Basic Commands

●

Man Pages

```
SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...
```

- Words in CAPS, <angle> or *italics* usually means they need to be substituted with something
- SOURCE, DEST need to be pathnames to files
- Words or options enclosed in [ ] are optional
- A | separating words indicates that they are mutually exclusive
  - [-r|-s]

    -r and -s are both optional but only one can be used at a time
- An ellipsis … following means more than one can be specified
  - SOURCE… means one or more source files

Pathnames

●

Relative and Absolute Pathnames

- Relative pathnames do not begin with a `/`
  - Are relative to the Current Working Directory
- Absolute pathnames always begin with `/`
  - Or relative to the root directory (/)
- All (even empty) directories have two special subdirectories
  - `.` is the directory itself
  - `..` is the parent of that directory

```
$ pwd
/home/ebasheer

$ cd Documents
$ pwd
/home/ebasheer/Documents

$ cd ..
$ pwd
/home/ebasheer
```

Essential Commands

Examples

- `pwd`
  Print current working directory to stdout
- `cd`
  Change the current working directory
- `mkdir`
  Create a directory
- `cp`
  Copy files and directories
- `mv`
  Move/rename files
- `rm`
  Remove/Delete files or directories

File Permission Bits

chmod

Change file mode

- Example Output of `ls -l`

```
$ ls -l
total 20
drwxrwxr-x 8 ebasheer ebasheer 4096 Aug 16 22:49 central
drwxrwxr-x 7 ebasheer ebasheer 4096 Aug 16 21:49 central.bak
-rwxrwxr-- 1 ebasheer ebasheer    0 Aug 23 21:40 convert.sh
-rw-rw-r-- 1 ebasheer ebasheer    0 Aug 23 21:39 inputexample.txt
drwxrwxr-x 3 ebasheer ebasheer 4096 Aug 16 22:47 myproject
drwxrwxr-x 3 ebasheer ebasheer 4096 Aug 16 22:43 myprojectclone
drwxrwxr-x 2 ebasheer ebasheer 4096 Aug 23 17:13 tmpdir
```

File Permission Bits


```
$ chmod u-rx script.sh
$ ls -l
total 0
--w-rwx--x 1 ebasheer ebasheer 0 Aug 23 20:46 script.sh

$ cat script.sh
cat: script.sh: Permission denied

$ chmod u+rx,g-w,o+r script.sh
$ ls -l script.sh
-rwxr--r-- 1 ebasheer ebasheer 0 Aug 31 11:24 script.sh
```

Essential Commands

Examples

- `echo`
  Write a line of text to stdout
- `cat`
  Print contents of file to `stdout`
- `find`
  Search for files
- `less`
  A “pager”. Text file viewer with scrolling and search
- `gzip, gunzip`
  Compress/Uncompress files

Essential Commands

•

Examples

- `sort`
  Sort lines of text files
- `uniq`
  Remove repeated lines
- `wc`
  Print line, word, byte counts
- `head, tail`
  Print first/last lines of a file
- `grep`
  Print lines in a file that match a pattern

Unix Toolbox Philosophy

Unix Toolbox

The Unix Toolbox Philosophy

See `info coreutils`

```
* Output of entire files::       cat tac nl od base32 base64
* Formatting file contents::     fmt pr fold
* Output of parts of files::     head tail split csplit
* Summarizing files::            wc sum cksum md5sum sha1sum sha2
* Operating on sorted files::    sort shuf uniq comm ptx tsort
* Operating on fields::          cut paste join
* Operating on characters::      tr expand unexpand
* Directory listing::            ls dir vdir dircolors
* Basic operations::             cp dd install mv rm shred
* Special file types::          mkdir rmdir unlink mkfifo mknod ln link readlink
* Changing file attributes::     chgrp chmod chown touch
* Disk usage::                   df du stat sync truncate
* Printing text::                echo printf yes
* Conditions::                   false true test expr
* Redirection::                  tee
* File name manipulation::       dirname basename pathchk mktemp realpath
* Working context::              pwd stty printenv tty
* User information::             id logname whoami groups users who
* System context::               date arch nproc uname hostname hostid uptime
* SELinux context::              chcon runcon
* Modified command invocation::  chroot env nice nohup stdbuf timeout
* Process control::              kill
* Delaying::                     sleep
* Numeric operations::           factor numfmt seq
```

Shell

Pipes

The | character connects two commands in the shell

- `cmd1 | cmd2`
  Connects stdout of `cmd1` to stdin of `cmd2`
- Many commands accept input from stdin by default
- Using `cat`, `sort` and `uniq` in a pipeline

Shell

Pipes

```
# Examples
$ cat cities.txt | sort | uniq
$ cat cities.txt | nl | sort -sk2 | uniq -f1 | sort -nk1 | cut -f2
$ cat randint.txt |grep '2$'|wc -l
```

Redirection

Redirect a standard stream to/from a file

- `cmd1 > file.txt`
  Redirects `stdout` of cmd1 to `file.txt`
- To write the output of our pipeline to a file

```
$ cat cities.txt | sort | uniq > uniqcities.txt
```

- `cmd1 < file.txt`
  Redirects `file.txt to stdin` of `cmd1`

```
$ sort < cities.txt
$ sort < cities.txt | uniq
$ sort cities.txt | uniq
```

Editing Text Files

Editing Text Files

Text Editors

- Terminal
  - `Vim`
  - `Emacs`
  - `Nano`
  - `Pico`
  - `...`
- GUI
  - `Gedit`
  - `Kate`
  - `...`

Editing Text Files

Vim

- Vim is a modal editor
  Command and text input mode
- Command mode: default mode when vim starts
  - Movement with `h, k, j, l` (or arrow keys)
  - Other key perform various functions. e.g
    - Delete a line, character or word
      `dd, x, dw`
    - Copy a line, character or word to a register (clipboard)
      `yy, yl, “ayw`
    - Paste
      `P, p`
    - Undo, Redo
      `u , Ctrl-r`
    - Combination of commands or commands repeated a number of times

Vim

- Input mode
  - `i, a, o` enter insert mode
  - Esc to return to command mode

Editing Text Files

Vim

Ex commands

Begin with a colon :

- Save file
  `:w` in command mode
- Quit Vim
  `:q` in command mode
- Quit discarding modifications
  `:q!` In command mode

type `vimtutor` on command line
and follow the tutorial

Introduction to Linux

- Chapters 2, 3, 5, 6

Introduction to git

Scenario

Begin a project with `test.py`

Make changes to the file

test a new feature for example

But want to keep the older version

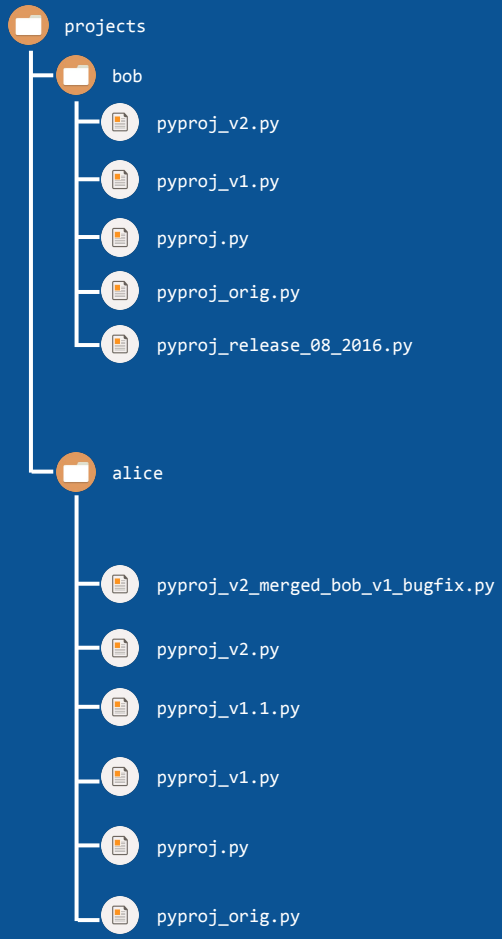
so that we can revert to it if we need to

Version numbers added as the project progresses and collaborators join

At each step a copy is made to keep history

Quickly becomes hard to manage

And easy to make mistakes



Version Control Systems

Version Control Systems

Basic Function

- Maintain a history of changes to files
- Allow retrieval of any previous revision
- Each change is recorded along with information that allows us to answer the questions
  - When was the change made
  - Who made the change
- Serves as documentation of progress

Version Control Systems

Well Known Tools

CVS
SVN (subversion)
git
Mercurial
Bazaar
LibreSource
...

git

git is a VCS created and developed by Linus Torvalds and the Linux kernel community as an alternative to a proprietary system they were using until 2005

- Distributed Development
- Built-in integrity
- Fast
- Free and Open Source

Setting your identity

```
$ git config --global user.name "Ershaad Basheer"
$ git config --global user.email ebasheer@example.com
```

Choosing an editor

```
$ git config --global core.editor vim
```

Viewing settings

```
$ git config --list
```

help command

`git help <command>`

```
$ git help config
$ git help commit
```

man pages

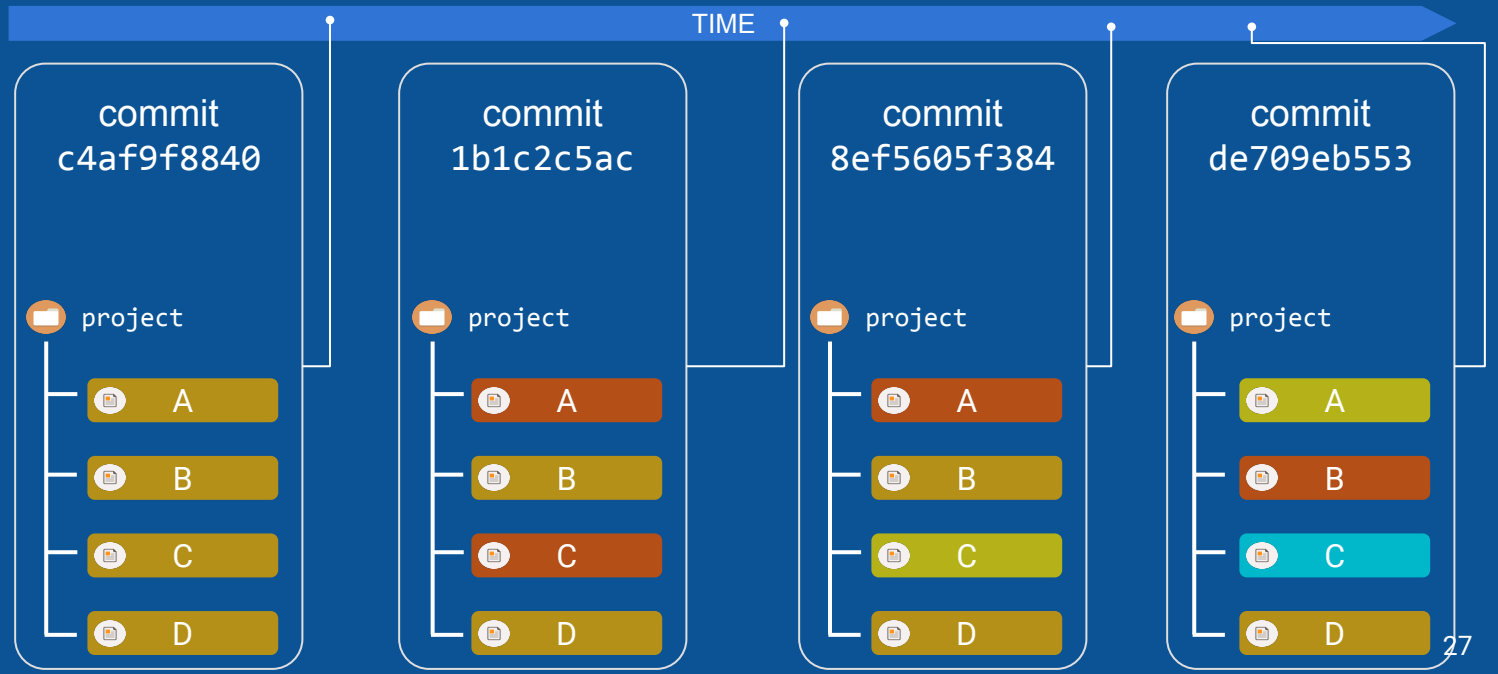
`man git-<command>`

```
$ man git-config
$ man git-commit
```

git

Terminology

- Commit
  - Snapshot of what the entire project looked like at a point in time
- Repository
  - All the files that are under version control and the complete history of commits



- Initializing a Repository
  - In an directory
  - Empty directory in our example

```
~/work/report$ git init
```

- git shows that no files are being tracked

```
~/work/report$ git status
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

- We can create or copy files

```
$ ls
expenses.csv
```

- git indicates that there are untracked files

```
$ git status
On branch master

Initial commit

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

      expenses.csv

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

- We add the files to the index
  - Required step before tracking changes to a new file

```
~/work/report$ git add expense.csv
```

- Git now shows us that a new file has been added to the index

```
$ git status
On branch master

Initial commit

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

      new file:   expenses.csv
```

- We still do not have any snapshots in our history

```
~/work/report$ git log
fatal: your current branch 'master' does not have any commits yet
```

- We create our first commit

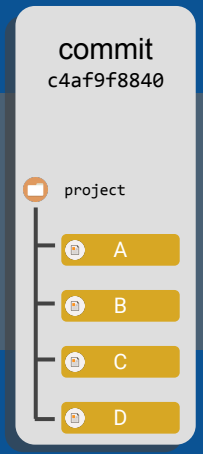
```
~/work/report$ git commit -m "initial commit"
[master (root-commit) 05751d1] initial commit
 2 files changed, 7 insertions(+)
 create mode 100644 expense1.csv
 create mode 100644 expense2.csv
```

- Since we haven't edited any files since the last commit
  - git status has nothing interesting to output

```
$ git status
On branch master
nothing to commit, working directory clean
```

- And our repository now has one commit
  - Copy of the state of files in our project saved in history

```
~/work/report$ git log
commit d8f0fcfda2d09d437cdbd339db11cf8b6365508f
Author: Ershaad Basheer <ebasheer@temple.edu>
Date:   Tue Aug 9 12:46:15 2016 -0400

    initial commit
```



- After editing one of the files
  - Make changes using a text editor
  - Or any other program (Like a word processor, etc)
  - Git doesn't care how the file is edited
- Git shows that one of the files has been modified

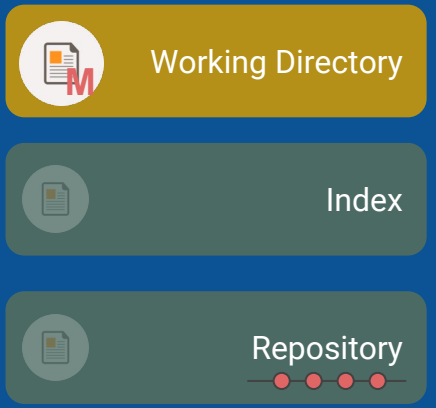
```
~/work/report$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      modified:   expenses.csv

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

- Before we can commit the changed file to the repository
  - We need to add it to the index. or *stage* the changed file

git

Staging a File

- The modified file now exists in the “Working Directory”

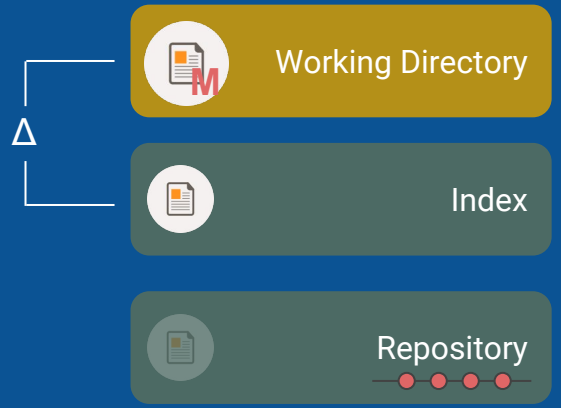


git

Staging a File

- To view the differences between what's in the working directory and the index

```
~/work/report$ git diff expenses.csv
diff --git a/expenses.csv b/expenses.csv
index 1885f91..29f8e17 100644
--- a/expenses.csv
+++ b/expenses.csv
@@ -1,4 +1,4 @@
 CATEGORY,BUDGET,CURRENT
-Groceries,1000,150
+Groceries,1001,150
 Electric,100,66
 Utilities,250,25
```



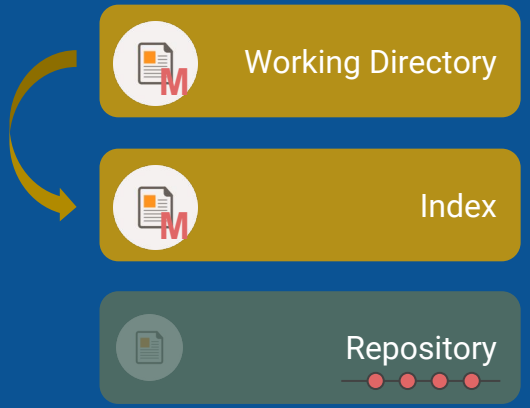
git

Staging a File

- We use git add once again to stage the file

```
~/work/report$ git add expenses.csv
```

- Now the modified file exists both in the working directory and the Index

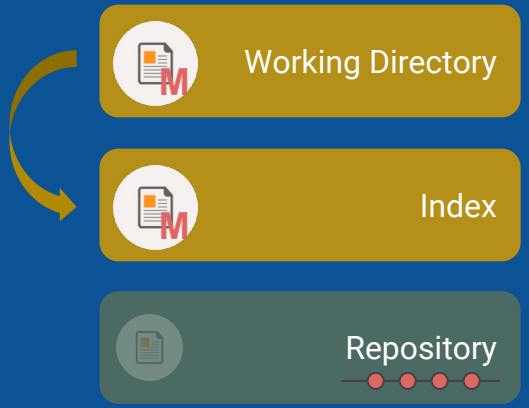


git

Staging a File

- Git tells us that the working directory and the index are identical

```
~/work/report$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

      modified:   expenses.csv
```



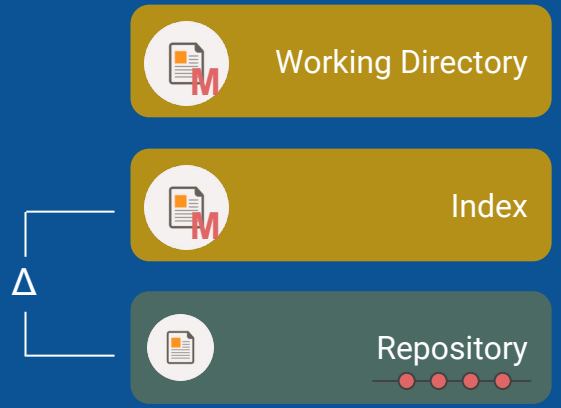
- But, they differ from the last commit in the repository

git

Staging a File

- To view the differences between what's in the index and the last commit

```
~/work/report$ git diff --staged
diff --git a/expenses.csv b/expenses.csv
index 1885f91..29f8e17 100644
--- a/expenses.csv
+++ b/expenses.csv
@@ -1,4 +1,4 @@
 CATEGORY,BUDGET,CURRENT
-Groceries,1000,150
+Groceries,1001,150
 Electric,100,66
 Utilities,250,25
```

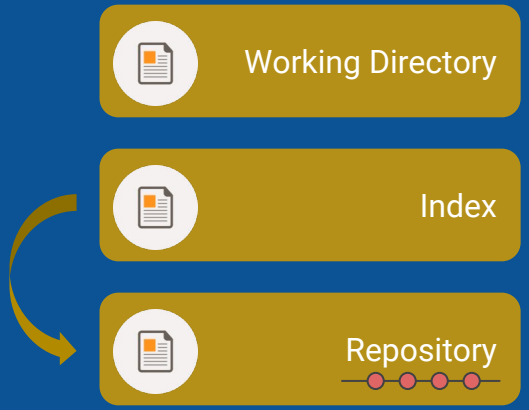


git

Commit

- When we're satisfied with what is in the index
  - We're ready to make a commit

```
~/work/report$ git commit -m "Update grocery\
budget"
[master cbbb2de] Update grocery budget
 1 file changed, 1 insertion(+), 1 deletion(-)
```



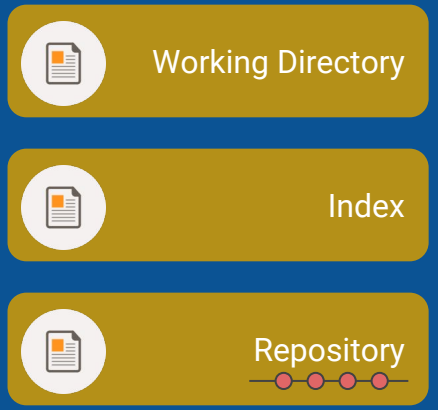
- Takes what is in the index and permanently stores it in the repository as a commit

git

Commit

- Working directory, index and last commit have the same content

```
$ git status
On branch master
nothing to commit, working directory clean
```



git

Commit History

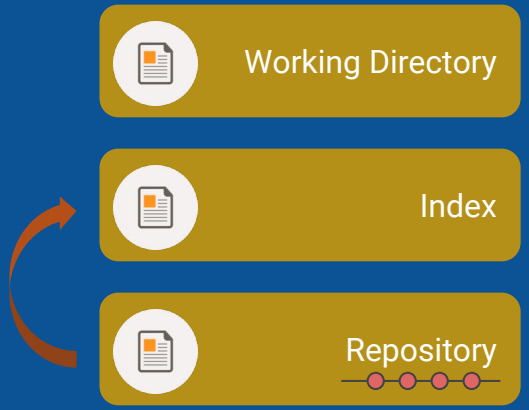
- We now have two snapshots of our project in the repository
  - Each identified by a commit hash
- At all times, we only see two files in our working directory
  - Git takes care of all the moving around of data to maintain revision history behind the scenes

```
~/work/report$ git log
commit cbbb2de252199245c4dc07b51d247ff90f997e4c
Author: Ershaad Basheer <ershaad.a@gmail.com>
Date:   Wed Aug 31 17:10:36 2016 -0400

    Update grocery budget

commit e884dd46443fed2fb22b101e8b373f23ee45e093
Author: Ershaad Basheer <ershaad.a@gmail.com>
Date:   Wed Aug 31 16:56:12 2016 -0400

    initial commit
```

- Unstaging a file
  - You staged a file for commit but changed your mind

```
~/work/report$ git reset HEAD expenses.csv
```

- Takes the version of the file in ref HEAD which is the last commit and copies it into the index
- Now the index is identical to the last commit

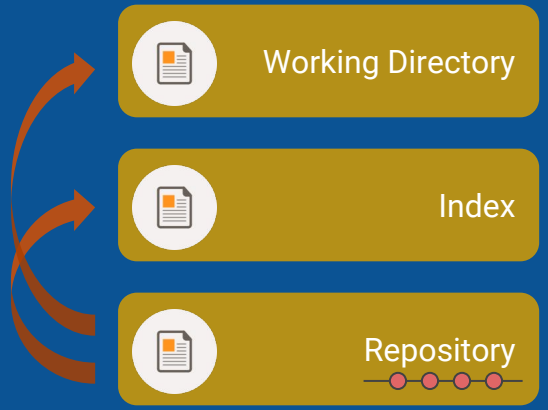


- Discarding modifications in the working directory
  - ⚠ Overwriting files in the working directory is dangerous
    - Cannot be undone

```
~/work/report$ git checkout expenses.csv
```

- Reverting to a version of a file from a previous commit
  - Copies the file from the commit to the index
  - And the working directory

```
~/work/report$ git checkout e884dd46 expenses.csv
```



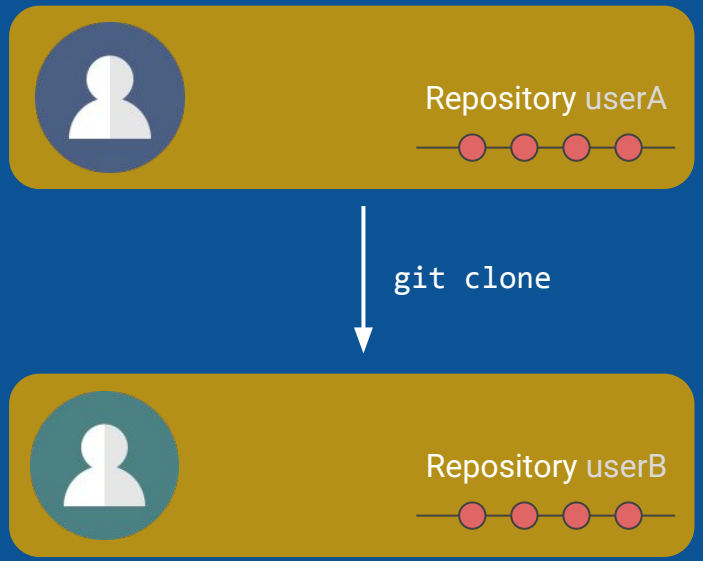
Collaboration in git

git

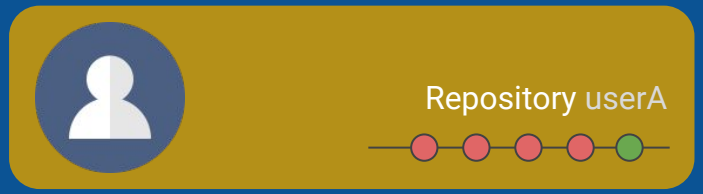
Collaboration in git

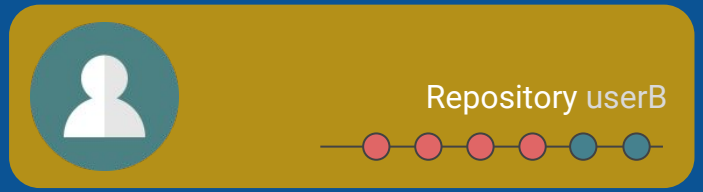
- git is a distributed version control system by design
- Collaborators work on a local copy of a repository

- Local repositories begin as a “clone” of another local repository



- Collaborators work on a local copy of a repository
  - Make changes and commit them to this local repository
    - Building a local history of commits

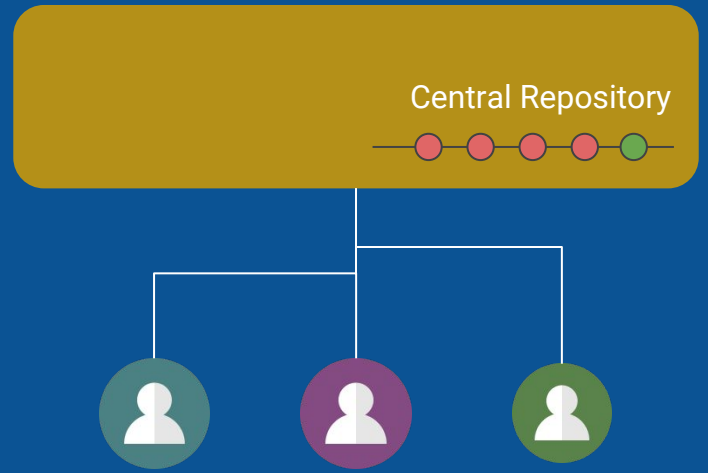




git

Simple Workflow

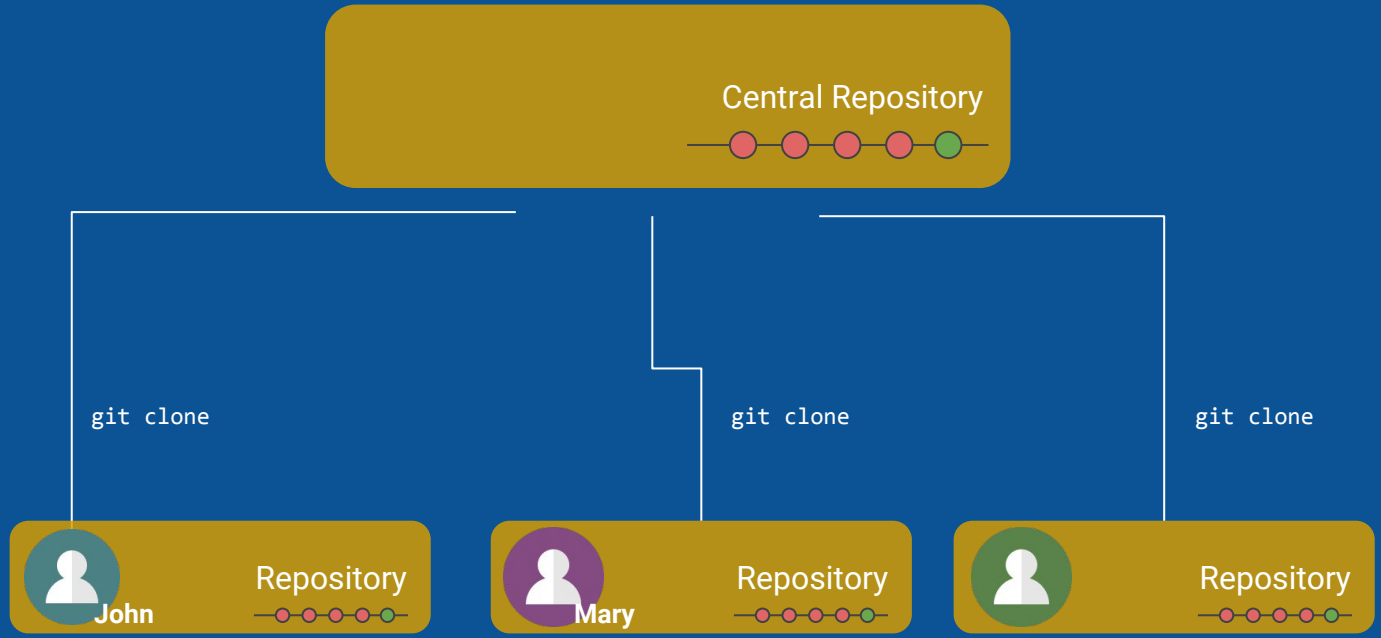
- One repository is designated as the “central” repository
  - Could be on a web-based Git repository hosting service like GitHub or Bitbucket



git

Cloning

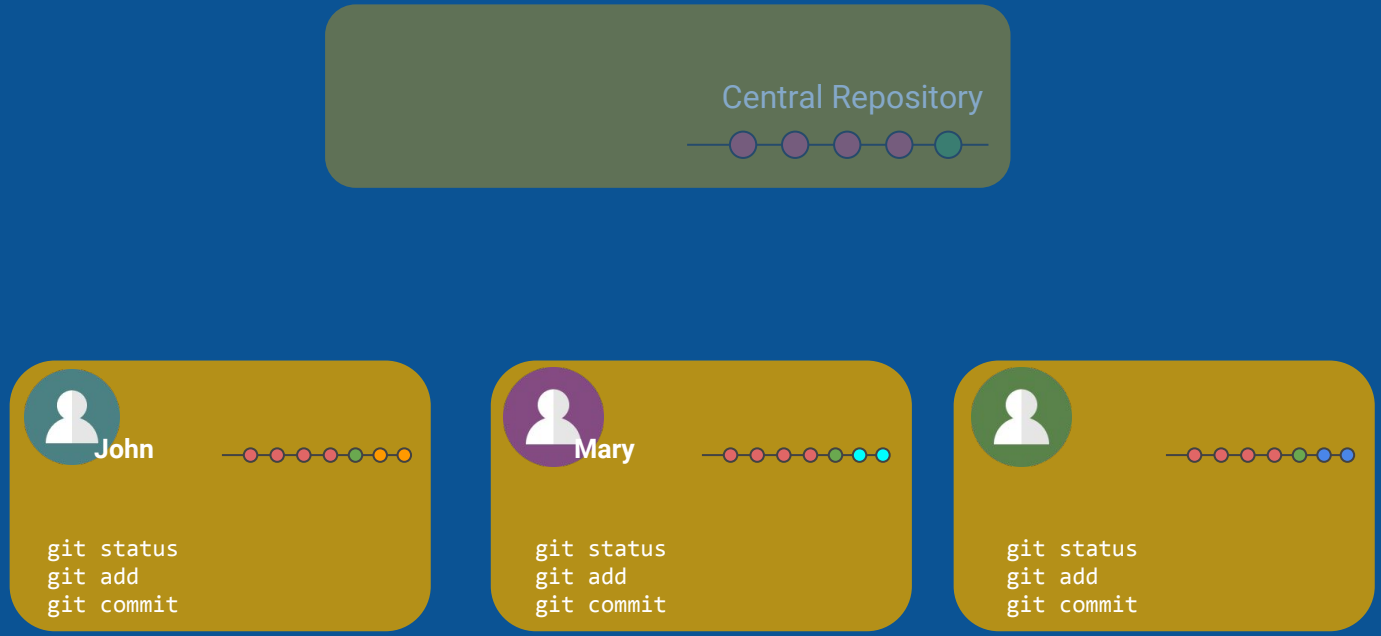
- Developers begin by cloning the central repository
- When cloning, git includes a reference to the “parent” repository called `origin`



git

Local repositories

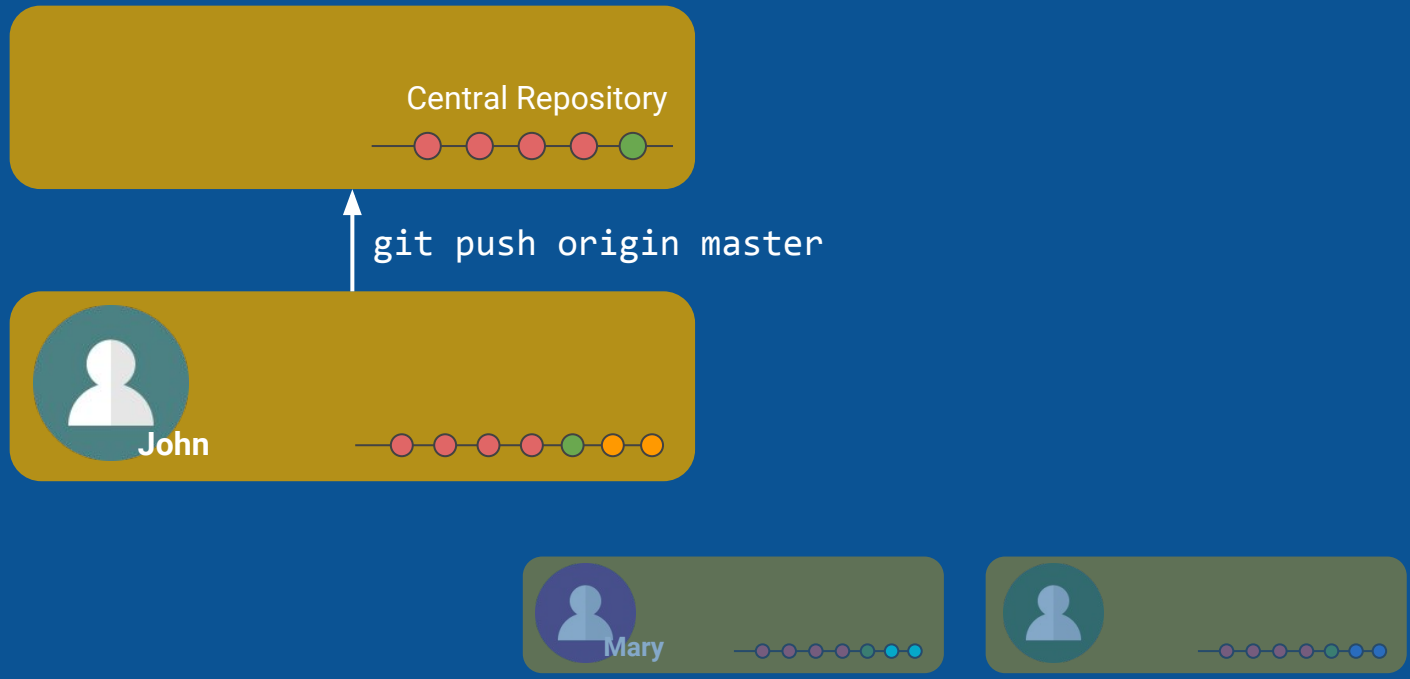
- Developers edit files and commit changes to their *local* repository
  - Oblivious to what other are doing in their repositories,
  - Or the central repository
  - Until they decide to synchronize upstream



git

Pushing to the Central Repository

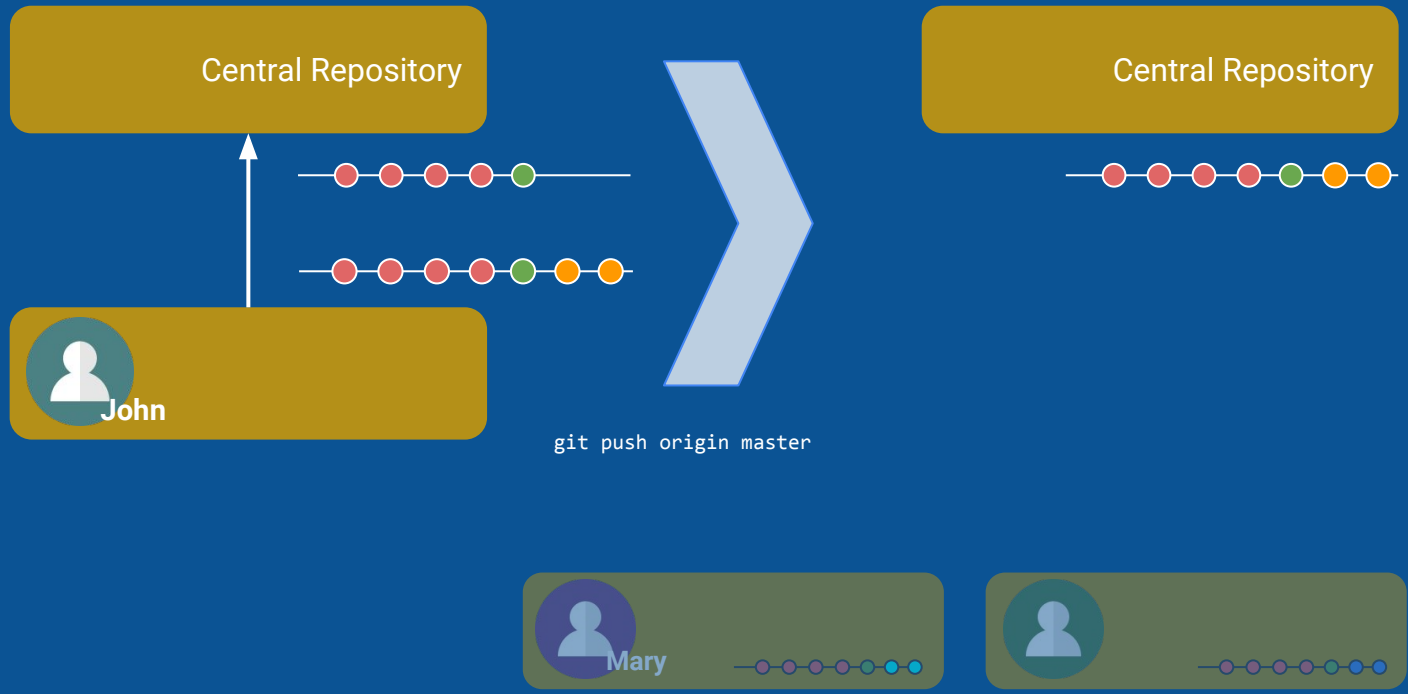
- This is done with a `git push`



git

Pushing to the Central Repository

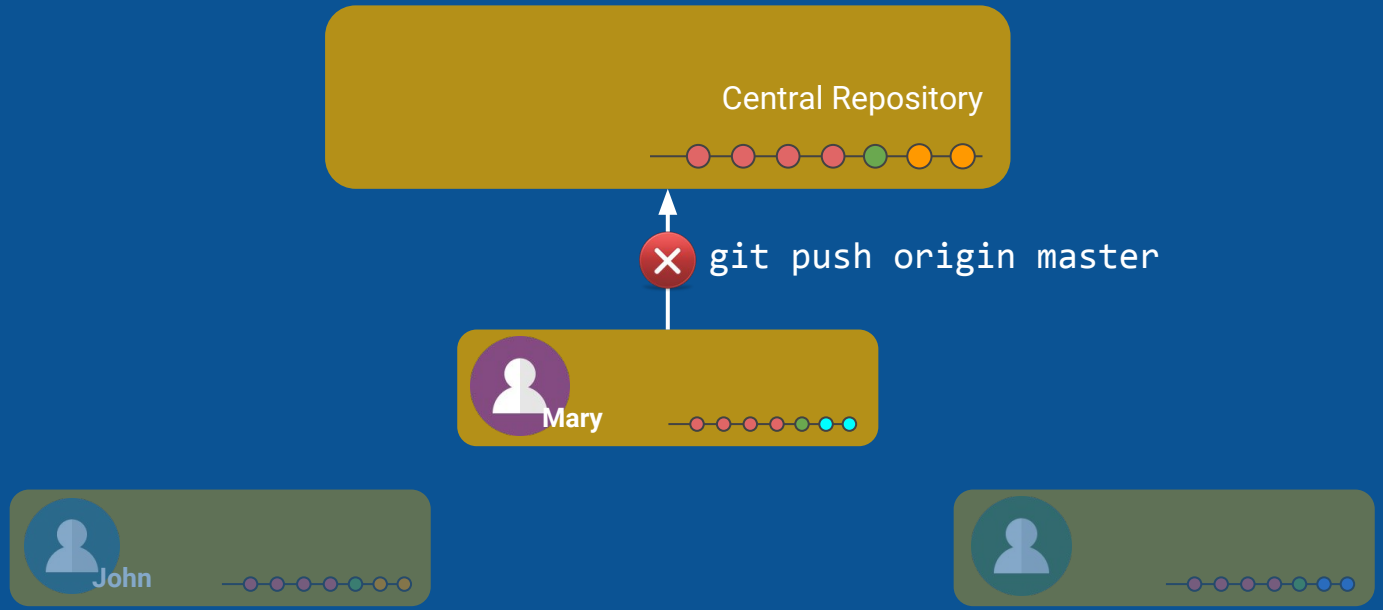
- Git detects that the central repository hasn't changed since John cloned it, so
- Commits can be directly merged into the central repository



git

Pushing to the Central Repository

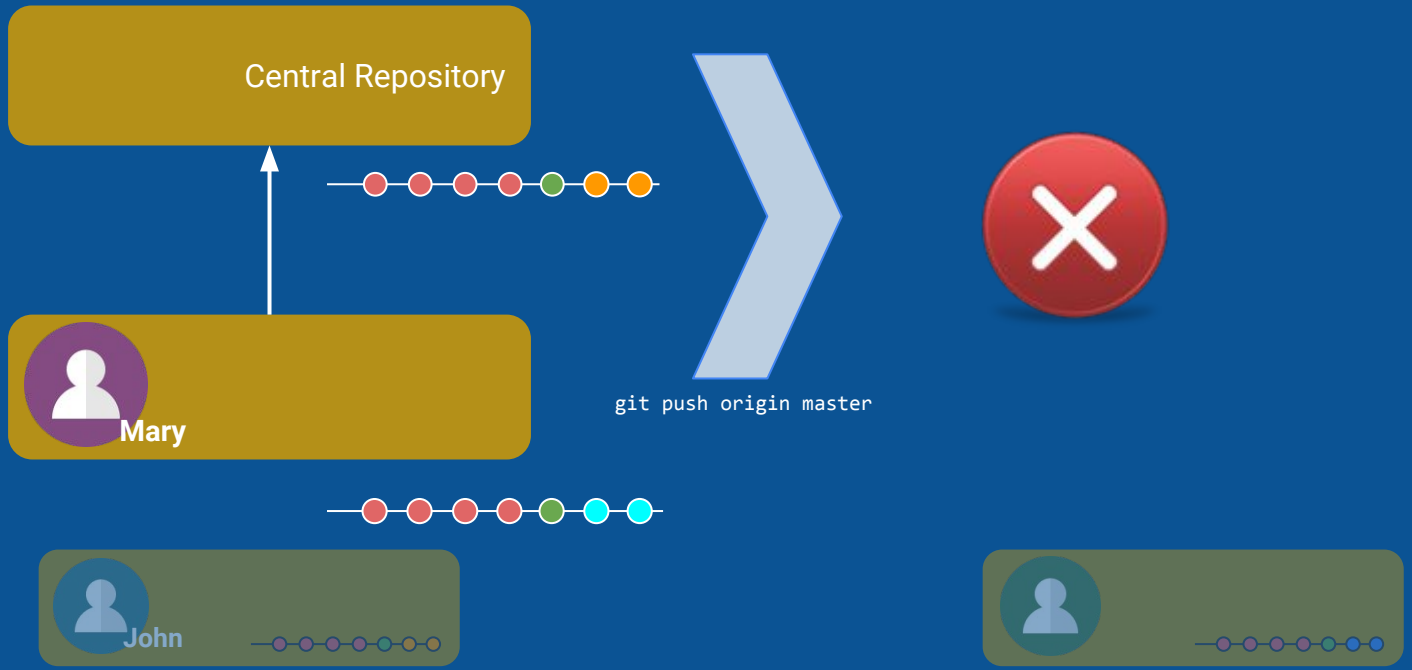
- Now, developer Mary decides it’s time to push her changes to the project
- But this will fail



git

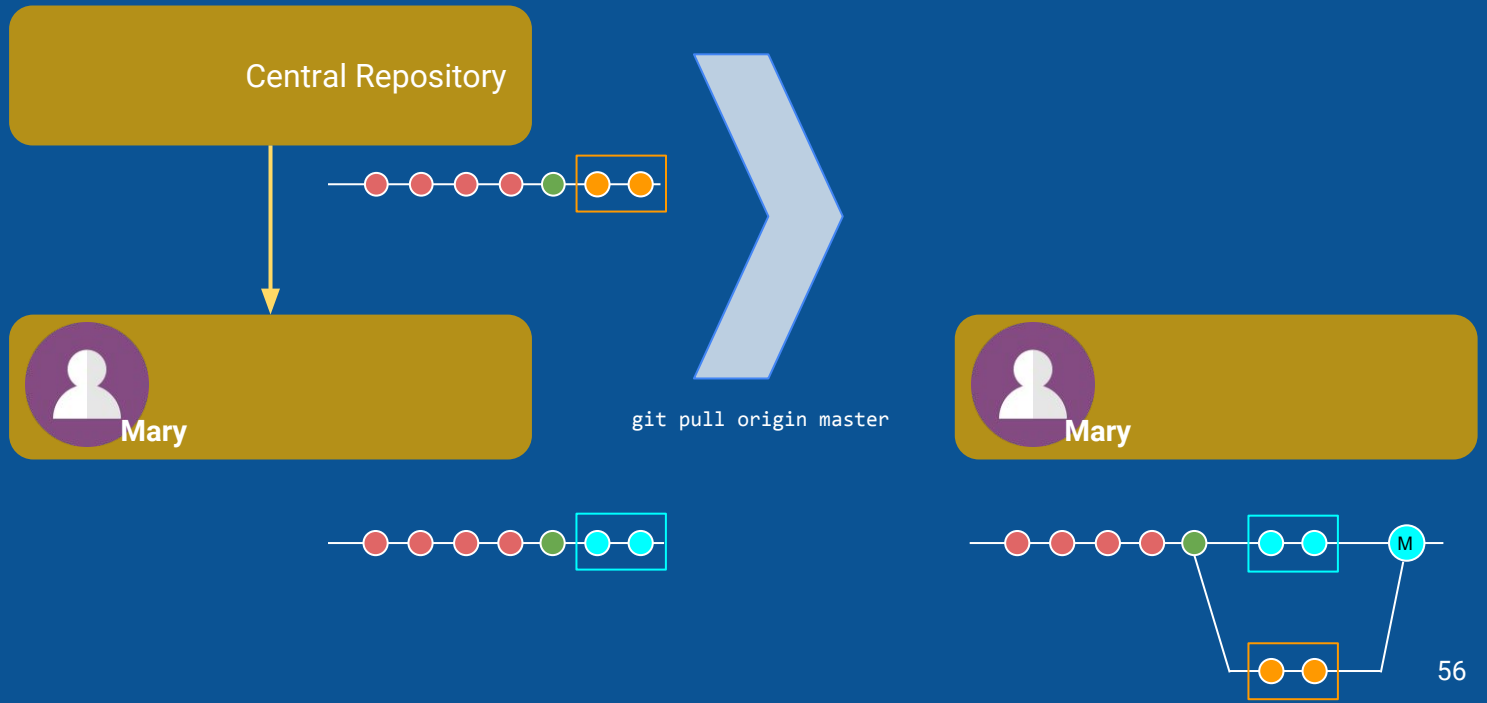
Pushing to the Central Repository

- Git determines that the central repository has changed since it was cloned
- And the commit histories therefore diverge
- Git refuses to merge changes into the central repository



Pulling changes

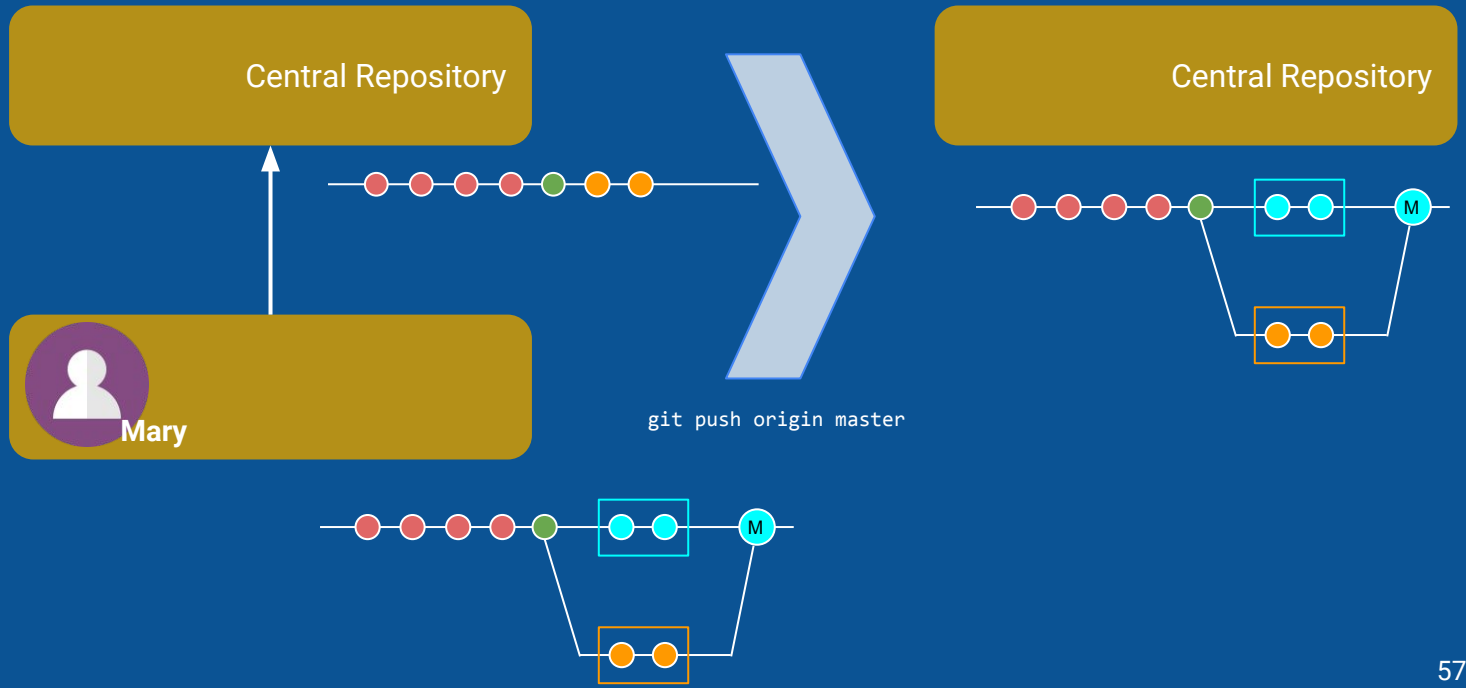
- Before pushing commits
- Upstream commits in the central repository need to be merged into the local repository with `git pull`. Creates a “merge commit”
- Can raise a merge conflict



git

Pushing to the Central Repository

- Now that the commit history of the central repository has been merged locally
- Mary will be ready to push her commits



Git Book

- Chapters 1 and 2