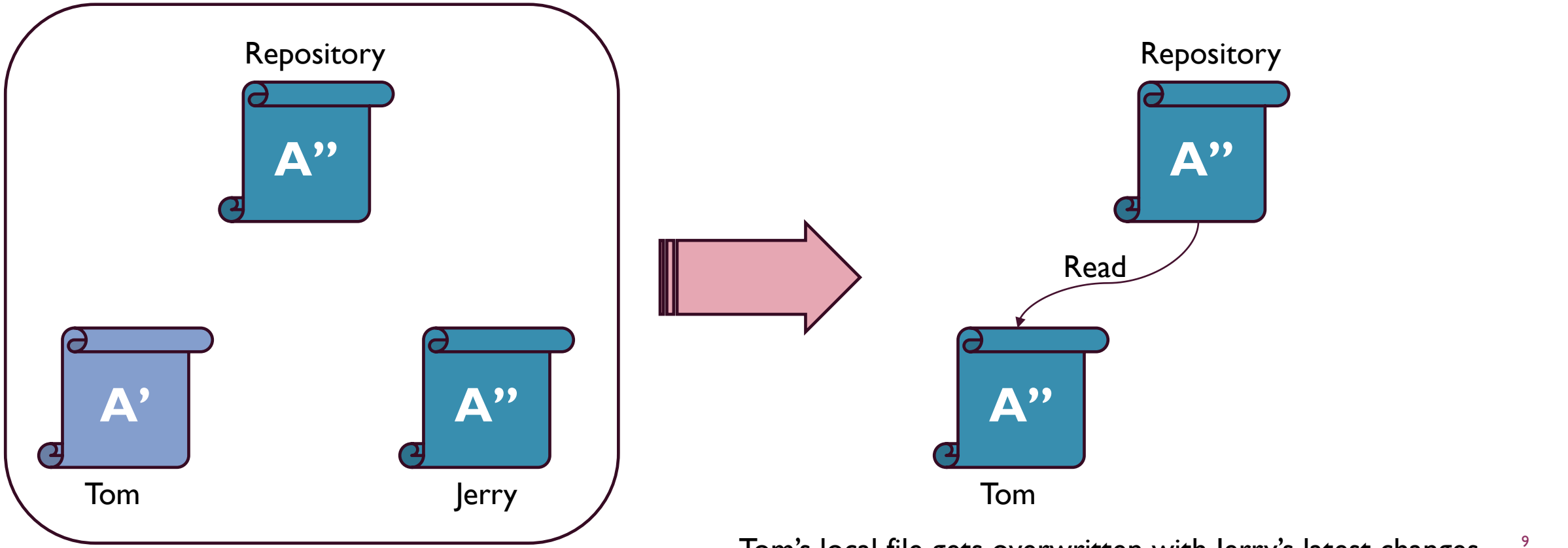
Tom publishes his version first

Jerry accidentally overwrites Tom's version



Tom's changes are still held locally

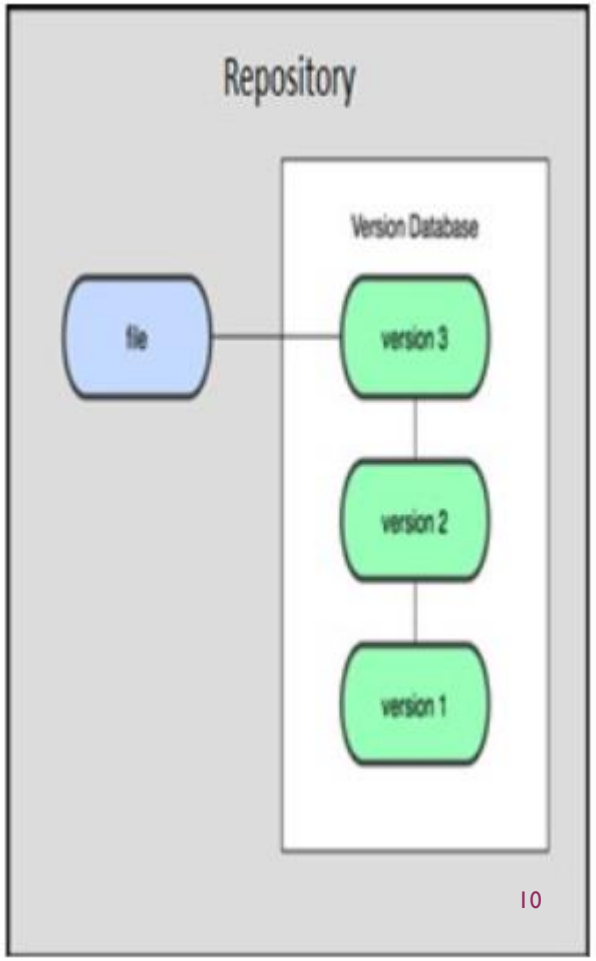
PROBLEM TO AVOID



Tom’s local file gets overwritten with Jerry’s latest changes

REVISION/VERSION HISTORY MIGHT HELP

- USB/SD, private network drives do not maintain revision history
- Cloud drives (e.g. Google Drive, Dropbox, iCould/Time Machine) maintain a (limited) history of the various revisions of a file
  - Google Drive: keeps last 100 revisions or last 30 days
  - Supported in "Classic" mode only – going away?
- IF you detect a collision, you can manually merge the differences and resynchronize your work
  - Might be workable for two people – what about 5 or 10?



OTHER DISADVANTAGES TO GOOGLE DRIVE/DROPBOX/ICLOUD

- Security?
- Support (bug fixes and new features)?
- Ownership of stored information?
- Cost?
- Ease of Use?
- Acceptance?

SPECIAL-PURPOSE REVISION CONTROL SYSTEMS HAVE BEEN AROUND A LOT LONGER THAN GOOGLE DRIVE OR DROPBOX

- SCCS, 1972 (IBM)
- RCS, 1982 (Unix)
- SourceSafe 1995 (Microsoft)
- CVS, 1986 (multi-platform)
- SVN, 2000 (multi-platform)
- Git and Mercurial, 2005 (multi-platform)

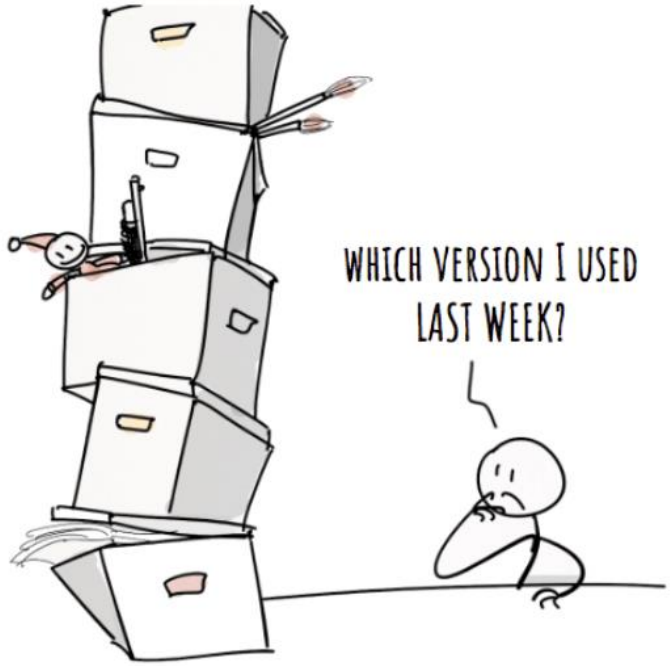
The way these work has evolved over the years

THE ISSUE: EFFECTIVE MANAGEMENT OF SOFTWARE ARTIFACTS, ESPECIALLY CODE

- SCM - Source Code Management
- SCCM - Source Code Configuration Management
- RCS - Revision Control System
- VCS - Version Control System
- DVCS - Distributed Version Control Systems

VERSION CONTROL SYSTEM

- A method for centrally storing files
- Keeping a record of changes
- Who did what, when in the system
- Covering yourself when things inevitably go wrong



ADVANTAGES AS AN INDIVIDUAL

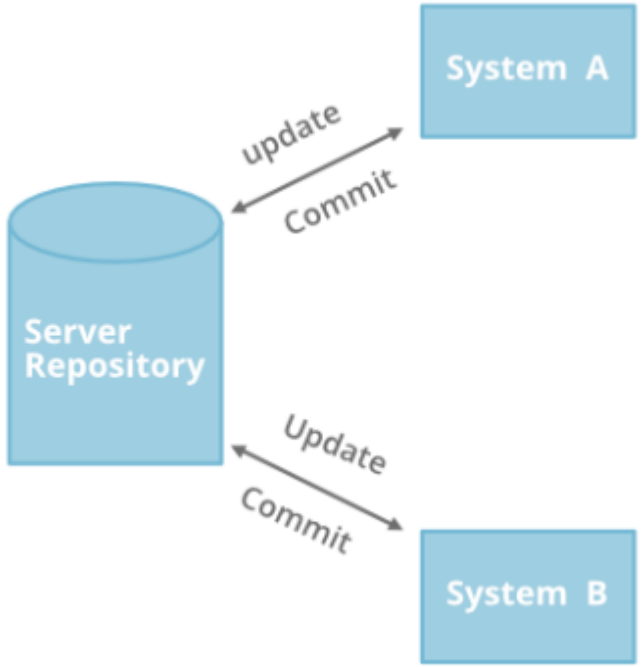
- Back-up methodology
- Increments — know which version is live
- Point in time marking aka. Tagging
- Branching — release versions maintained & main development can continue
- Change history — when features were added or amended

ADVANTAGES AS A TEAM

- As Individual plus: -
- Allow multiple developers (in remote locations) to work on same code base
- Merge changes across same files — handle collisions
- Answer who did what — blame / praise

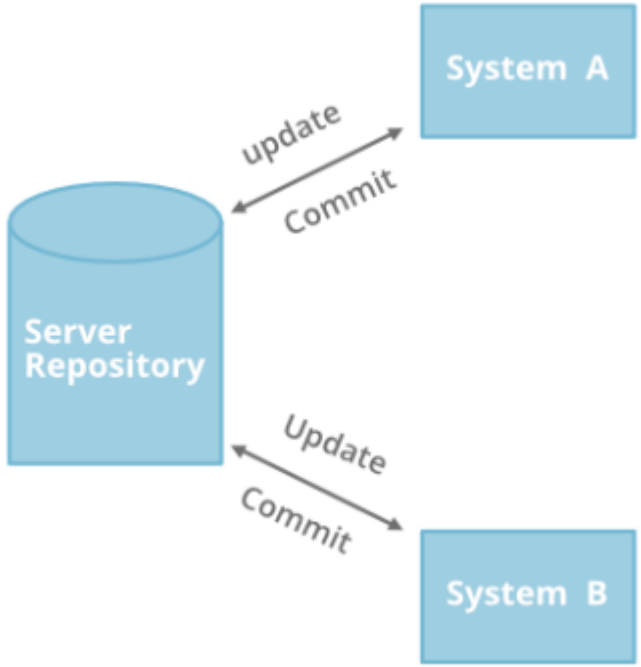
CENTRALIZED VERSION CONTROL

- A server and a client are involved
- Server - master repository
  - Contains all the versions of the code
- To work on any project, firstly user or client needs to get the code from the master repository or server
- So the client communicates with the server and pulls all the code or current version of the code from the server to their local machine



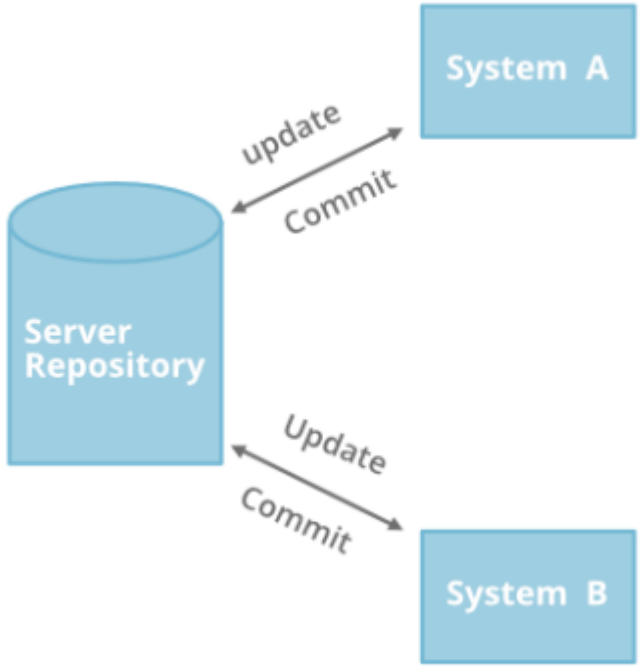
CENTRALIZED VERSION CONTROL

- Master repository must be update to get the local copy of the code in the system
- Once the latest version of the code is available the programmer can make changes in the code and commit those changes straight forward into the master repository
- Commit → merging the local code into the master repository or making a new version of the source code

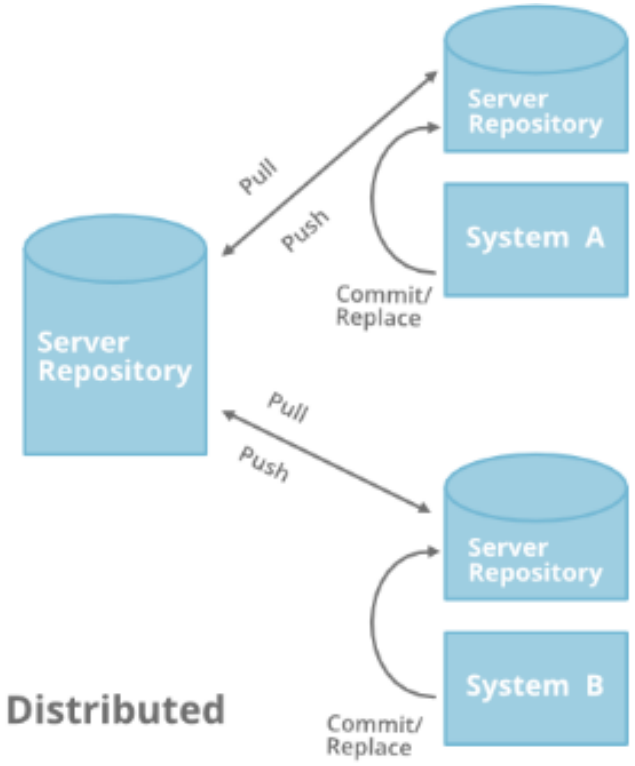


CENTRALIZED VERSION CONTROL

- There will be just one repository and that will contain all the history or version of the code and different branches of the code
- Basic workflow:
  - Getting the latest version of the code from a central repository that will contain other people's code
  - Making changes in the code
  - Committing or merging those changes into the central repository

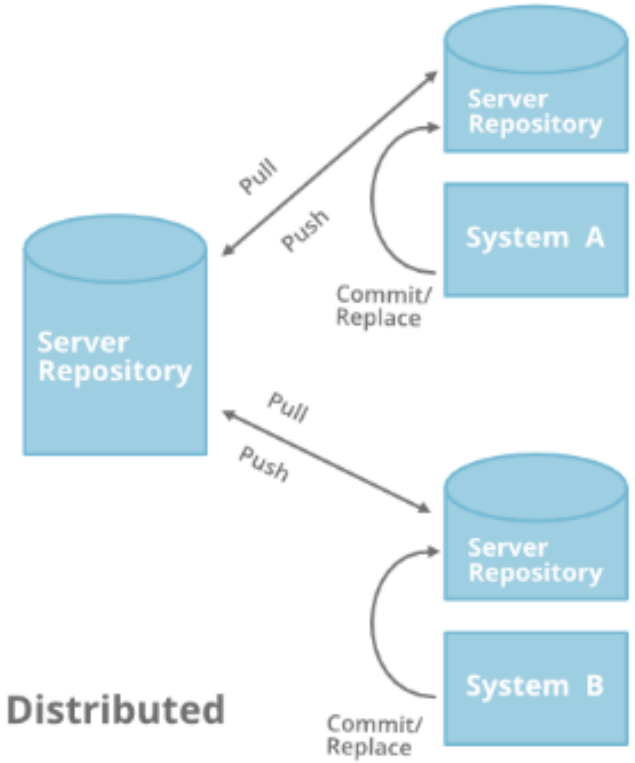


DISTRIBUTED VERSION CONTROL



- Instead of one single repository which is the server, here every single developer or client has their own server
  - They will have a copy of the entire history or version of the code and all of its branches in their local server or machine
- Every client or user can work locally and disconnected which is more convenient than centralized source control and that's why it is called distributed

DISTRIBUTED VERSION CONTROL



- Master repo. – data and version → local machine
- After making the changes in the local machine, the programmer can commit in the local machine
- A request is issued from local machine to master repository in order to push the changes into the master repository
- Master repository will have different 'sets of changes' from each individual developer's repository

PROS AND CONS

- DVCS has the biggest advantage in that it allows to work offline and gives flexibility
  - The entire history of the code is available in local hard drive, so all the changes will be committed in local server or local repository which doesn't require an internet connection, but this is not in the case of CVCS
- DVCS is faster than CVCS because each and every command need not be communicated to the remote server
- Working on branches is easy in DVCS. Every developer has an entire history of the code in DVCS, so developers can share their changes before merging all the 'sets of changes to the remote server
  - In CVCS it's difficult and time-consuming to work on branches because it requires to communicate with the server directly

PROS AND CONS

- If the project has a long history or the project contain large binary files, in that case, downloading the entire project in DVCS can take more time and space than usual, whereas in CVCS you just need to get few lines of code because you don't need to save the entire history or complete project in your own server so there is no requirement for additional space.
- If the main server goes down or it crashes in DVCS, you can still get the backup or entire history of the code from your local repository or server where the full revision of the code is already saved.
  - This is not in the case of CVCS, there is just a single remote server that has entire code history.
- Merge conflicts with other developer's code are less in DVCS. Because every developer work on their own piece of code.
  - Merge conflicts are more in CVCS in comparison to DVCS.

VERSION CONTROL TYPES

- CVS — Concurrent Version System - http://www.nongnu.org/cvs/
- SVN — Subversion - http://subversion.tigris.org/
- Git* - http://git.or.cz/
- Bazaar* - http://bazaar-vcs.org/
- Mercurial* - http://www.selenic.com/mercurial/
- Monotone* - http://www.monotone.ca/
- VSS — Visual Source Safe — Microsoft visual tool

* Distributed version control

GIT

GLOBAL INFORMATION TRACKER

INITIALIZATION

- Link for downloading git:

  https://git-scm.com/download/win

COMMANDS

```
add         Add file contents to the index
bisect      Find the change that introduced a bug by binary search
branch      List, create, or delete branches
checkout    Checkout and switch to a branch
clone       Clone a repository into a new directory
commit      Record changes to the repository
diff        Show changes between commits, the commit and working trees, etc
fetch       Download objects and refs from another repository
grep        Print lines matching a pattern
```

COMMANDS

init	Create an empty git repository or reinitialize an existing one
log	Show commit logs
merge	Join two or more development histories
mv	Move or rename a file, a directory, or a symlink
pull	Fetch from and merge with another repository or a local branch
push	Update remote refs along with associated objects
rebase	Forward-port local commits to the updated upstream head
reset	Reset current HEAD to the specified state
rm	Remove files from the working tree and from the index
show	Show various types of objects
status	Show the working tree status
tag	Create, list, delete, or verify a tag object signed with GPG

INITIALIZATION

- Username and email are provided to start with VCS
- Not for credential, but for logging in the history for commits
- $ git config --global user.name "Ranjith“
- user.name – key, Ranjith – value
- $ git config --global user.email “r.ranjith.engg@gmail.com”

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1
$ git config --global user.name
Ranjith

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1
$ git config --global user.email
“r.ranjith.engg@gmail.com”

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1
$ git config user.email
“r.ranjith.engg@gmail.com”

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1
$ git config user.name
Ranjith

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1
$
```

INITIALIZATION

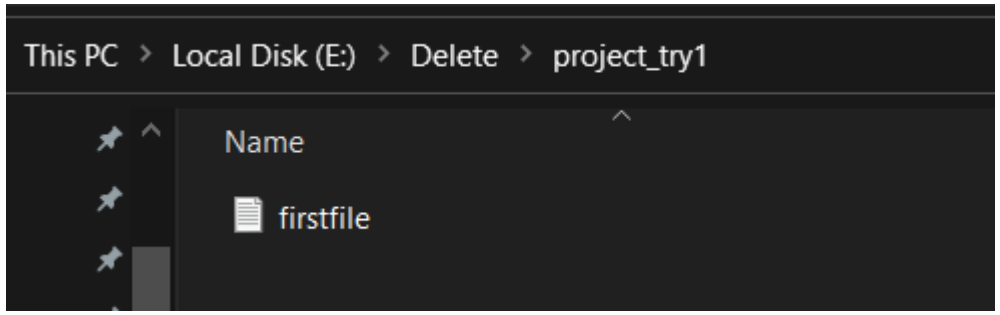
- `$ git --version`

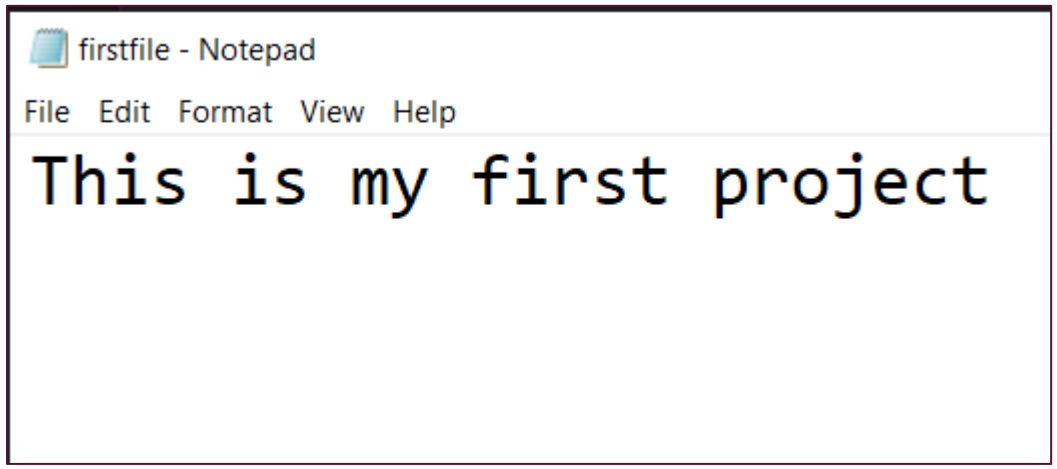
Repository Creation

- `$ mkdir ~/project_try`
- `$ cd ~/project_try/`

File Creation

- `$ echo 'This is my first project' > firstfile.txt`

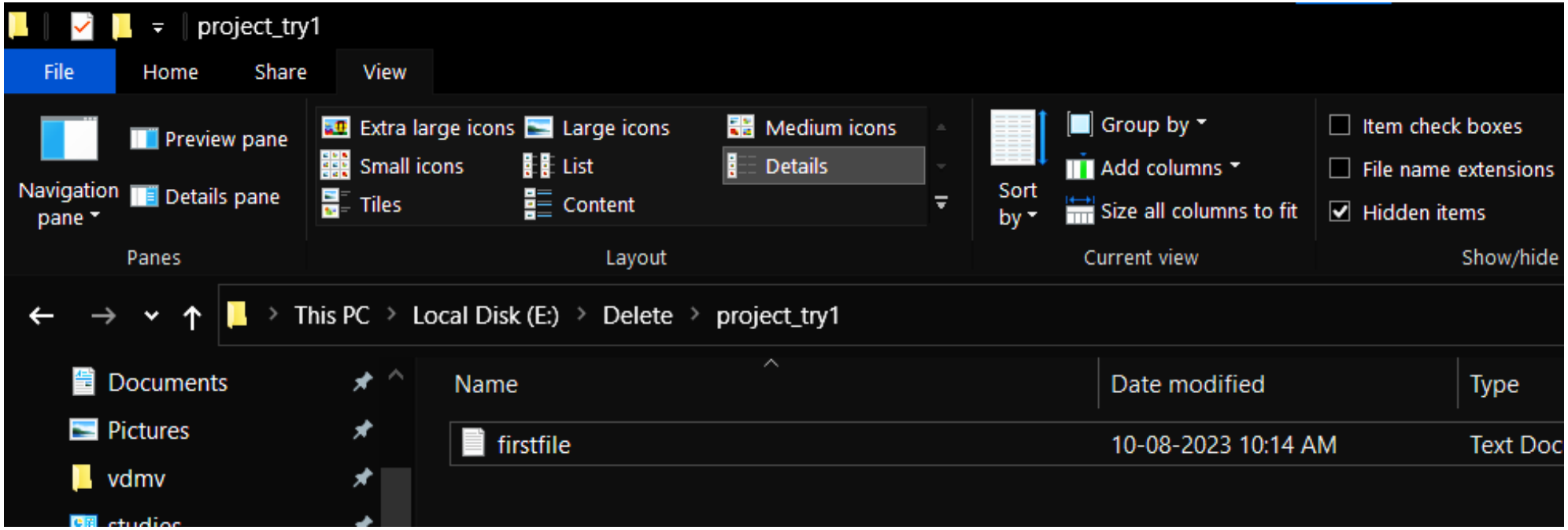

This PC
Local Disk (E:)
Delete
project_try1
Name
firstfile


firstfile - Notepad
File Edit Format View Help
This is my first project

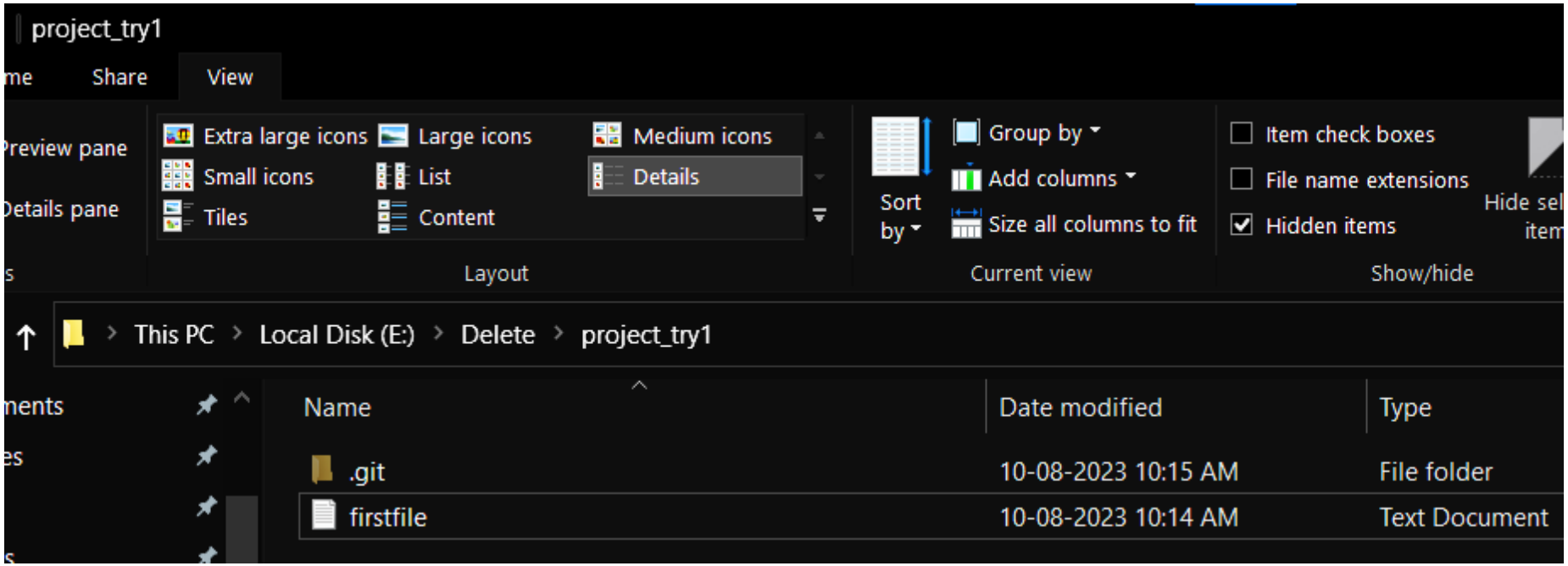
GIT REPOSITORY INITIALIZATION

Convert created repo into git repo

- **`$ git init`**
- Same for empty or full directory
- Above command creates a hidden directory -> .git at the top level
- Initially git repository is empty


project_try1
File
Home
Share
View
Navigation pane
Preview pane
Details pane
Panes
Extra large icons
Large icons
Medium icons
Small icons
List
Details
Tiles
Content
Layout
Sort by
Group by
Add columns
Size all columns to fit
Current view
Item check boxes
File name extensions
Hidden items
Show/hide
This PC
Local Disk (E:)
Delete
project_try1
Documents
Pictures
vdmv
Name
Date modified
Type
firstfile
10-08-2023 10:14 AM
Text Doc

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1
$ git init
Initialized empty Git repository in E:/Delete/project_try1/.git/
```



ADDITION OF FILE INTO REPOSITORY

Addition of single file

- **$ git add firstfile.txt**

Addition of multiple files

- **$ git add .**

This process is called Staging

A step before commit

- **$ git status**

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git add firstfile.txt
warning: in the working copy of 'firstfile.txt', LF will be replaced by CRLF the
 next time Git touches it

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master

No commits yet

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

ADDITION OF FILE INTO REPOSITORY

- **$ git status**

Message: file will be added to the repository in the next commit

- **$ git commit -m"new addtion“**

To commit using a text editor

- **$ export GIT_EDITOR=vim**

After the commit message, press ctrl+c and type :wq

MINGW64:/e/Delete/project_try1

```
First file added to the brancd - First commit
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   firstfile.txt
#
~
~
~
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git commit
[master (root-commit) 6f8e119] First file added to the brancd - First commit
 1 file changed, 1 insertion(+)
 create mode 100644 firstfile.txt
```

DETAILS OF COMMIT

- **`$ git log`**
- **`$ git show <commit number>`** → to see details of a specific commit
- **`$ git show-branch`** → details of the branch and the recent commit
- **`$ git show-branch --more=10`** → 10 recent commits with details

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git log
commit 6f8e119db7f10a2dc705e3b6231549c4abadb93f (HEAD -> master)
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Thu Aug 10 10:19:40 2023 +0530

    First file added to the brancd - First commit

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git show 6f8e119db7f10a2dc705e3b6231549c4abadb93f
commit 6f8e119db7f10a2dc705e3b6231549c4abadb93f (HEAD -> master)
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Thu Aug 10 10:19:40 2023 +0530

    First file added to the brancd - First commit

diff --git a/firstfile.txt b/firstfile.txt
new file mode 100644
index 0000000..d2239b7
--- /dev/null
+++ b/firstfile.txt
@@ -0,0 +1 @@
+This is my first project

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git show-branch
[master] First file added to the brancd - First commit
```

COMMIT DIFFERENCES

Add new line to the text file

- `$ echo “try diff option” > firstfile.txt`
- `$ git diff`

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ echo 'New line added after first commit' > firstfile.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git diff
warning: in the working copy of 'firstfile.txt', LF will be replaced by CRLF the next time Git touches it
diff --git a/firstfile.txt b/firstfile.txt
index d2239b7..2cb6084 100644
--- a/firstfile.txt
+++ b/firstfile.txt
@@ -1 +1 @@
-This is my first project
+New line added after first commit
```

ADD FILES

New file creation

- $ echo "second file addition" > sec_file.txt

View contents in the file

- $ cat sec_file.txt
- $ git status

Shows the status about newly added file

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ echo "second file addition" > sec_file.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ cat sec_file.txt
second file addition

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   firstfile.txt

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

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

ADD FILES

Stage the file

- $ git add sec_file.txt

Commit the file

- $ git commit -m"second file added on 07-08-23“

Check status

- $ git status

Check files in the repository

- $ ls

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git add sec_file.txt
warning: in the working copy of 'sec_file.txt', LF will be replaced by CRLF the next time Git touches it

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git commit -m"Second commit: new file added, first file edited"
[master 75e7a00] Second commit: new file added, first file edited
 1 file changed, 1 insertion(+)
 create mode 100644 sec_file.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   firstfile.txt

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

Only second file was staged, so second commit contains only the newly created file

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git add .
warning: in the working copy of 'firstfile.txt', LF will be replaced by CRLF the next time Git touches it

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git commit -m"Third commit: first file edited but not committed in previous commit. This commit includes first file"
[master 3bbd8e5] Third commit: first file edited but not committed in previous commit. This commit includes first file
 1 file changed, 1 insertion(+), 1 deletion(-)

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
nothing to commit, working tree clean

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git diff
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git log
commit 3bbd8e5ec898210854b519a0bf3d0723304a45ef (HEAD -> master)
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Thu Aug 10 14:22:25 2023 +0530

    Third commit: first file edited but not committed in previous commit. This commit includes first file

commit 75e7a0073020a5f5ecf1295ec724fdae654803a5
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Thu Aug 10 14:20:04 2023 +0530

    Second commit: new file added, first file edited

commit 6f8e119db7f10a2dc705e3b6231549c4abadb93f
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Thu Aug 10 10:19:40 2023 +0530

    First file added to the brancd - First commit
```

REMOVE FILE

Remove the file and stage the change

- $ git rm sec_file.txt

Check status for the file removed

- $ git status

Commit the changes

- $ git commit -m"sec_file.txt removed on 07-08-2023"

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git rm sec_file.txt
rm 'sec_file.txt'

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    sec_file.txt


rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git diff

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git commit -m"Fourth commit: sec_file removed"
[master e02bcc3] Fourth commit: sec_file removed
 1 file changed, 1 deletion(-)
 delete mode 100644 sec_file.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

RENAME FILE

Case 1

- $ mv first.txt second.txt
- $ git rm first.txt
- $ git add second.txt
- $ git commit –m”first got renamed to second”

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ ls
firstfile.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ mv firstfile.txt firstfile_new.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ ls
firstfile_new.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    firstfile.txt

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

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

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git rm firstfile.txt
rm 'firstfile.txt'

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    firstfile.txt

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


rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git add firstfile_new.txt
warning: in the working copy of 'firstfile_new.txt', LF will be replaced by CRLF the next time Git touches it

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    firstfile.txt -> firstfile_new.txt


rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ git commit -m"Fifth commit: first file renamed to firstfile_new"
[master d5834d0] Fifth commit: first file renamed to firstfile_new
 1 file changed, 0 insertions(+), 0 deletions(-)
```

RENAME FILE

Case 11

- $ git mv first.txt second.txt
- $ git commit –m”first got renamed to second”

COPY OF REPOSITORY

Go out of the git repo and enter home directory

- $ cd ~

Clone the git repository within home directory

- $ git clone project_try project_try_clone

On a local file system, cloning is similar to cp –a or rsync

Explore the original and cloned repo

- $ ls -lsa project_try project_try_clone/

l – long listing format; s – size; a – all

- $ diff -r project_try project_try_clone/

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete/project_try1 (master)
$ cd ~
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete
$ git clone project_try1 project_try_clone
Cloning into 'project_try_clone'...
done.
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete
$ ls -lsa project_try1 project_try_clone
project_try1:
total 21
 0 drwxr-xr-x 1 rranj 197609  0 Aug 10 14:28 ./
16 drwxr-xr-x 1 rranj 197609  0 Aug 10 15:36 ../
 4 drwxr-xr-x 1 rranj 197609  0 Aug 10 14:30 .git/
 1 -rw-r--r-- 1 rranj 197609 34 Aug 10 14:16 firstfile_new.txt

project_try_clone:
total 21
 0 drwxr-xr-x 1 rranj 197609  0 Aug 10 15:36 ./
16 drwxr-xr-x 1 rranj 197609  0 Aug 10 15:36 ../
 4 drwxr-xr-x 1 rranj 197609  0 Aug 10 15:36 .git/
 1 -rw-r--r-- 1 rranj 197609 35 Aug 10 15:36 firstfile_new.txt
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/Delete
$ diff -r project_try1 project_try_clone
Only in project_try1/.git: COMMIT_EDITMSG
diff -r project_try1/.git/config project_try_clone/.git/config
7a8,13
> [remote "origin"]
>       url = E:/Delete/project_try1
>       fetch = +refs/heads/*:refs/remotes/origin/*
> [branch "master"]
>       remote = origin
>       merge = refs/heads/master
Binary files project_try1/.git/index and project_try_clone/.git/index differ
diff -r project_try1/.git/logs/HEAD project_try_clone/.git/logs/HEAD
1,5c1
< 0000000000000000000000000000000000000000 6f8e119db7f10a2dc705e3b6231549c4abadb93f Ranjith <"r.ranjith.engg@gmail.com"> 1691642980 +05
30      commit (initial): First file added to the brancd - First commit
< 6f8e119db7f10a2dc705e3b6231549c4abadb93f 75e7a0073020a5f5ecf1295ec724fdae654803a5 Ranjith <"r.ranjith.engg@gmail.com"> 1691657404 +05
30      commit: Second commit: new file added, first file edited
< 75e7a0073020a5f5ecf1295ec724fdae654803a5 3bbd8e5ec898210854b519a0bf3d0723304a45ef Ranjith <"r.ranjith.engg@gmail.com"> 1691657545 +05
30      commit: Third commit: first file edited but not committed in previous commit. This commit includes first file
< 3bbd8e5ec898210854b519a0bf3d0723304a45ef e02bcc3fb5150c8d9257bfd015dc0721720503f9 Ranjith <"r.ranjith.engg@gmail.com"> 1691657775 +05
30      commit: Fourth commit: sec_file removed
< e02bcc3fb5150c8d9257bfd015dc0721720503f9 d5834d0820e01e5ccbde25e7280760e0316ef77e Ranjith <"r.ranjith.engg@gmail.com"> 1691658000 +05
30      commit: Fifth commit: first file renamed to firstfile_new
---
> 0000000000000000000000000000000000000000 d5834d0820e01e5ccbde25e7280760e0316ef77e Ranjith <"r.ranjith.engg@gmail.com"> 1691661990 +05
30      clone: from E:/Delete/project_try1
diff -r project_try1/.git/logs/refs/heads/master project_try_clone/.git/logs/refs/heads/master
1,5c1
< 0000000000000000000000000000000000000000 6f8e119db7f10a2dc705e3b6231549c4abadb93f Ranjith <"r.ranjith.engg@gmail.com"> 1691642980 +05
30      commit (initial): First file added to the brancd - First commit
< 6f8e119db7f10a2dc705e3b6231549c4abadb93f 75e7a0073020a5f5ecf1295ec724fdae654803a5 Ranjith <"r.ranjith.engg@gmail.com"> 1691657404 +05
30      commit: Second commit: new file added, first file edited
< 75e7a0073020a5f5ecf1295ec724fdae654803a5 3bbd8e5ec898210854b519a0bf3d0723304a45ef Ranjith <"r.ranjith.engg@gmail.com"> 1691657545 +05
30      commit: Third commit: first file edited but not committed in previous commit. This commit includes first file
< 3bbd8e5ec898210854b519a0bf3d0723304a45ef e02bcc3fb5150c8d9257bfd015dc0721720503f9 Ranjith <"r.ranjith.engg@gmail.com"> 1691657775 +05
30      commit: Fourth commit: sec_file removed
< e02bcc3fb5150c8d9257bfd015dc0721720503f9 d5834d0820e01e5ccbde25e7280760e0316ef77e Ranjith <"r.ranjith.engg@gmail.com"> 1691658000 +05
30      commit: Fifth commit: first file renamed to firstfile_new
---
> 0000000000000000000000000000000000000000 d5834d0820e01e5ccbde25e7280760e0316ef77e Ranjith <"r.ranjith.engg@gmail.com"> 1691661990 +05
30      clone: from E:/Delete/project_try1
Only in project_try_clone/.git/logs/refs: remotes
Only in project_try_clone/.git: packed-refs
Only in project_try_clone/.git/refs: remotes
diff -r project_try1/firstfile_new.txt project_try_clone/firstfile_new.txt
1c1
< New line added after first commit
---
> New line added after first commit
```

GIT REPO

- Git repository not only provides a complete working copy of all the files in the repository, but also a copy of the repository itself with which to work
- Unlike file data and other repository metadata, configuration settings are not propagated from one repository to another during a clone, or duplicating, operation
- Git manages and inspects configuration and setup information on a per-site, per-user, and per-repository basis

GIT DATA STRUCTURES

Within a repository, Git maintains two primary data structures

- Object store
  - Designed to be efficiently copied during a clone operation as part of the mechanism that supports a fully DVCS
- Index
  - Transitory information, is private to a repository and can be created or modified on-demand as needed

SHA (SECURE HASH ALGORITHM)

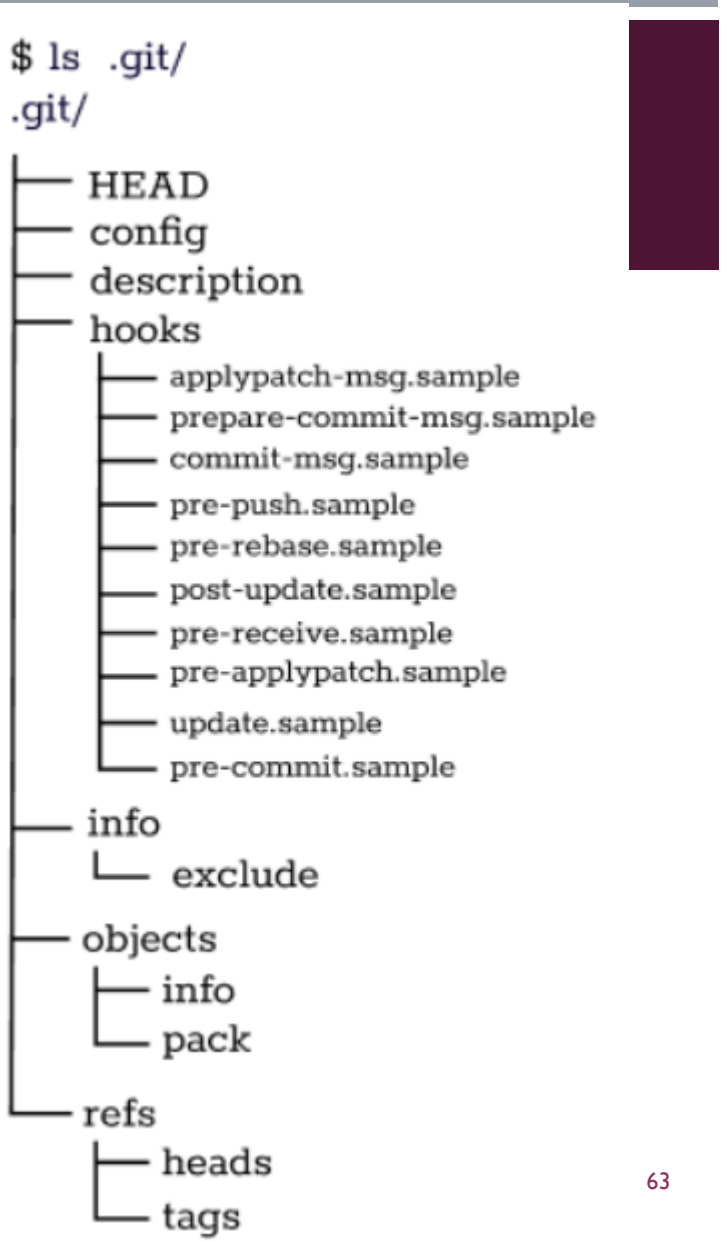
- All the information needed to represent the history of a project is stored in files referenced by a 40-digit "object name"
  - 8hf67f4664981e4397625791c8eabbb5f2279a31
- The name is calculated by taking the *SHA1* hash of the contents of the object
- The *SHA1* hash has a cryptographic hash function
- It is virtually impossible to find two different objects with the same name
  - Object can be compared with the name to infer that they are identical or not
  - Same file in different repository will be stored in the same name (SHA)

GIT OBJECT STORE

- Contains your original data files and all the log messages, author information, dates, and other information required to rebuild any revision or branch of the project
- Every object has
  - Size – Size of the content
  - Content - Content depends on of object
  - Type – Blob, Tree, Commit, Tag

CONTENTS IN THE HIDDEN .GIT

- Create a new directory
- Initialize it as git repo
- Open the folder to check .git
  - $ cd .git/objects
  - $ ls



CREATE TWO FILES

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ touch index.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ echo "Hello World!!" > index.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ touch README.md

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ echo "This is my hello world project" > README.md
```

CHECK STATUS OF NEW FILES

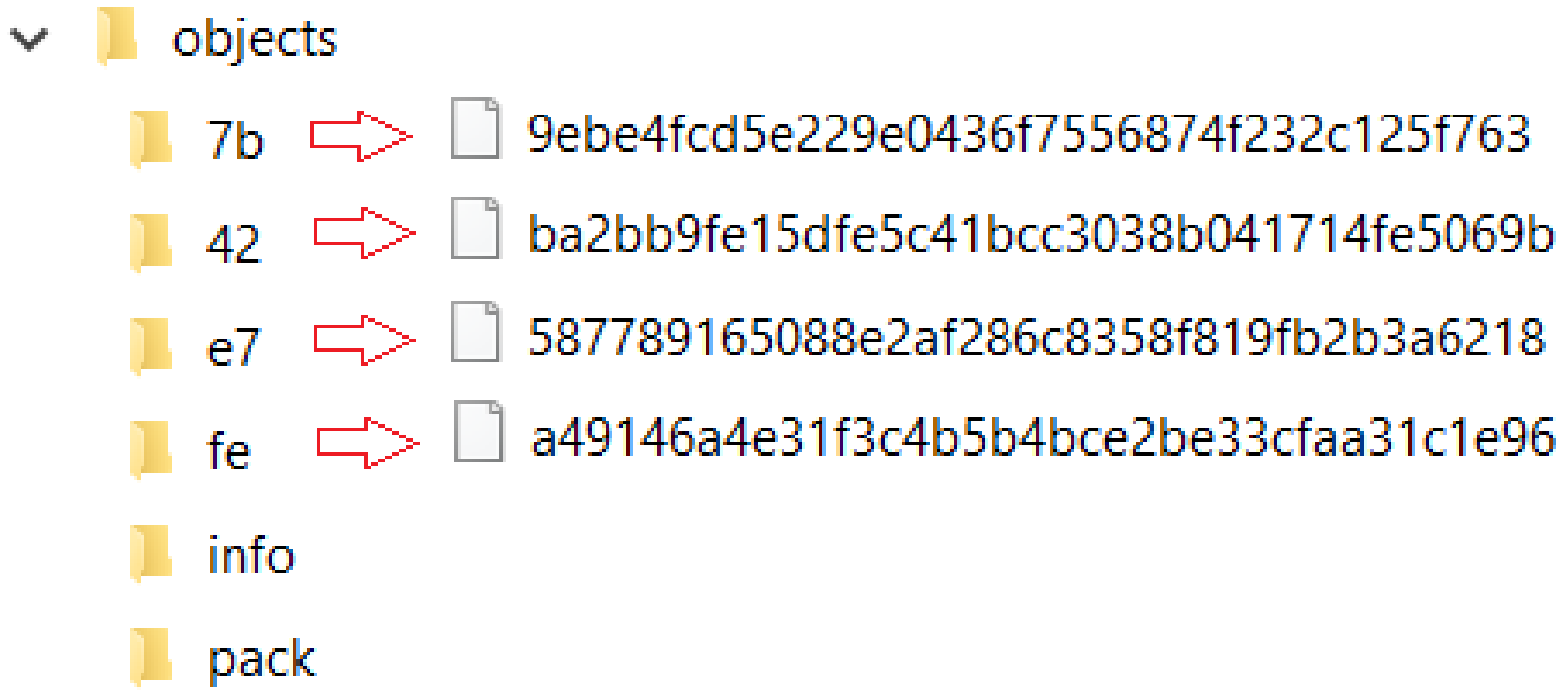
```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git status
On branch master

No commits yet

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

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

STAGE AND COMMIT THE CHANGES

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git add -A
warning: in the working copy of 'README.md', LF will be replaced by CRLF the nex
t time Git touches it
warning: in the working copy of 'index.txt', LF will be replaced by CRLF the nex
t time Git touches it

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git commit -m"Two files newly created"
[master (root-commit) fea4914] Two files newly created
 2 files changed, 2 insertions(+)
 create mode 100644 README.md
 create mode 100644 index.txt
```

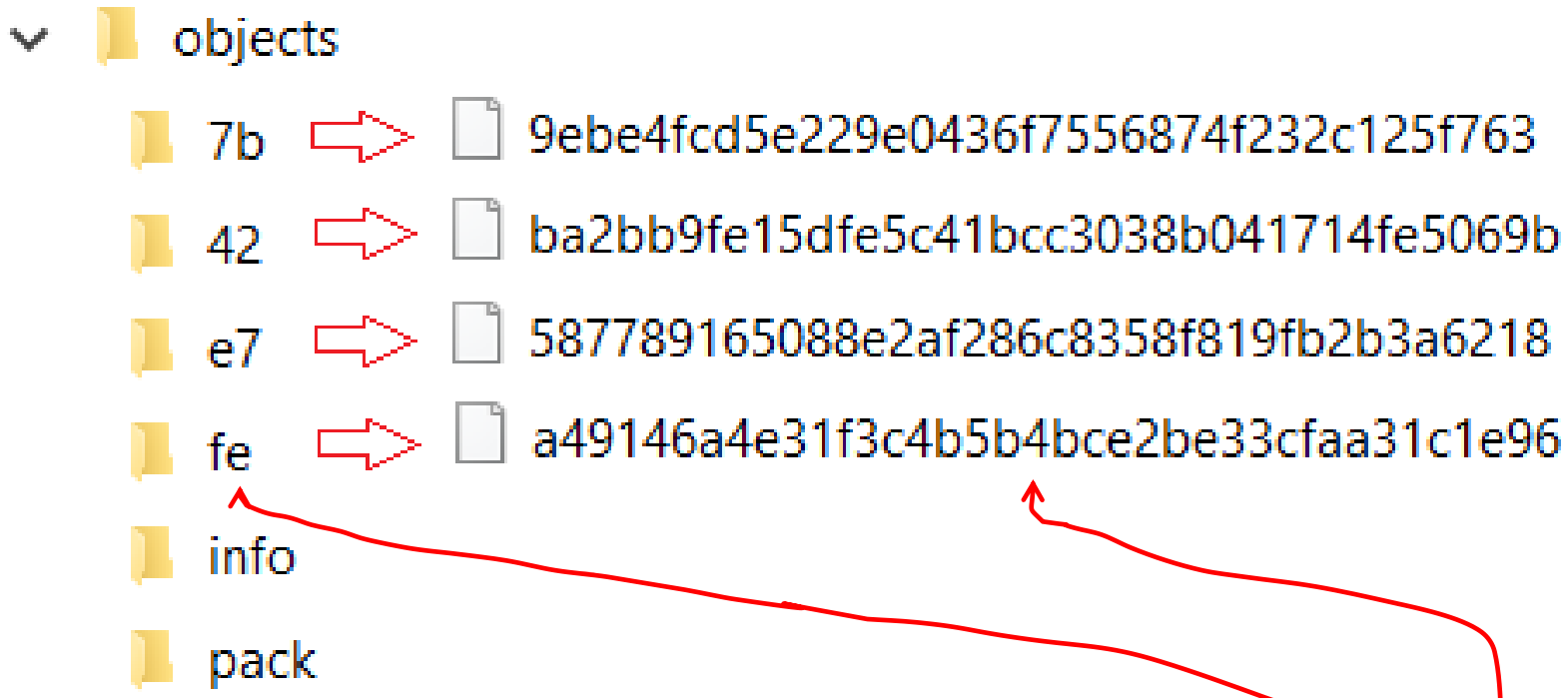
```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git status
On branch master
nothing to commit, working tree clean
```

CHECK OBJECT FOLDER



Newly created folders and files

CHECK OBJECT FOLDER



2 characters – Directory name

38 characters – File name

Git generates 40 character check sum, 40 = 2 + 38

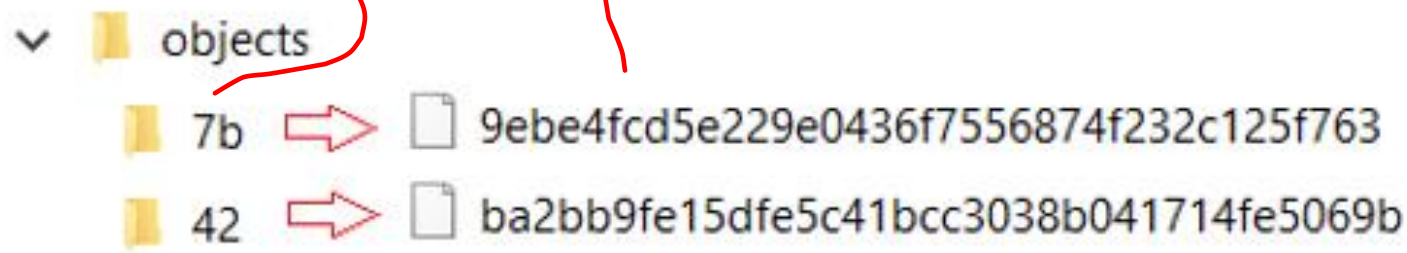
BLOB OBJECT – BINARY LARGE OBJECT

- First kind of object that gets created when a file is committed – Blob
- Contains snapshots of the file at the time of commit with check sum header

BLOB OBJECT – BINARY LARGE OBJECT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -p 7b9ebe4fcd5e229e0436f7556874f232c125f763
Hello World

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t 7b9ebe4fcd5e229e0436f7556874f232c125f763
blob

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -s 7b9ebe4fcd5e229e0436f7556874f232c125f763
27
```

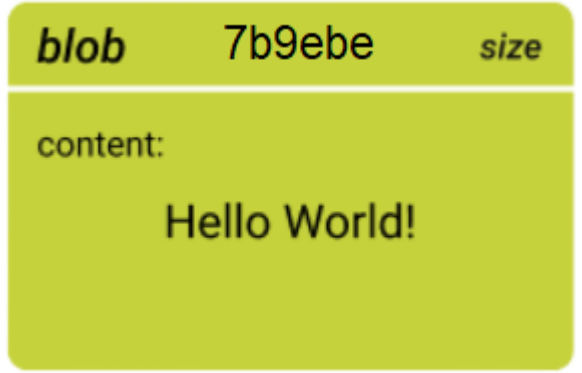
-p → Print object content
-t → Show object type
-s → Show object size

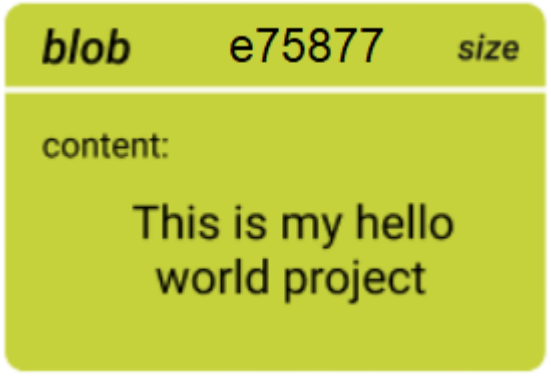


BLOB OBJECT – BINARY LARGE OBJECT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -p 7b9ebe4fcd5e229e0436f7556874f232c125f763
Hello World

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t 7b9ebe4fcd5e229e0436f7556874f232c125f763
blob

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -s 7b9ebe4fcd5e229e0436f7556874f232c125f763
27
```

Blob of index.txt





Blob of README.md

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -p e7587789165088e2af286c8358f819fb2b3a6218
This is my hello world project

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t e7587789165088e2af286c8358f819fb2b3a6218
blob

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -s e7587789165088e2af286c8358f819fb2b3a6218
31
```

BLOB OBJECT – BINARY LARGE OBJECT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file blob e7587789165088e2af286c8358f819fb2b3a6218
This is my hello world project
```

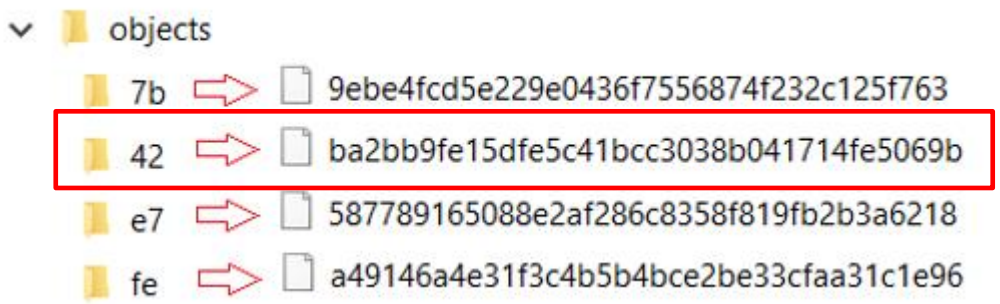
Alternate method to view content of the blob

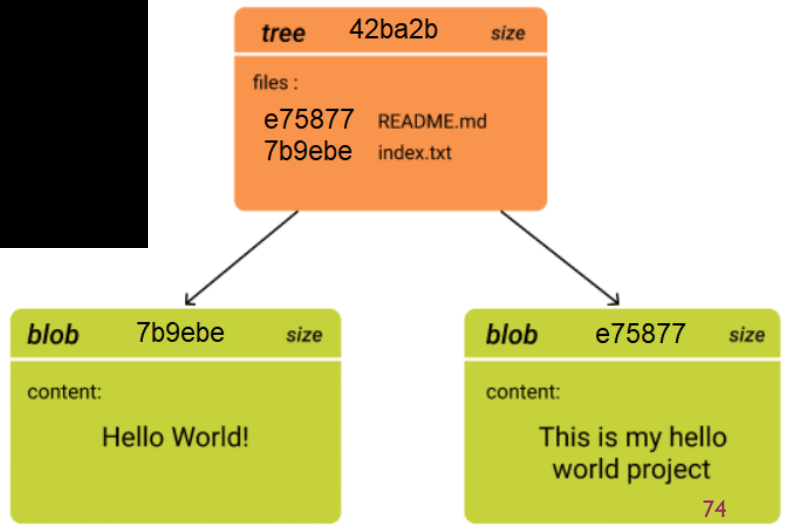
TREE OBJECT

- In this case, one tree is present
- It contains a list of all files in the current project
- It also contains a pointer to the blob object assigned with each file

TREE OBJECT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -p 42ba2bb9fe15dfe5c41bcc3038b041714fe5069b
100644 blob e7587789165088e2af286c8358f819fb2b3a6218    README.md
100644 blob 7b9ebe4fcd5e229e0436f7556874f232c125f763    index.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t 42ba2bb9fe15dfe5c41bcc3038b041714fe5069b
tree

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -s 42ba2bb9fe15dfe5c41bcc3038b041714fe5069b
74
```





TREE OBJECT

The tree object contains one line per file or subdirectory, with each line giving file permissions, object type, object hash, and filename.
An object type is usually one of "blob" for a file or "tree" for a subdirectory.

```
100644 blob e758778916508 8e2af286c8358f819fb2b3a6218    README.md
100644 blob 7b9ebe4fcd5e229e0436f7556874f232c125f763    index.txt
```

File's permission -- 100644
The file has 644 permission (ignoring the 100)
Permissions of 600 mean that the owner has full read and write access to the file, while no other user can access the file
Permissions of 644 mean that the owner of the file has read and write access, while the group members and other users on the system only have read access

TREE OBJECT

The tree object contains one line per file or subdirectory, with each line giving file permissions, object type, object hash, and filename.
An object type is usually one of "blob" for a file or "tree" for a subdirectory.

```
100644 blob e758778916508 8e2af286c8358f819fb2b3a6218    README.md
100644 blob 7b9ebe4fcd5e229e0436f7556874f232c125f763    index.txt
```

The content of this entry is represented by a blob, rather than a tree

TREE OBJECT

The tree object contains one line per file or subdirectory, with each line giving file permissions, object type, object hash, and filename.
An object type is usually one of "blob" for a file or "tree" for a subdirectory.

```
100644 blob e758778916508 8e2af286c8358f819fb2b3a6218    README.md
100644 blob 7b9ebe4fcd5e229e0436f7556874f232c125f763     index.txt
```

hash of the blob

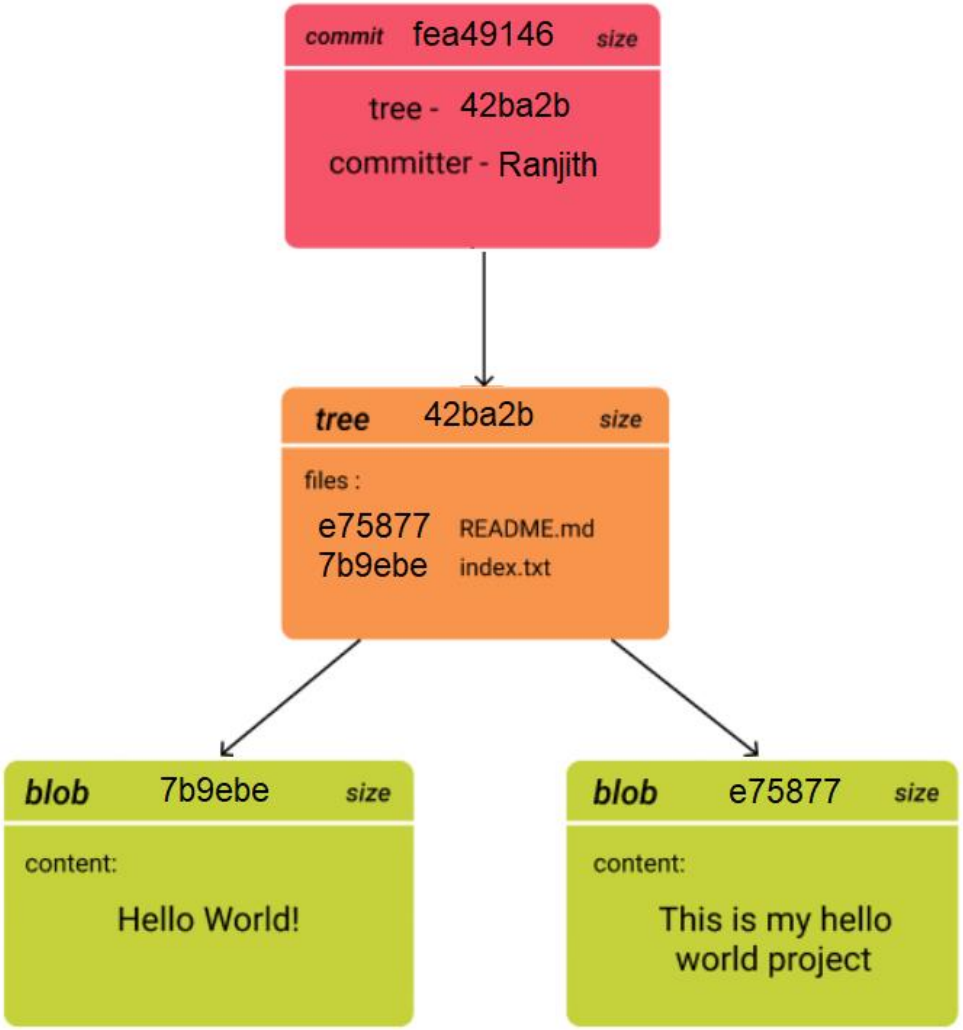
filename

COMMIT OBJECT

- The commit object contains the directory tree object hash, parent commit hash, author, committer, date, and message

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git log
commit fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96 (HEAD -> master)
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Fri Sep 30 12:13:45 2022 +0530

    Two files newly created
```

COMMIT OBJECT




```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -p fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96
tree 42ba2bb9fe15dfe5c41bcc3038b041714fe5069b
author Ranjith <"r.ranjith.engg@gmail.com"> 1664520225 +0530
committer Ranjith <"r.ranjith.engg@gmail.com"> 1664520225 +0530

Two files newly created

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96
commit

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -s fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96
204
```

A COMMIT IS DEFINED BY

- A **tree**: The SHA1 name of a tree object, representing the contents of a directory at a certain point in time
- A **parent(s)**: The SHA1 name of some number of commits which represent the immediately previous step(s) in the history of the project. The example above has **no parent**; merge commits may have more than one. A commit with *no parent* is called a "**root**" commit and represents the initial revision of a project. Each project must have at **least one root**.
- An **author**: The name of the person responsible for this change, together with its date.
- A **committer**: The name of the person who actually created the commit, with the date it was done. This may be different from the author; for example, if the author wrote a patch and emailed it to another person who used the patch to create the commit.
- A **comment** describing the commit.

A commit is usually created by git commit , which creates a commit whose parent is normally the current HEAD, and whose tree is taken from the content currently stored in the index.

TAG OBJECT

- A tag object contains
  - An object name(called simply '***object***'),
  - Object type,
  - Tag name,
  - The name of the person ("***tagger***") who created the tag, and
  - A message

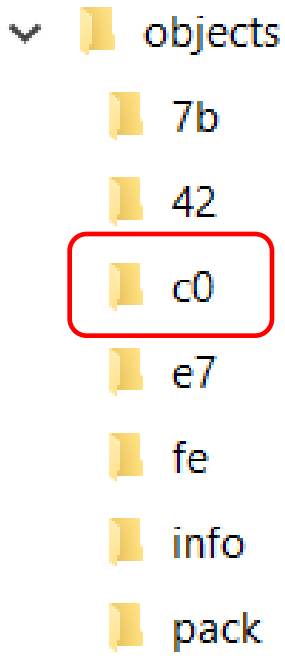
TAG OBJECT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git log
commit fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96 (HEAD -> master)
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Fri Sep 30 12:13:45 2022 +0530

    Two files newly created

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git tag -a "first tag" -m"Tag pointing to the first commit"
fatal: 'first tag' is not a valid tag name.

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git tag -a "first_tag" -m"Tag pointing to the first commit"

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git log
commit fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96 (HEAD -> master, tag: first_tag)
Author: Ranjith <“r.ranjith.engg@gmail.com”>
Date:   Fri Sep 30 12:13:45 2022 +0530

    Two files newly created
```

Tag Message

Tag Name

TAG OBJECT – CREATION IN .GIT




```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/
42/  7b/  c0/  e7/  fe/  info/  pack/
```

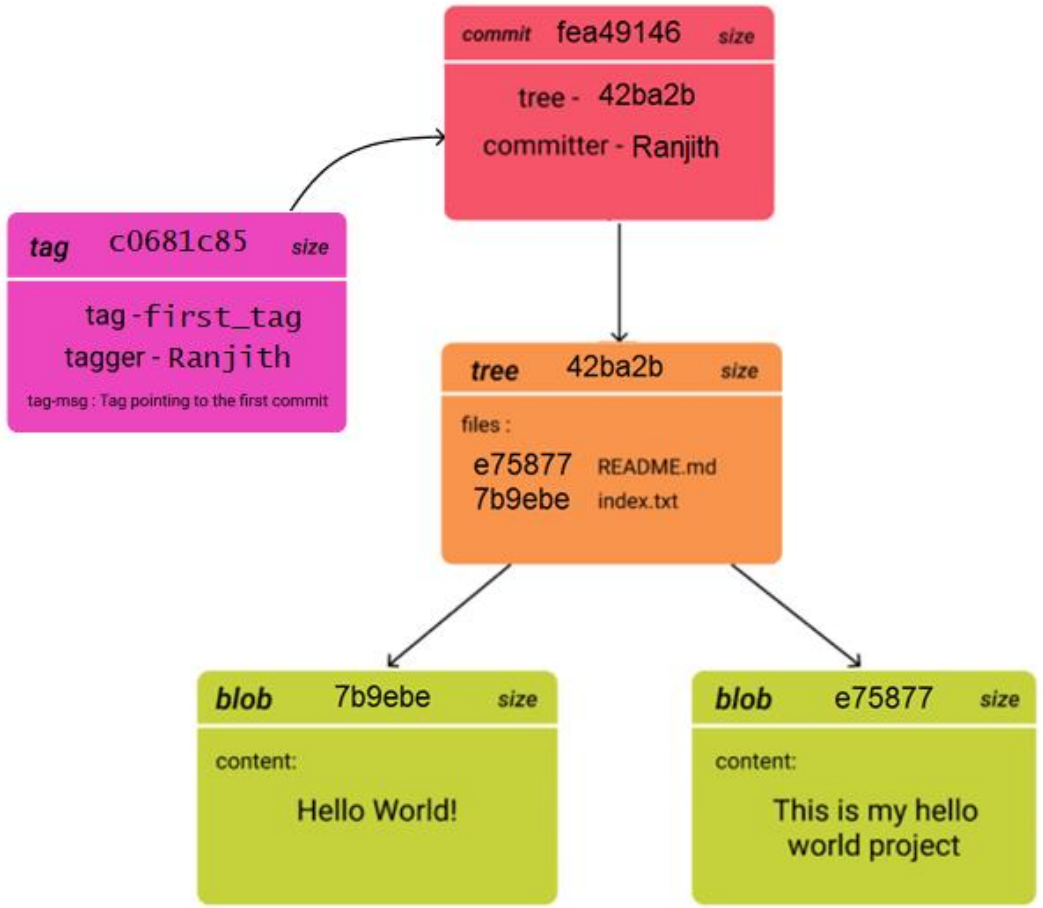
TAG OBJECT - CONTENT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects//c0/
681c85a30fddeb839b610e9aae027f1a36974f

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t c0681c85a30fddeb839b610e9aae027f1a36974f
tag

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -p c0681c85a30fddeb839b610e9aae027f1a36974f
object fea49146a4e31f3c4b5b4bce2be33cfaa31c1e96
type commit
tag first_tag
tagger Ranjith <"r.ranjith.engg@gmail.com"> 1664606564 +0530

Tag pointing to the first commit
```

tag object type contains

- the hash of the tagged object,
- the type of tagged object (usually a commit),
- the tag name, author, date, and message

TAG OBJECT – DATA STRUCTURE



EDIT FILE CONTENT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git show :index.txt
Hello World

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ echo "Welcome to the Programming World" > index.txt

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git commit -am "second commit"
warning: in the working copy of 'index.txt', LF will be replaced by CRLF the next time Git touch
es it
[master ec06aec] second commit
 1 file changed, 1 insertion(+), 1 deletion(-)
```

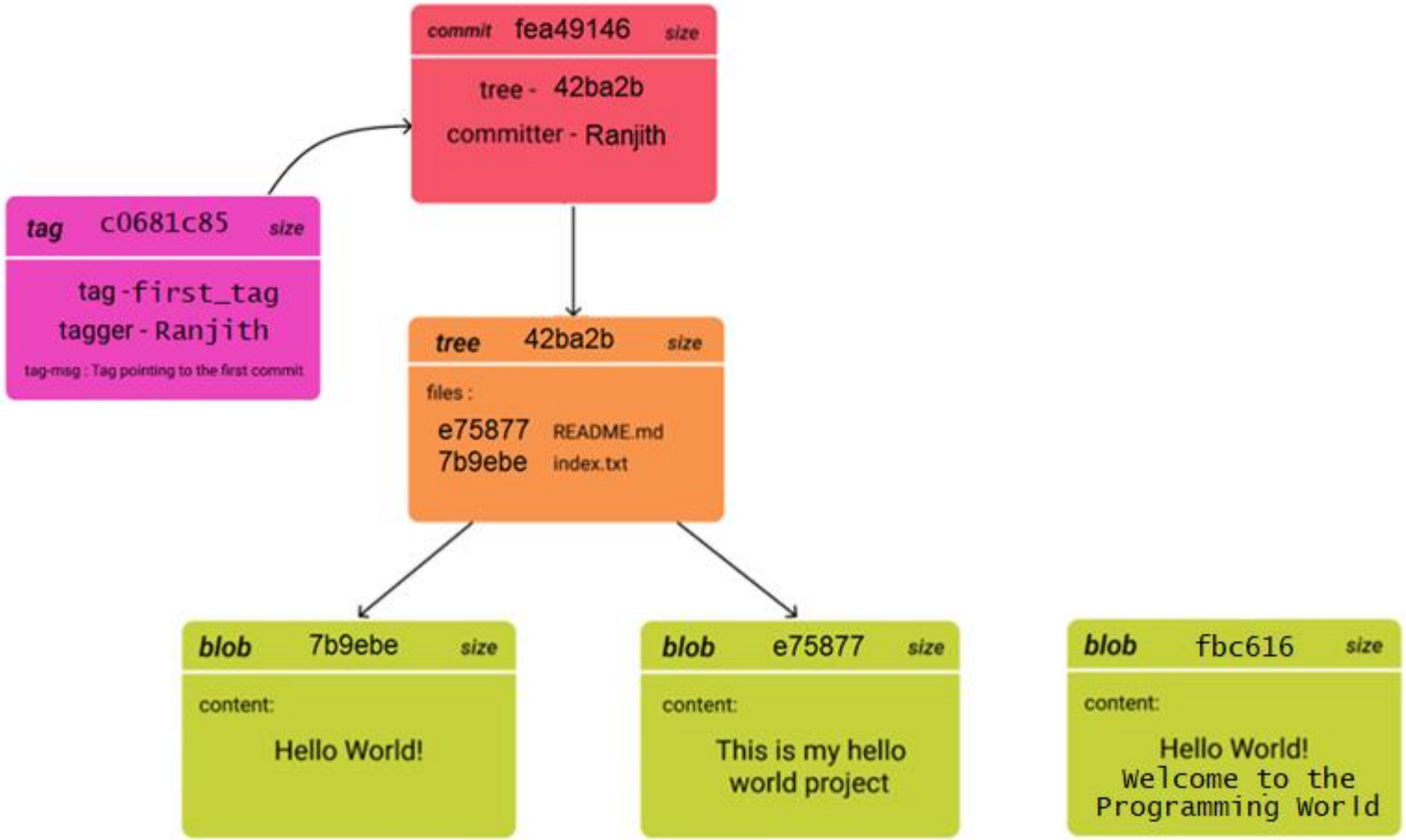
EDIT FILE CONTENT – CHANGE IN OBJECT CONTENT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/
0a/  42/  7b/  c0/  e7/  ec/  fb/  fe/  info/  pack/
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/fb
c6163eedbf1039a3ee6cfdacc260697a3c89cb

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t fbc6163eedbf1039a3ee6cfdacc260697a3c89cb
blob
```

git has now created 3 new objects for the second commit

EDIT FILE CONTENT – CHANGE IN OBJECT CONTENT

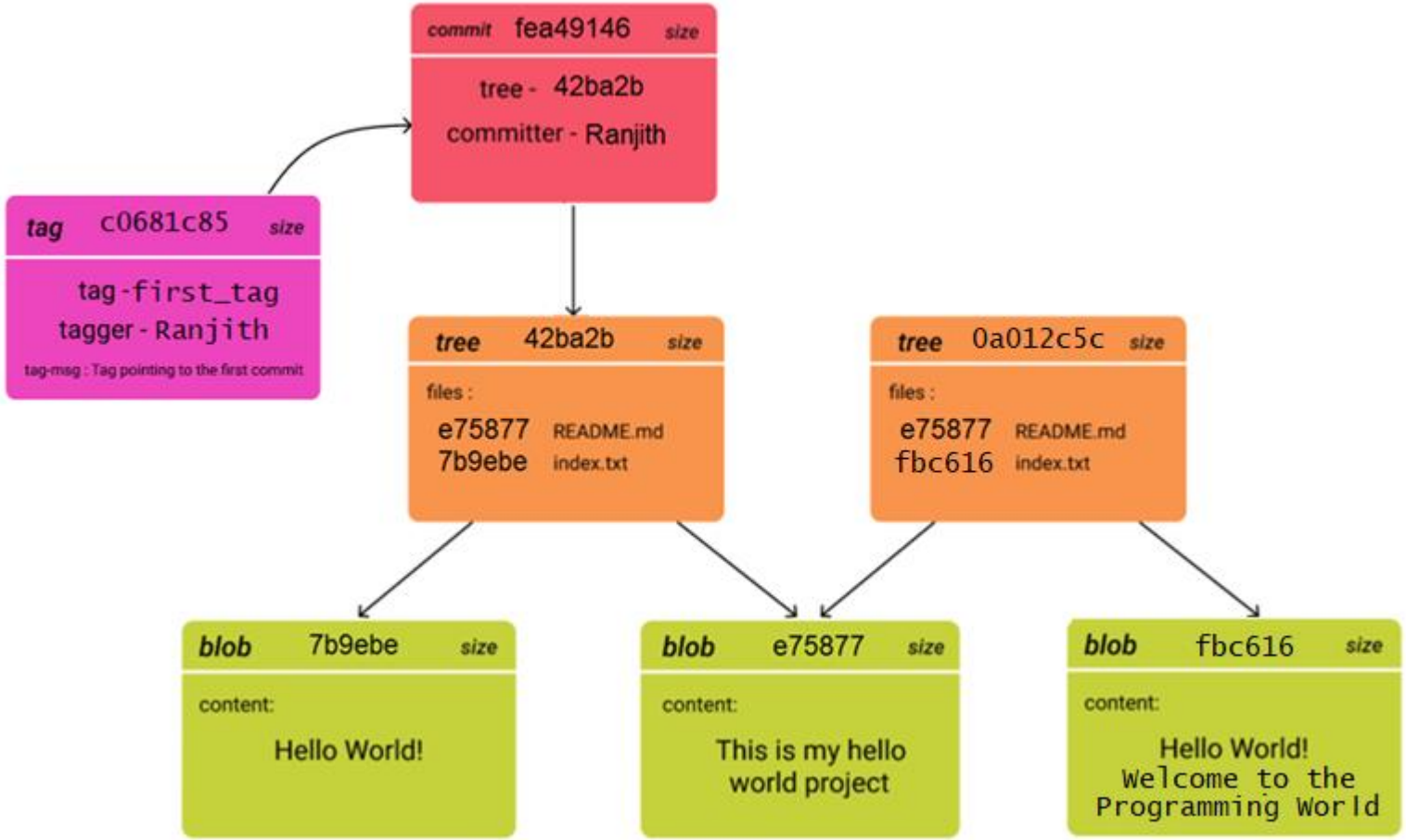


EDIT FILE CONTENT – CHANGE IN OBJECT CONTENT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/
0a/  42/  7b/  c0/  e7/  ec/  fb/  fe/  info/  pack/
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/0a
012c5cfd19d814ee1baaaf9a968d1b070ca0ea

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t 0a012c5cfd19d814ee1baaaf9a968d1b070ca0ea
tree
```

EDIT FILE CONTENT – CHANGE IN OBJECT CONTENT

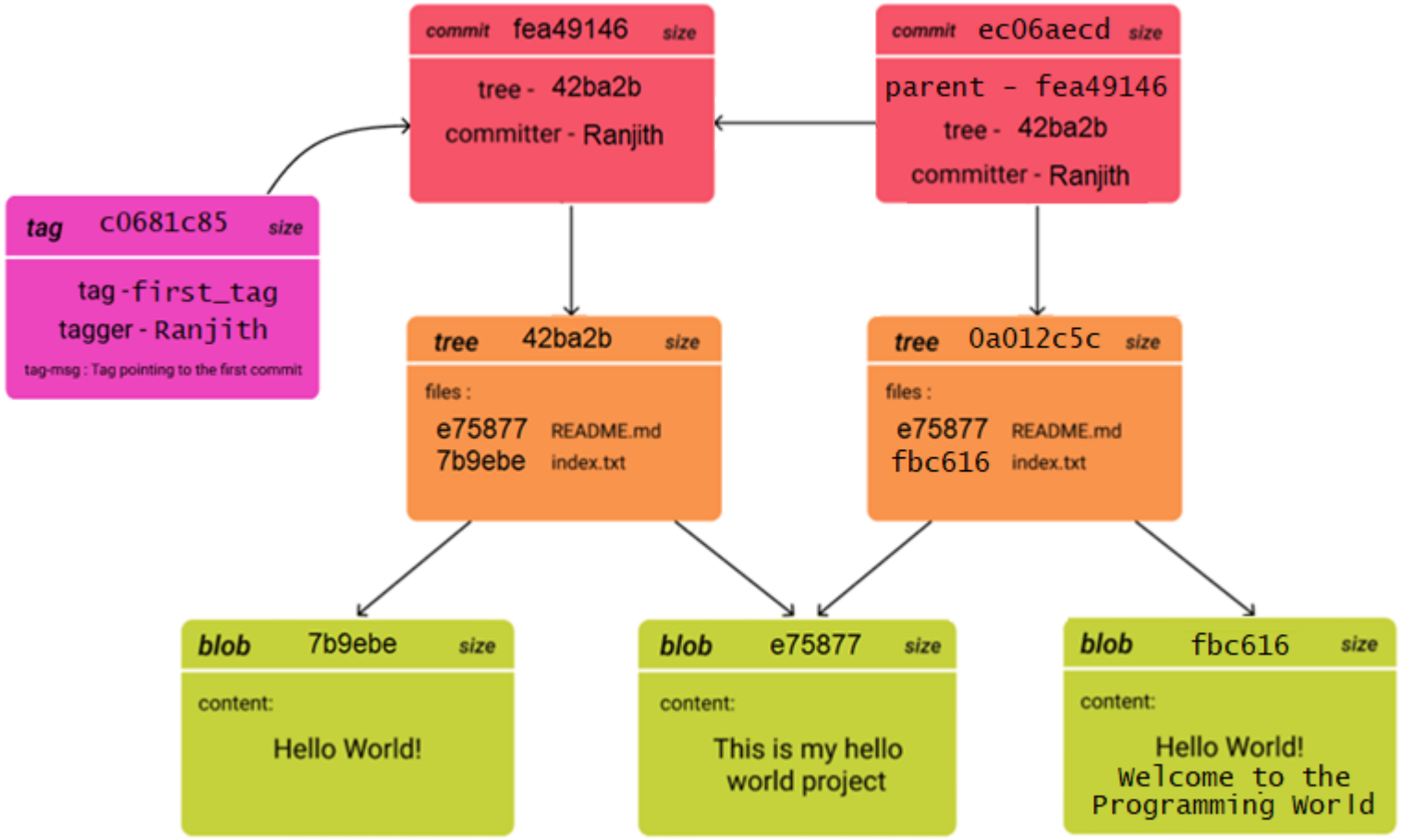


EDIT FILE CONTENT – CHANGE IN OBJECT CONTENT

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/
0a/  42/  7b/  c0/  e7/  ec/  fb/  fe/  info/  pack/
```

```
rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ ls .git/objects/ec
06aecda5bcbacb1c8f0cb14f7e00c63d8eb8ee

rranj@LAPTOP-FBEQTLF3 MINGW64 /e/delete/git_try (master)
$ git cat-file -t ec06aecda5bcbacb1c8f0cb14f7e00c63d8eb8ee
commit
```

EDIT FILE CONTENT – CHANGE IN OBJECT CONTENT



FIND ANSWERS

- Tracked files
- Untracked files
- Ignored files
- How to create branch and create sync between local and remote repositories?