git

Teachers

Roberto Esposito and **Rossano Schifanella**

a.a. 2021/22

About Version Control

https://www.git-scm.com/book/en/v2/Getting-Started-About-Version-Control

About Version Control

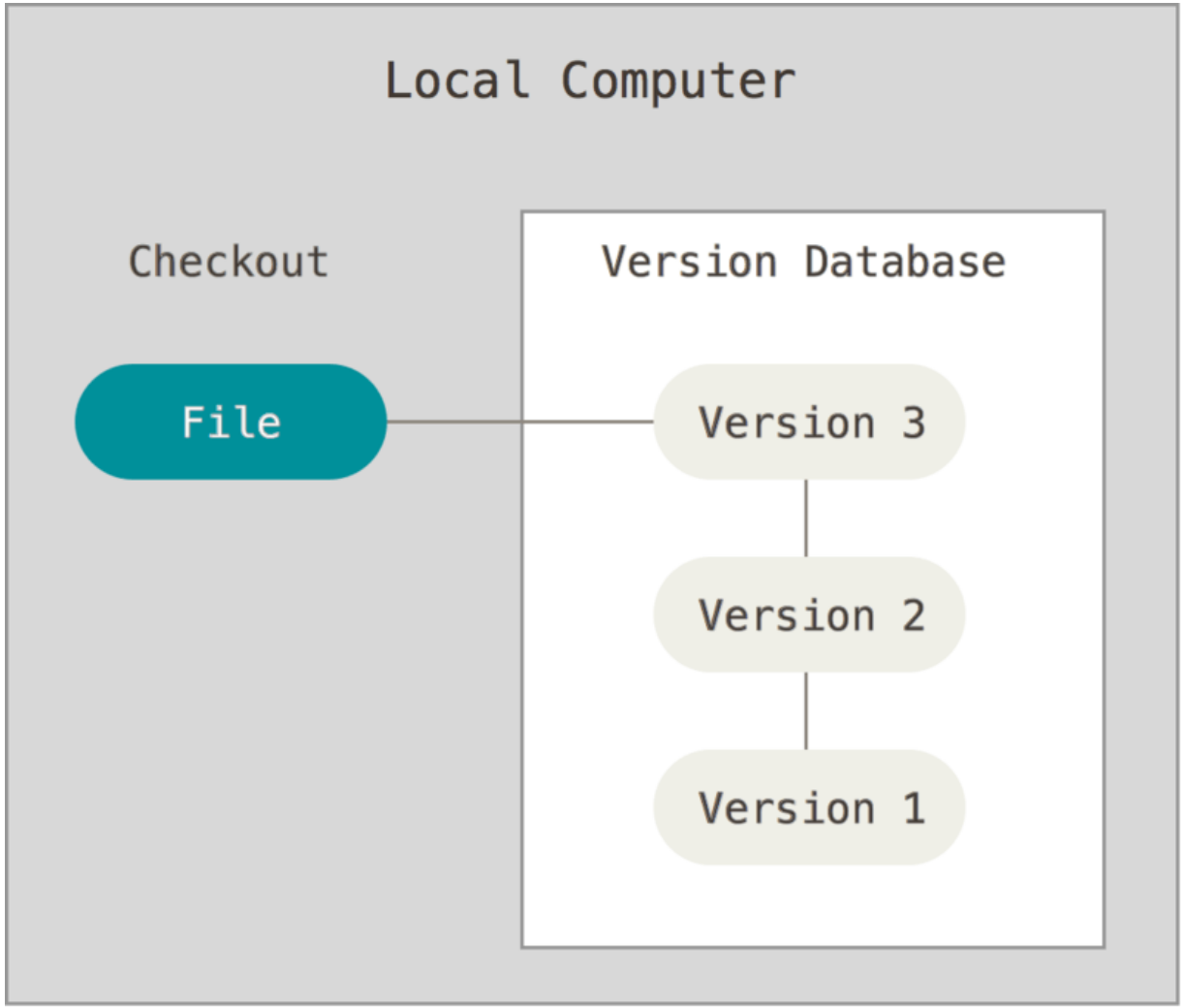
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

This approach is very common because it is so simple, but it is also incredibly error prone.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

Local Version Control Systems

Many people's version-control method of choice is to copy files into another directory.

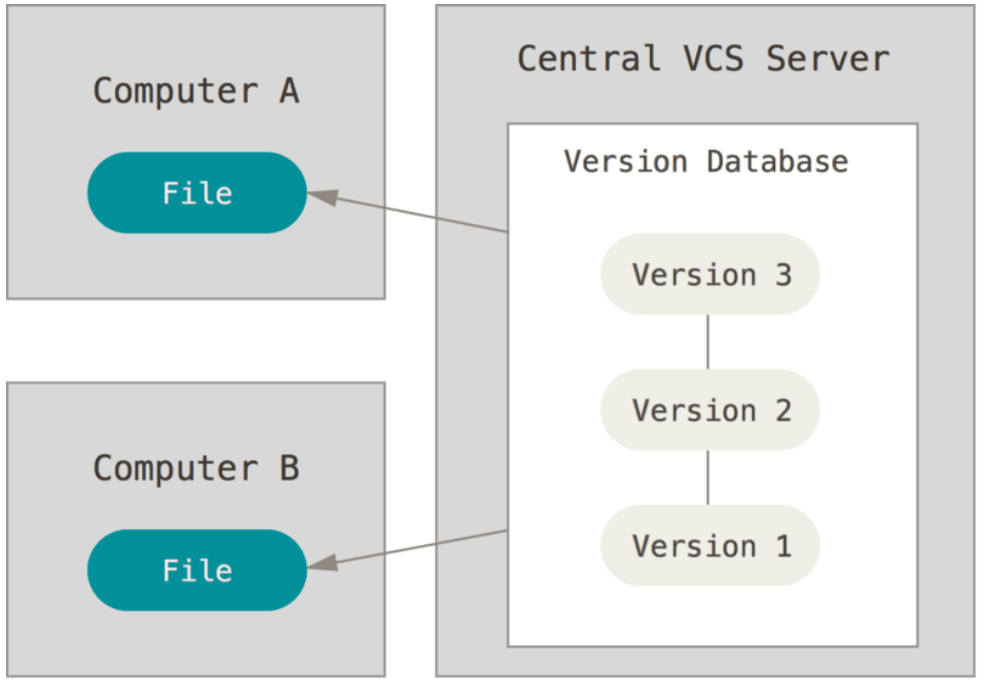


Centralized Version Control Systems

The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed.

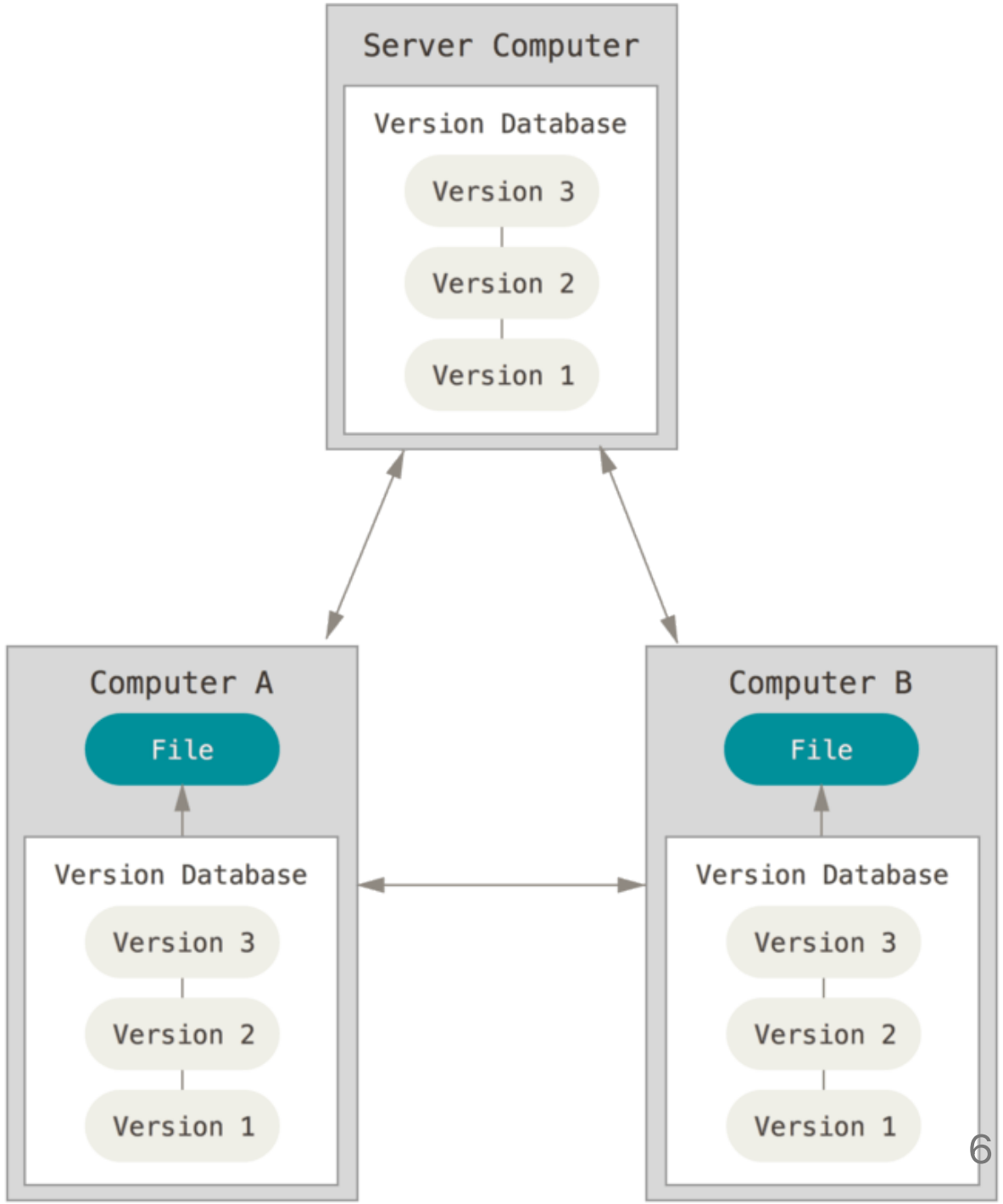
The most obvious problem with centralized VCS is the single point of failure that the centralized server represents.

- server can go down
- the hard disk the central database can become corrupted
- ...



Distributed Version Control Systems

- solve the above mentioned problems
- allows you to set up several types of workflows that aren't possible in centralized systems, such as hierarchical models.



First-Time Git Setup

https://www.git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

git config

Git comes with a tool called `git config` that lets you get and set configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places:

1. `[path]/etc/gitconfig` If you pass the option `--system` to `git config`, it reads and writes from this file specifically.
2. `~/.gitconfig` or `~/.config/git/config` file: Values specific personally to you, the user. Use the `--global` option.
3. `config` file in the Git directory (that is, `.git/config`) of whatever repository you're currently in. This is the default.

Each level overrides values in the previous level, so values in `.git/config` trump those in `[path]/etc/gitconfig`.

Introducing yourself to `git`

Your Identity

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Your Editor

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

Your Editor -- On windows -- 🤦

```
$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' \
-multiInst -notabbar -nosession -noPlugin"
```

Introducing yourself to `git`

Your Default Branch Name

```
$ git config --global init.defaultBranch main
```

Checking your settings

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

Getting a Git Repository

https://www.git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Initializing a Repository in an Existing Directory

```
$ git init
```

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial project version'
```

Cloning an Existing Repository

```
$ git clone https://github.com/libgit2/libgit2
```

If you want to name the directory something different:

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

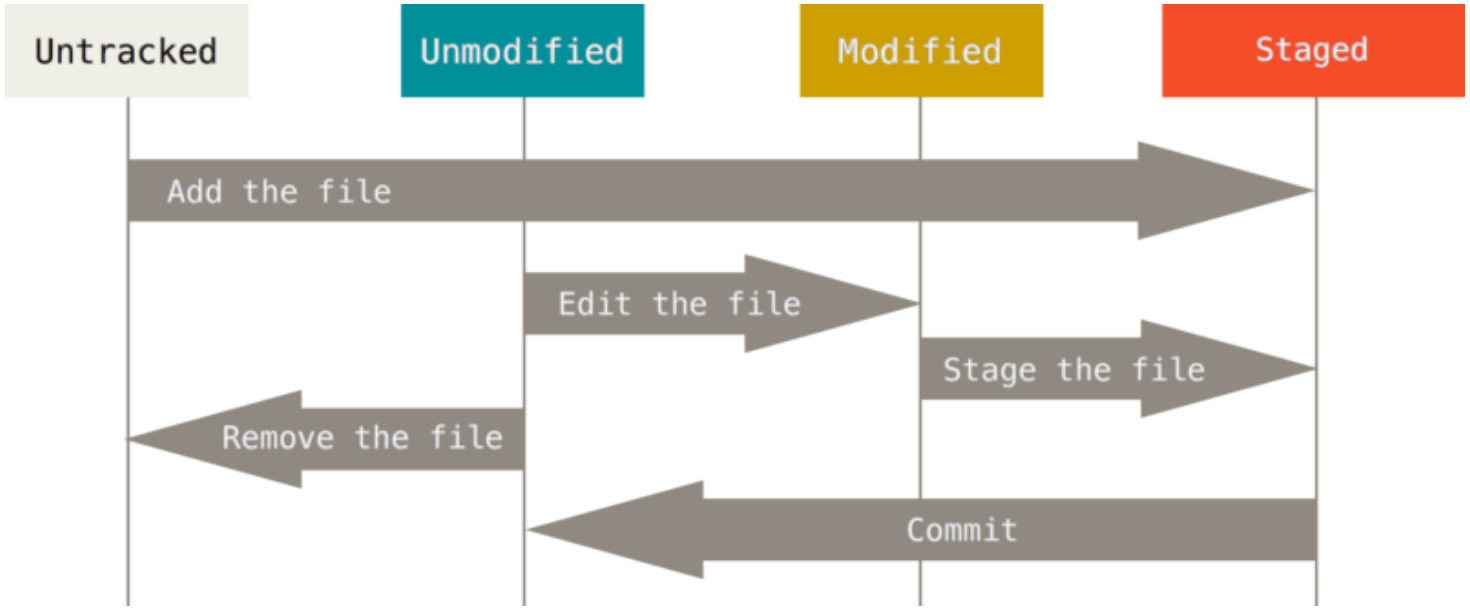
Recording Changes to the Repository

https://www.git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Files lifecycle

working copy.

- files can be:
  - **untracked**
  - **tracked**
    - unmodified
    - modified
    - staged



Checking the Status of Your Files

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

Untracked files

```
$ echo 'My Project' > README
$ 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)

    README

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

Adding a File to the Staging Area

```
$ git add README
```

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    new file:   README
```

If you commit README will be part of the commit and will be a tracked file from now on.

Modifying a file

Let's assume we change `CONTRIBUTING.md` which is already tracked.

```
$ 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)

    new file:   README

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:   CONTRIBUTING.md
```

In this case, since `CONTRIBUTING.md` is being tracked, `git` reports that it changed on disk, but it **does not** add it to the commit.

```
$ git add CONTRIBUTING.md
$ 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)

    new file:   README
    modified:   CONTRIBUTING.md
```

What happens if you change `CONTRIBUTING.md` again before committing?

```
$ vim CONTRIBUTING.md
$ 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)

    new file:   README
    modified:   CONTRIBUTING.md

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:   CONTRIBUTING.md
```

Ignoring Files

```
$ cat .gitignore
*.[oa]
*~
```

gitignore syntax

- Blank lines or lines starting with `#` are ignored.
- Standard glob patterns work, and will be applied recursively throughout the entire working tree.
- You can start patterns with a forward slash (`/`) to avoid recursivity.
- You can end patterns with a forward slash (`/`) to specify a directory.
- You can negate a pattern by starting it with an exclamation point (`!`).

Ignoring Files

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

Viewing Your Staged and Unstaged Changes

To see what you've changed but not yet staged, type `git diff` with no other arguments:

```
$ git diff
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 8ebb991..643e24f 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -65,7 +65,8 @@ branch directly, things can get messy.
 Please include a nice description of your changes when you submit your PR;
 if we have to read the whole diff to figure out why you're contributing
 in the first place, you're less likely to get feedback and have your change
-merged in.
+merged in. Also, split your changes into comprehensive chunks if your patch is
+longer than a dozen lines.

 If you are starting to work on a particular area, feel free to submit a PR
 that highlights your work in progress (and note in the PR title that it's
```

Viewing Your Staged and Unstaged Changes

If you want to see what you've staged that will go into your next commit, you can use `git diff --staged`. This command compares your staged changes to your last commit:

```
$ git diff --staged
diff --git a/README b/README
new file mode 100644
index 0000000..03902a1
--- /dev/null
+++ b/README
@@ -0,0 +1 @@
+My Project
```

Committing Your Changes

```
$ git commit
```

The editor displays the following text (this example is a Vim screen):

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#       new file:   README
#       modified:   CONTRIBUTING.md
#
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

Committing Your Changes

If you want to skip the staging area and proceed directly to the commit:

```
$ git commit -a -m 'Add new benchmarks'
```

Viewing the Commit History

https://www.git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

Viewing the Commit History

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

Viewing the Commit History

```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

 lib/simplegit.rb | 5 -----
 1 file changed, 5 deletions(-)

...
```

Viewing the Commit History

```
$ git log --pretty=oneline
ca82a6dff817ec66f44342007202690a93763949 Change version number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 Remove unnecessary test
a11bef06a3f659402fe7563abf99ad00de2209e6 Initial commit
```

Viewing the Commit History

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 Ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Add method for getting the current branch
* | 30e367c Timeout code and tests
* | 5a09431 Add timeout protection to grit
* | e1193f8 Support for heads with slashes in them
|/
* d6016bc Require time for xmlschema
*  11d191e Merge branch 'defunkt' into local
```

Working with Remotes

https://www.git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes

Working with Remotes

```
$ git clone https://github.com/schacon/ticgit
Cloning into 'ticgit'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
$ cd ticgit
$ git remote
origin
```

```
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
```

Push

`git push` is the command used to push your local changes to the remote server.

```
$ git push
```

This will succeed only if the server did not receive any other change since your last update. In case this is the case, just pull (see next topic) the changes and retry.

Pull

One of the most used commands in `git` is

```
$ git pull
```

which updates the current branch with updates from the remote server.

Pull

`pull` is the concatenation of two commands:

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

Which fetches updates from the given remote server.

```
$ git merge <remote>/<branchname>
```

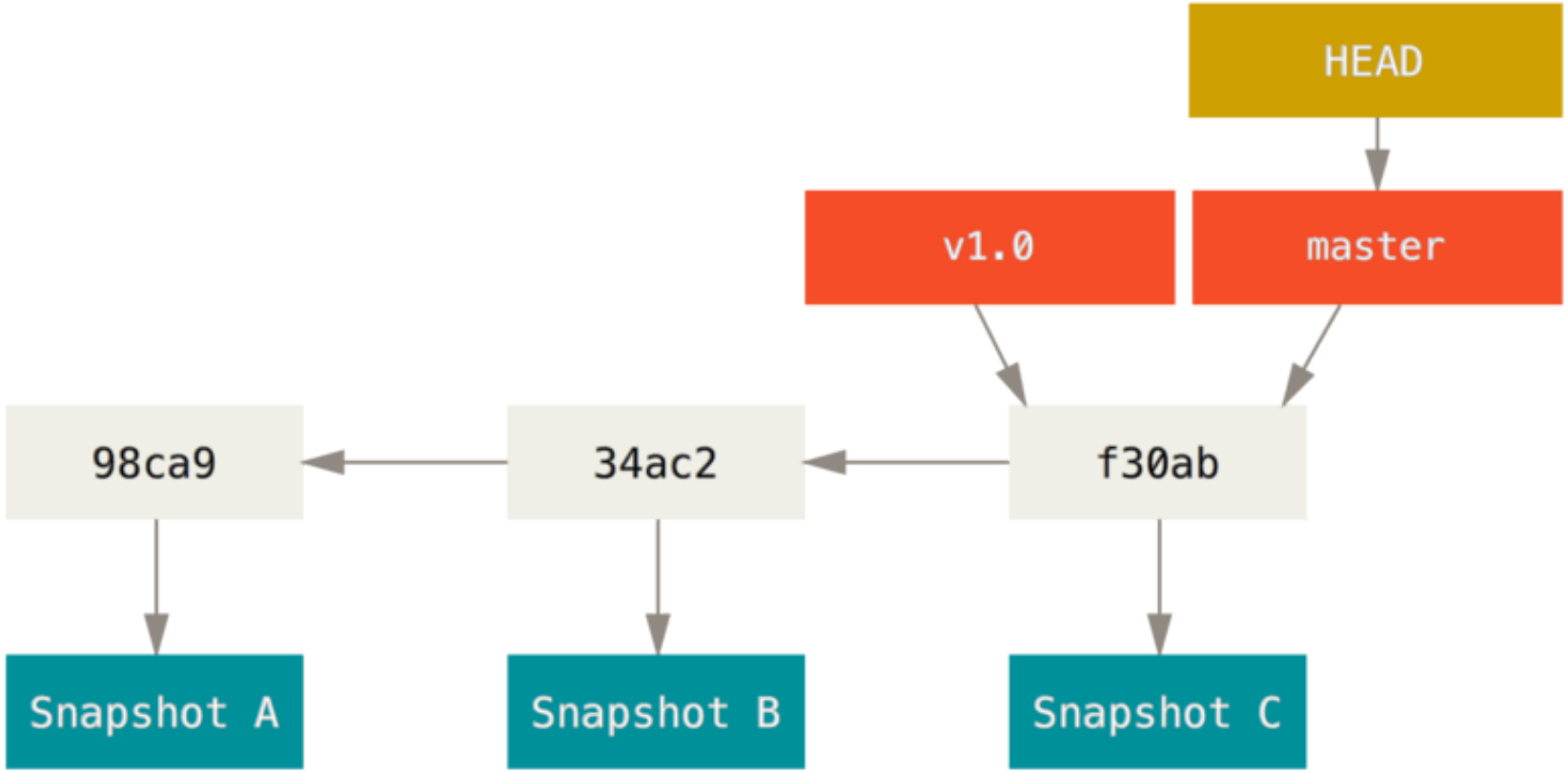
Keeping this in mind is useful, because as you are merging changes from remote, you need to be prepared to work with the merge process (e.g., resolving conflicts -- which will be introduces shortly).

Branches in a Nutshell

https://www.git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

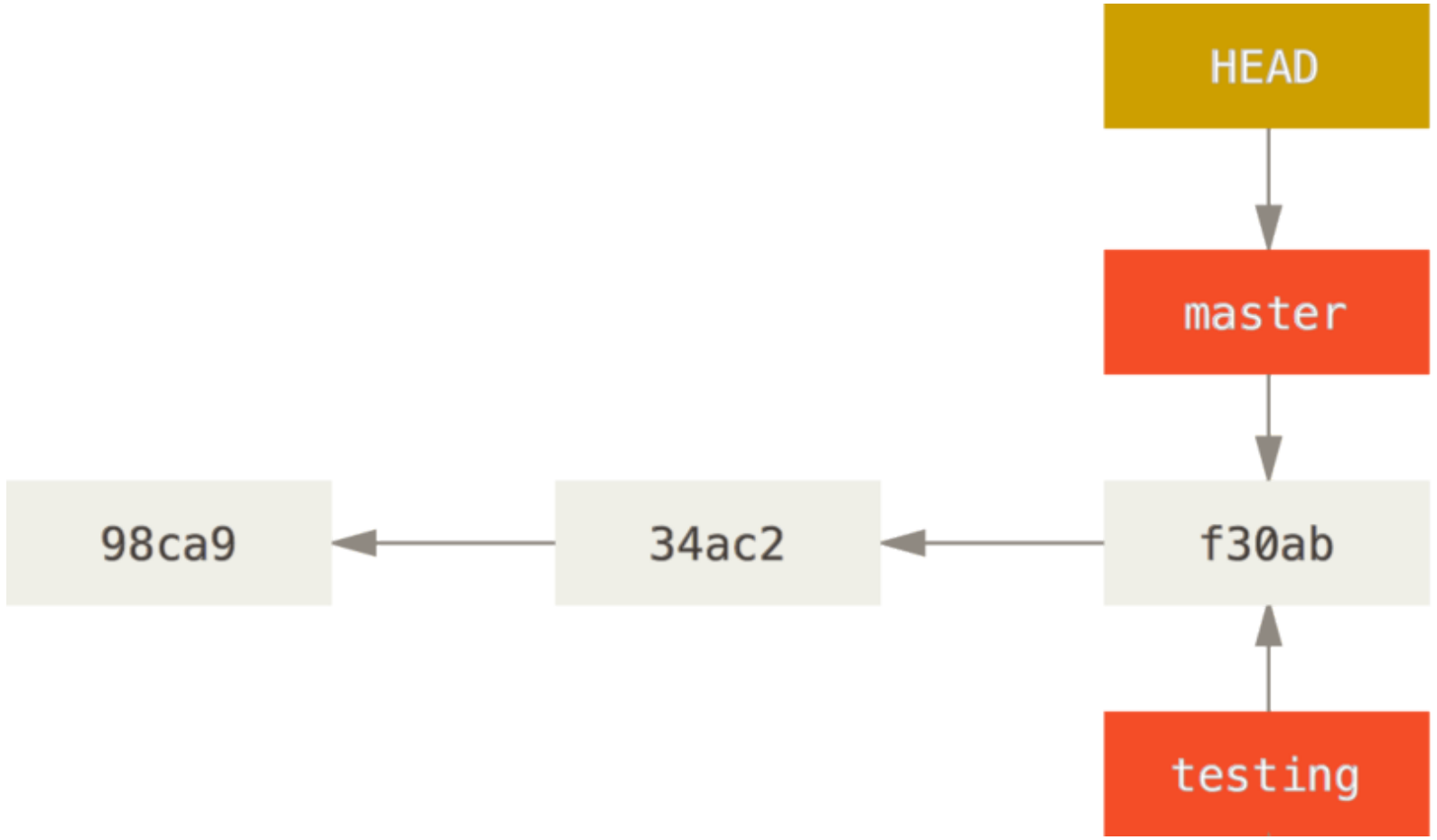
> Git encourages workflows that branch and merge often.

Branches in a Nutshell



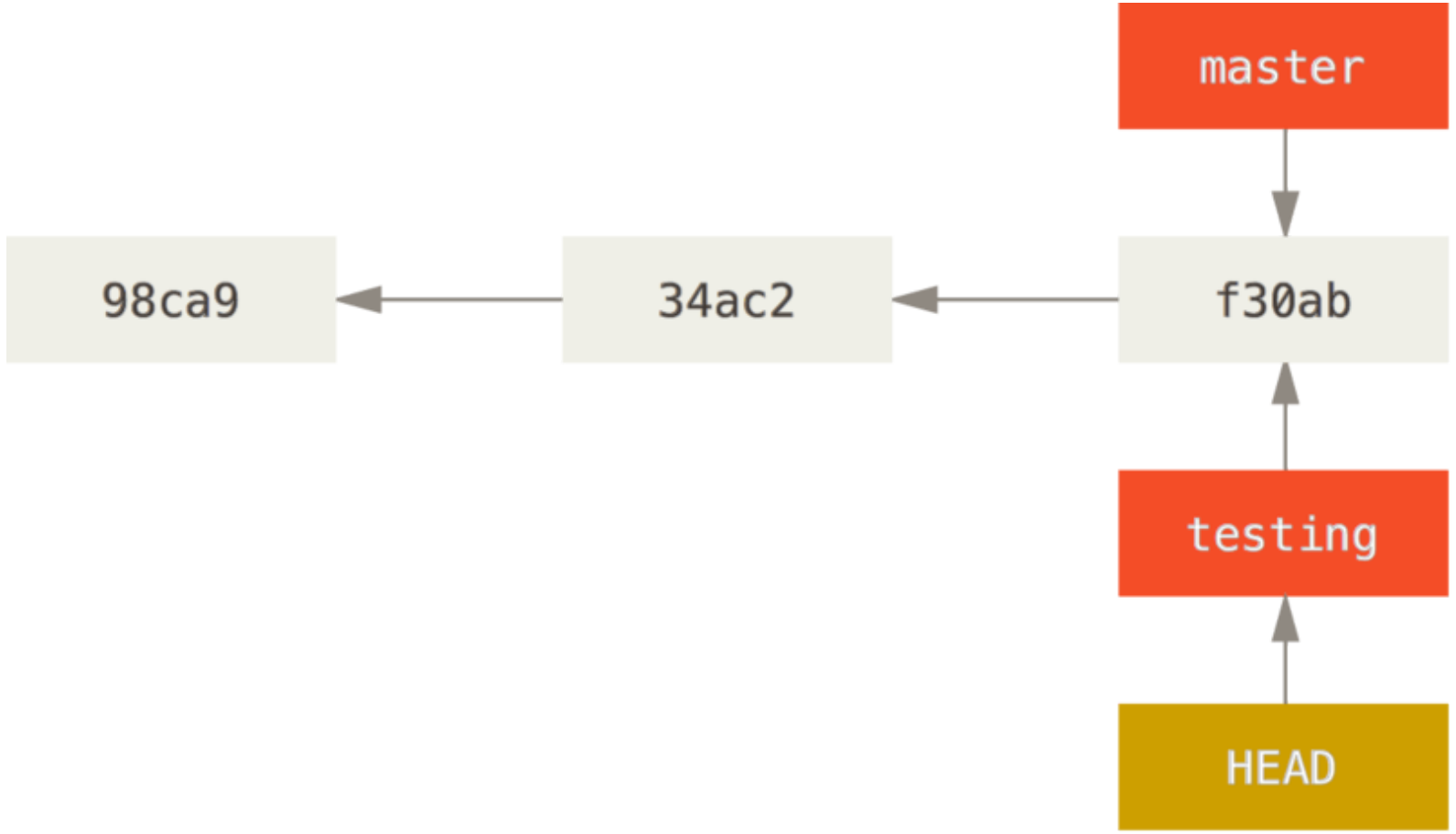
Branches in a Nutshell

```
$ git branch testing
```



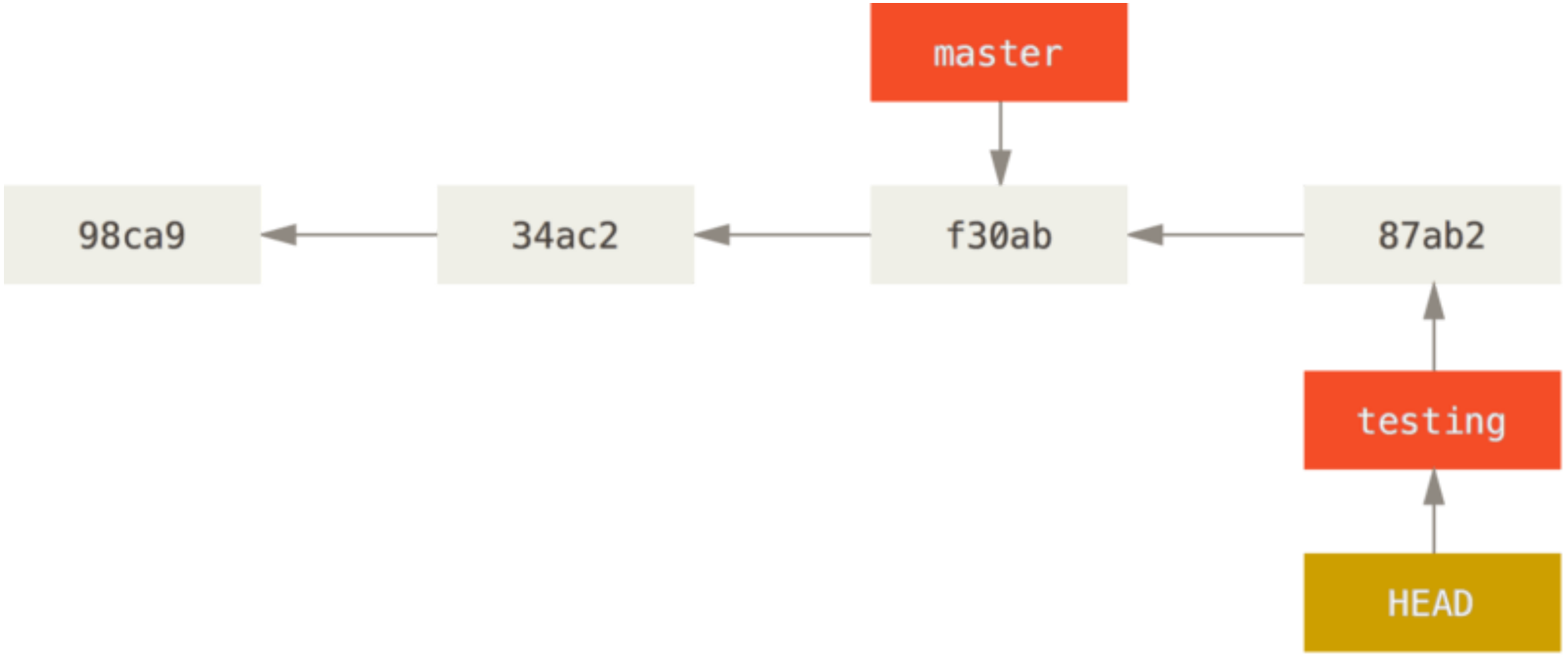
Branches in a Nutshell

```
$ git checkout testing
```



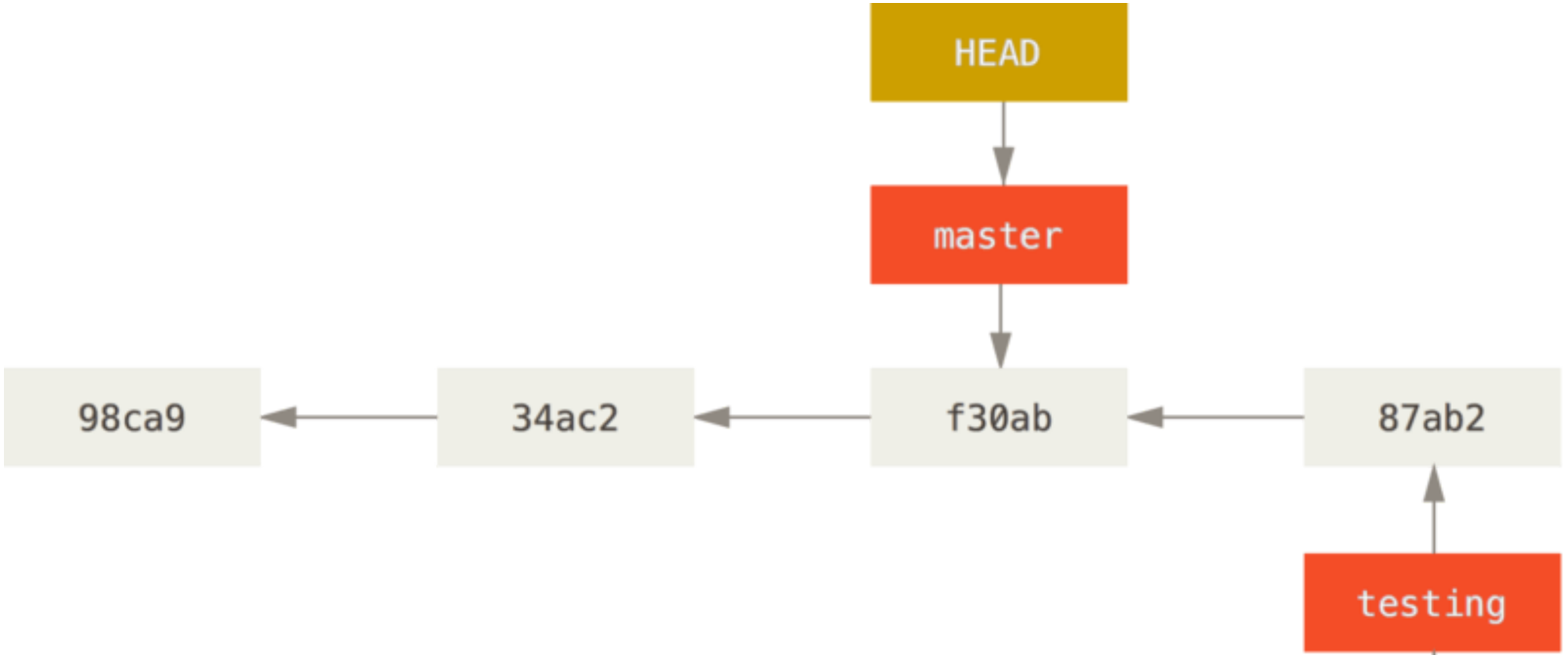
Branches in a Nutshell

```
$ vim test.rb
$ git commit -a \
  -m 'made a change'
```



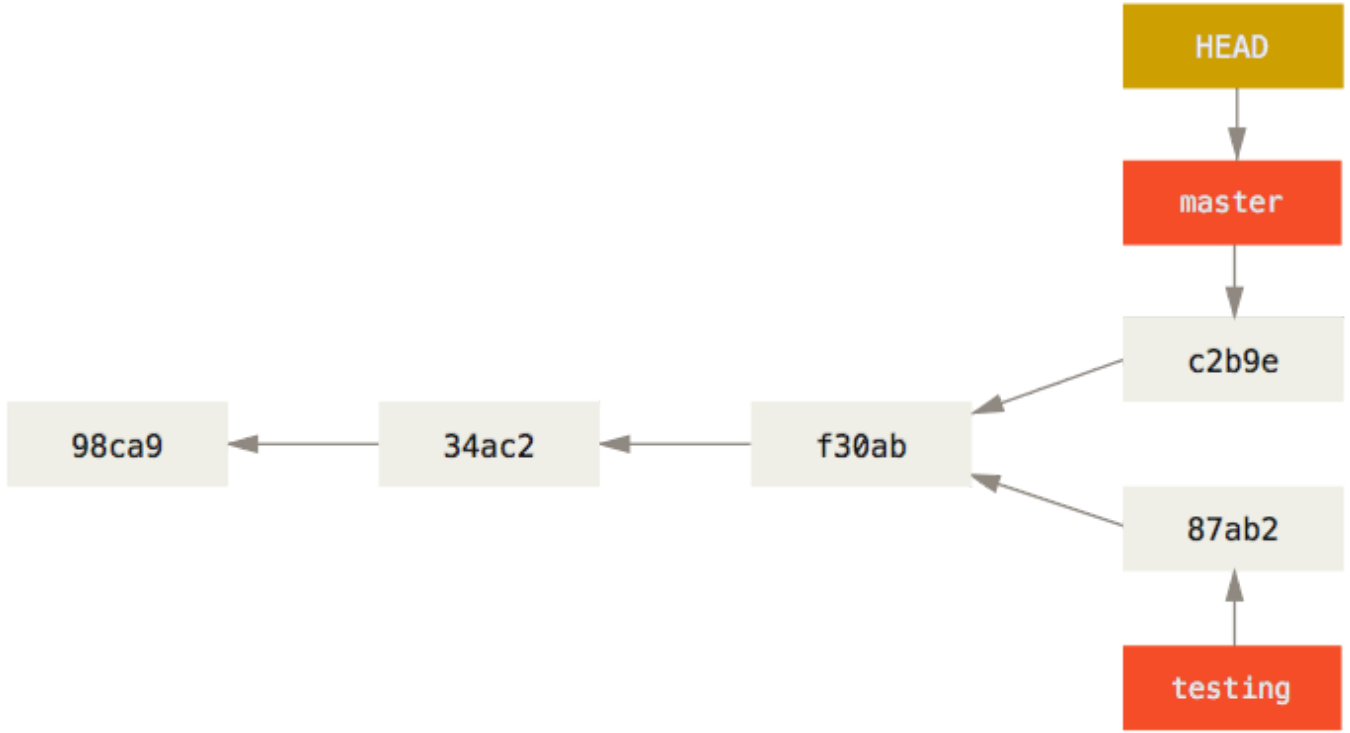
Branches in a Nutshell

```
$ git checkout master
```



Branches in a Nutshell

```
$ vim test.rb
$ git commit -a \
  -m 'other changes'
```

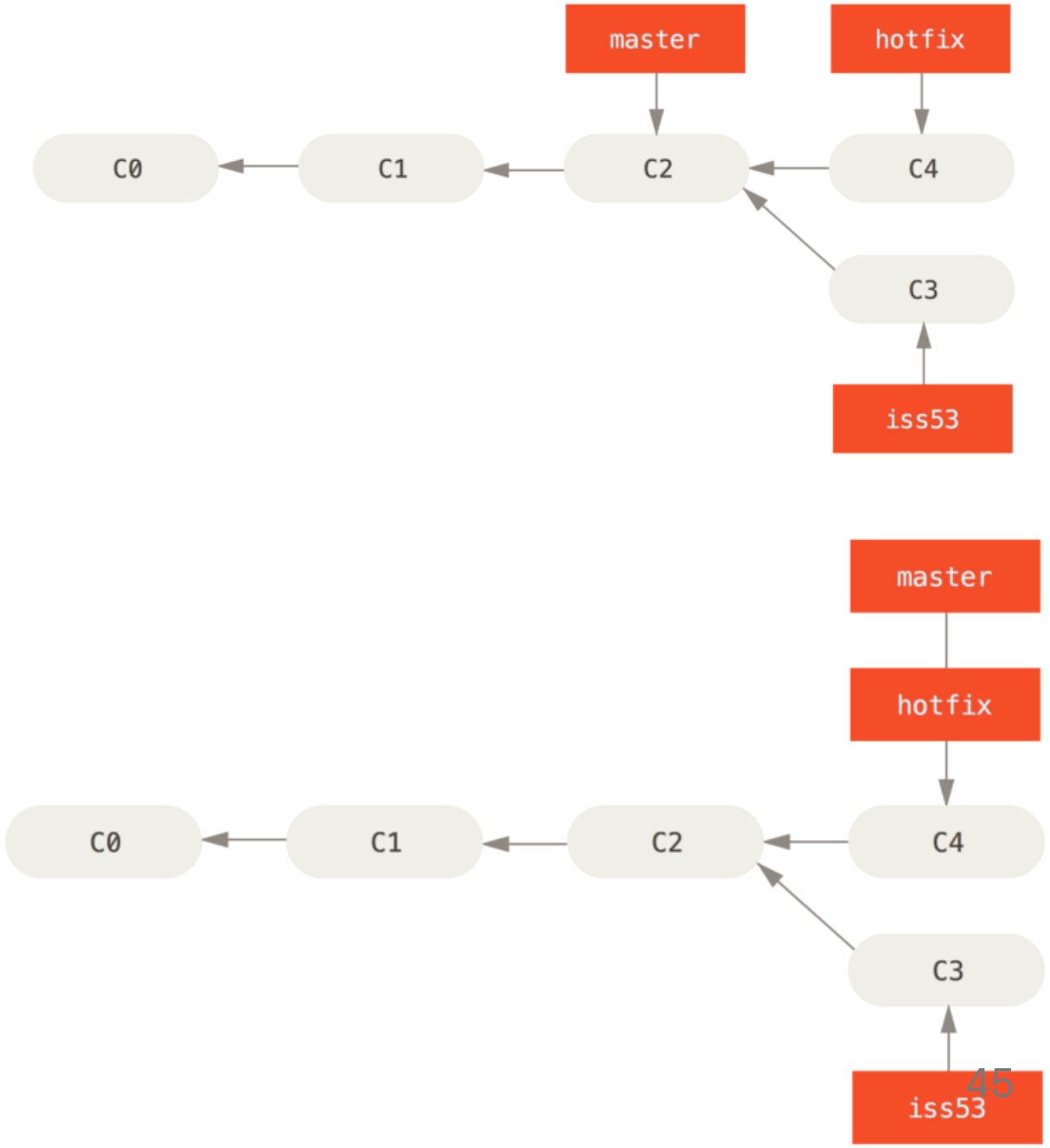


Basic Branching and Merging

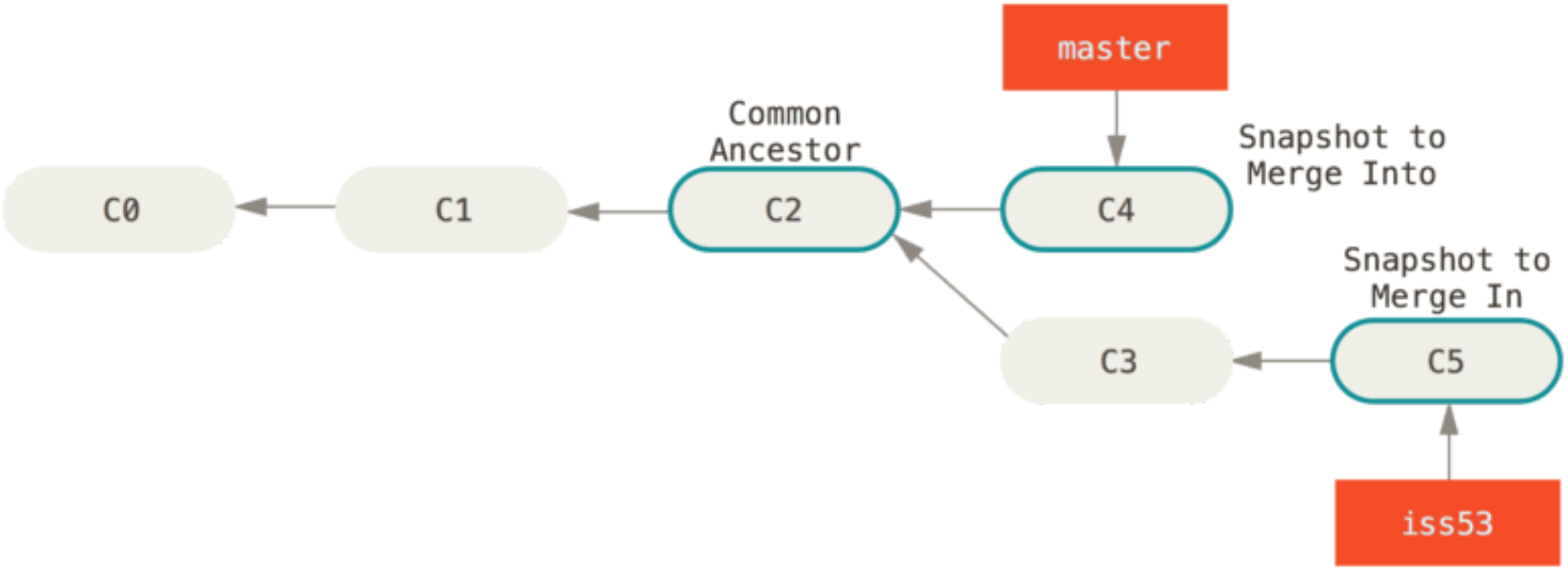
https://www.git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Fast-forward

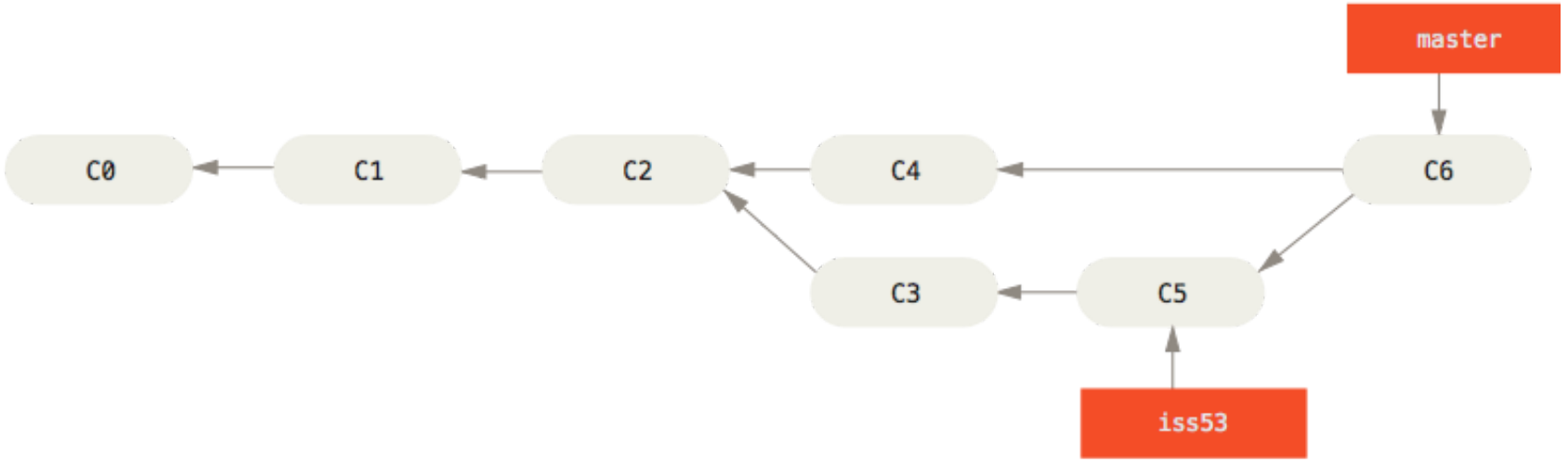
```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```



Basic Merging



Basic Merging



Conflicts

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Conflicts

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

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

Conflicts

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Mergetool

```
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge
gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3
codecompare vimdiff emerge

Merging:
index.html

Normal merge conflict for 'index.html':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (opendiff):
```

Committing the result

```
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    modified:   index.html

Merge branch 'iss53'

Conflicts:
    index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   index.html
#
```

GitLab

https://www.git-scm.com/book/en/v2/Git-on-the-Server-GitLab

Generating Your SSH Public Key

https://www.git-scm.com/book/en/v2/Git-on-the-Server-Generating-Your-SSH-Public-Key

Generating Your SSH Public Key

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/schacon/.ssh/id_rsa):
Created directory '/home/schacon/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/schacon/.ssh/id_rsa.
Your public key has been saved in /home/schacon/.ssh/id_rsa.pub.
The key fingerprint is:
d0:82:24:8e:d7:f1:bb:9b:33:53:96:93:49:da:9b:e3 schacon@mylaptop.local
```

Your public key (i.e. the content of `~/.ssh/id_rsa.pub`) should be copied to GitLab.

Setting up GitHub to use your ssh key

From any page on GitLab after login select your profile -> SSH keys and add the content of the id_rsa.pub to the large text box. Fill in the "title" (usually the name of the computer where you generated the key) and push `add key`.

Add members

You can add as many collaborators as you want to your repository (on the GitLab page for the repository just select members on the left sidebar and invite other GitLab users from there).