Images are stolen grom the git scm book. read the full story there!

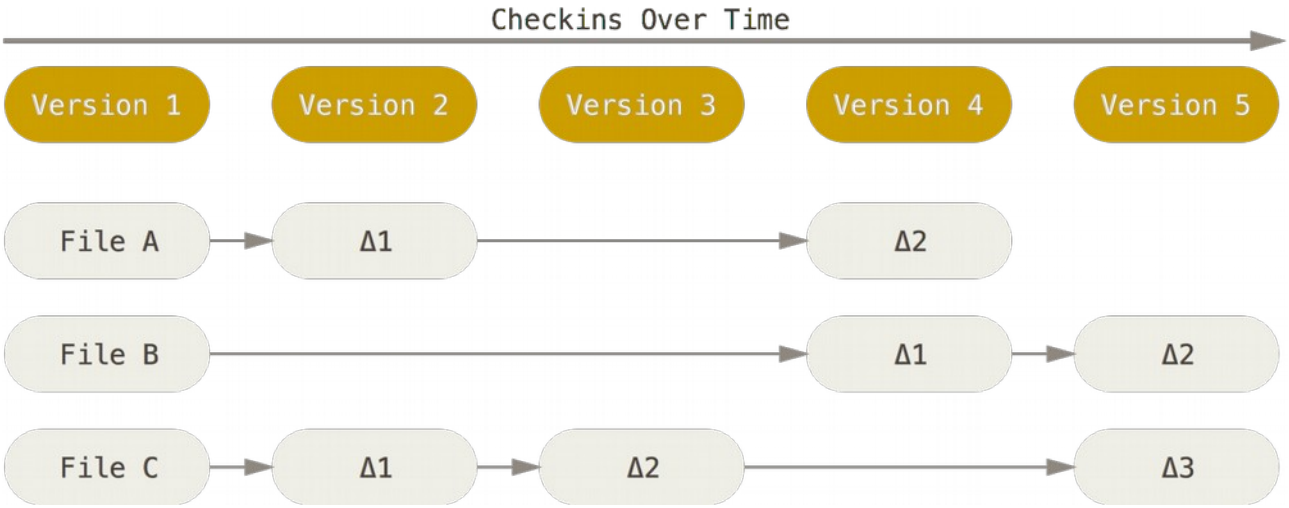
Concepts / Theory

txt diffs

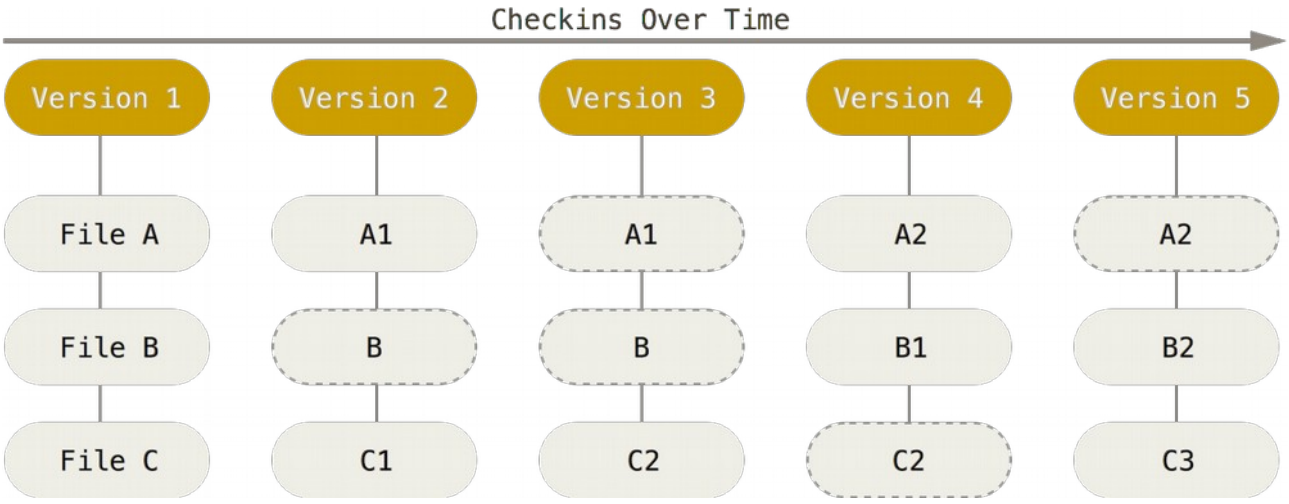
working (ONLY!) on txt - based files
diff on txt files
→ format intelligently
→ stay with it (no reformating)
→ → agree on "code style"

BUT:
Git doesn't store deltas, git stores entire shapshots

not like this:

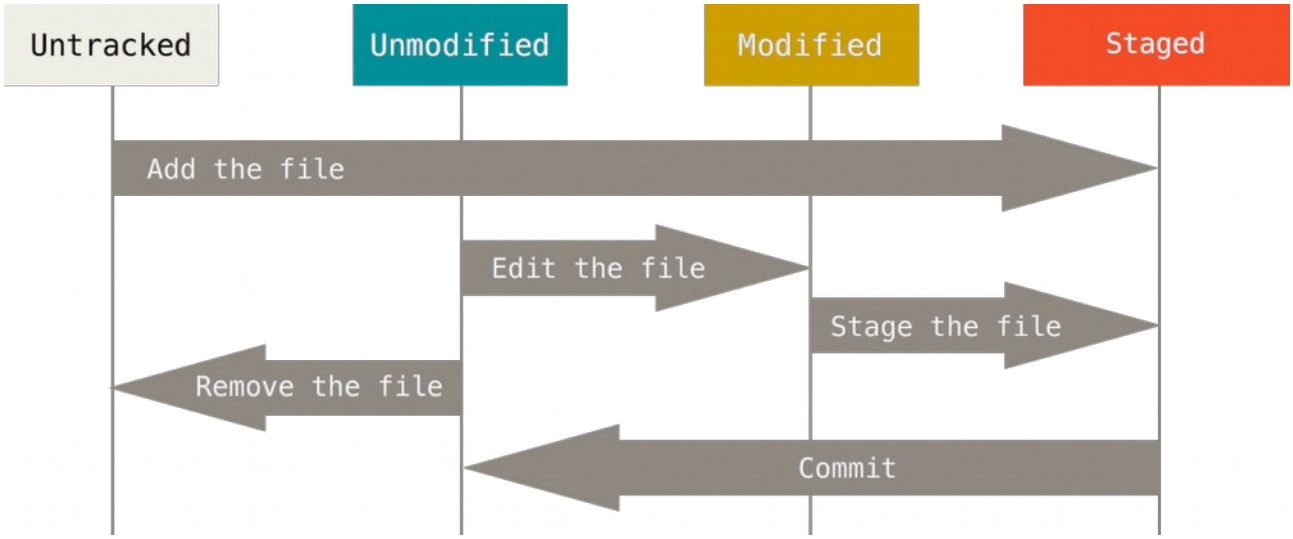


but like this:



→ avoid binary files / large txt files that change often (images, datafiles) !!!! (use `.gitignore` for them)

Status of a file:

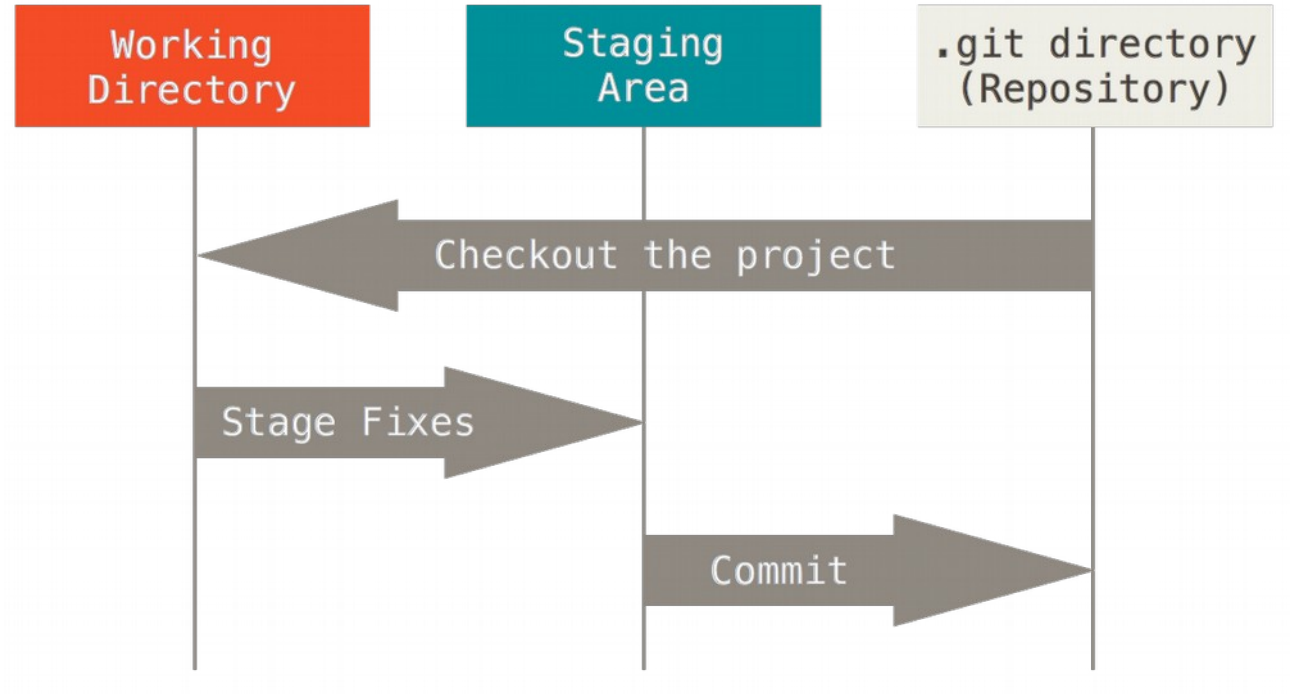


check the overall status with

```
git status
```

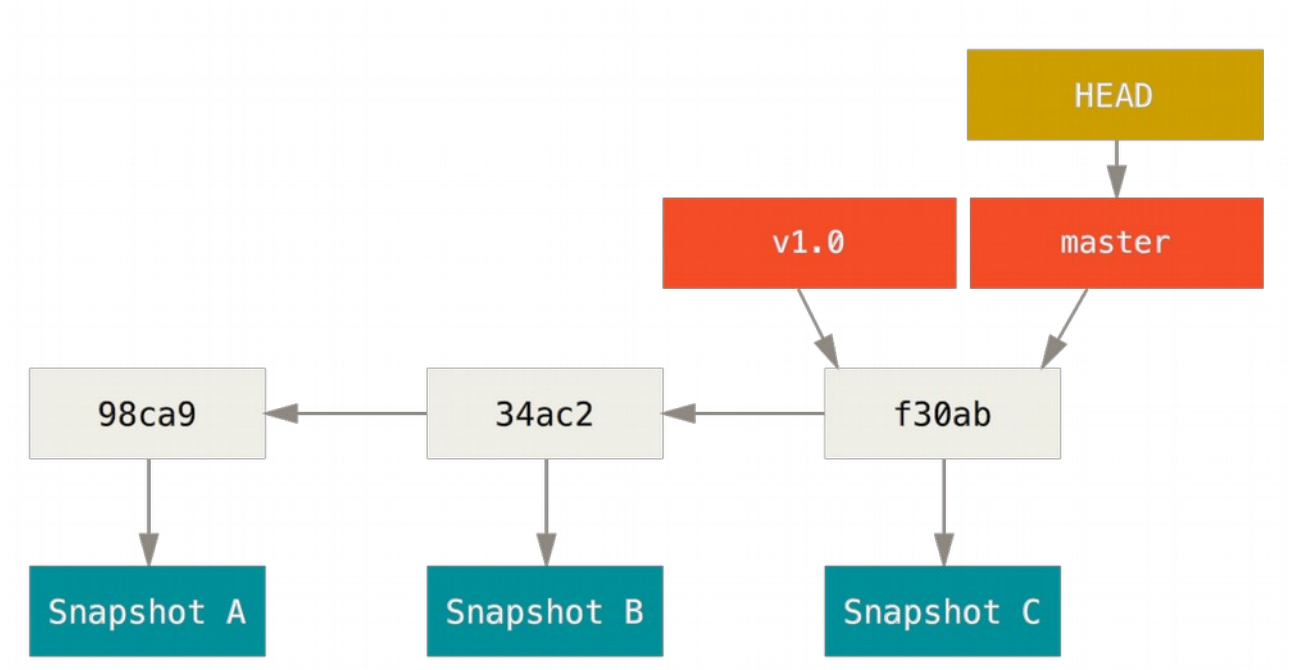
Additionally a file can be ignored completely, if its listed on `.gitignore`. Details later.

The three states (+stash)



Bare vs NonBare repro

History of States / Versions / Commits:



This is a branch → branch named "master" (is default name)
tags mark specific points in the branch "v1.0"
branch name always points to the last commit in a line
HEAD always points to the top of the current branch, the one you checked out and are working on.
→ there can be more than one branch!
→ and thats a good thing!!! (do it often)

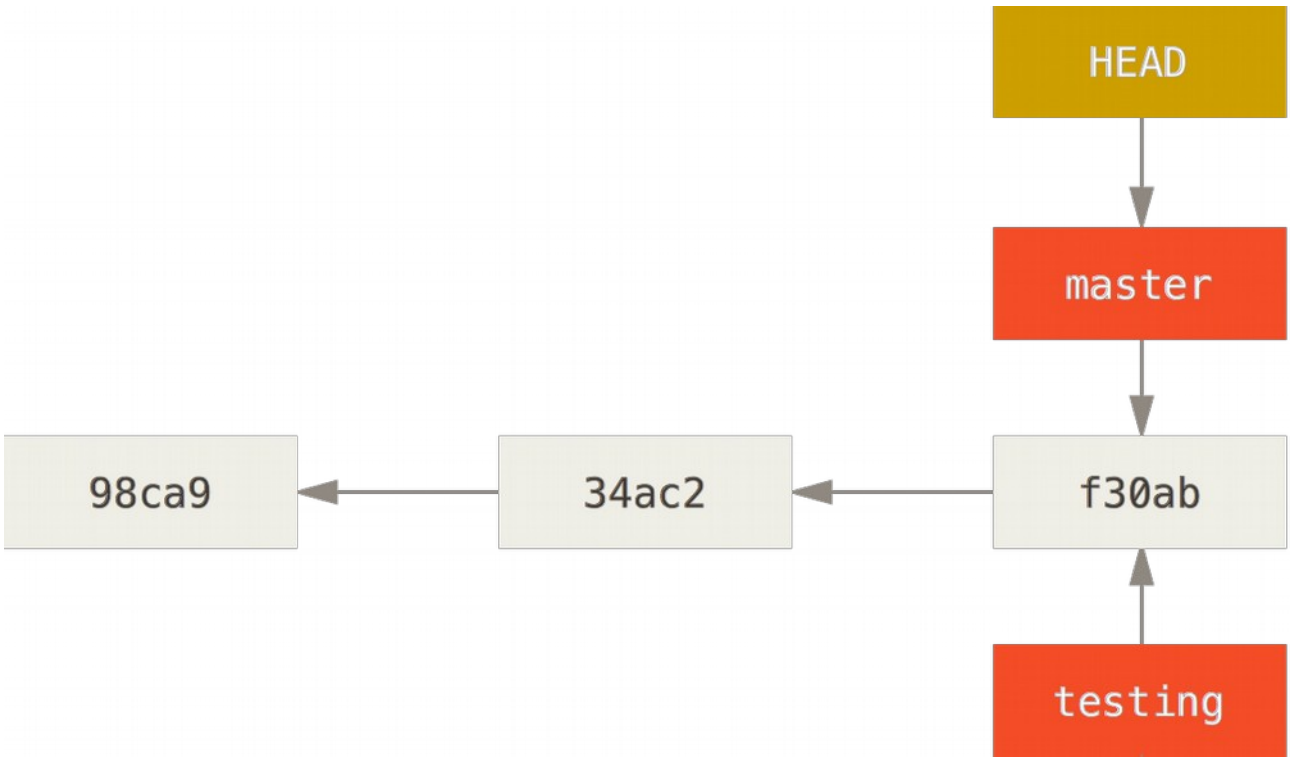
Using branches

Create branch

use this to create a new branch named "testing"

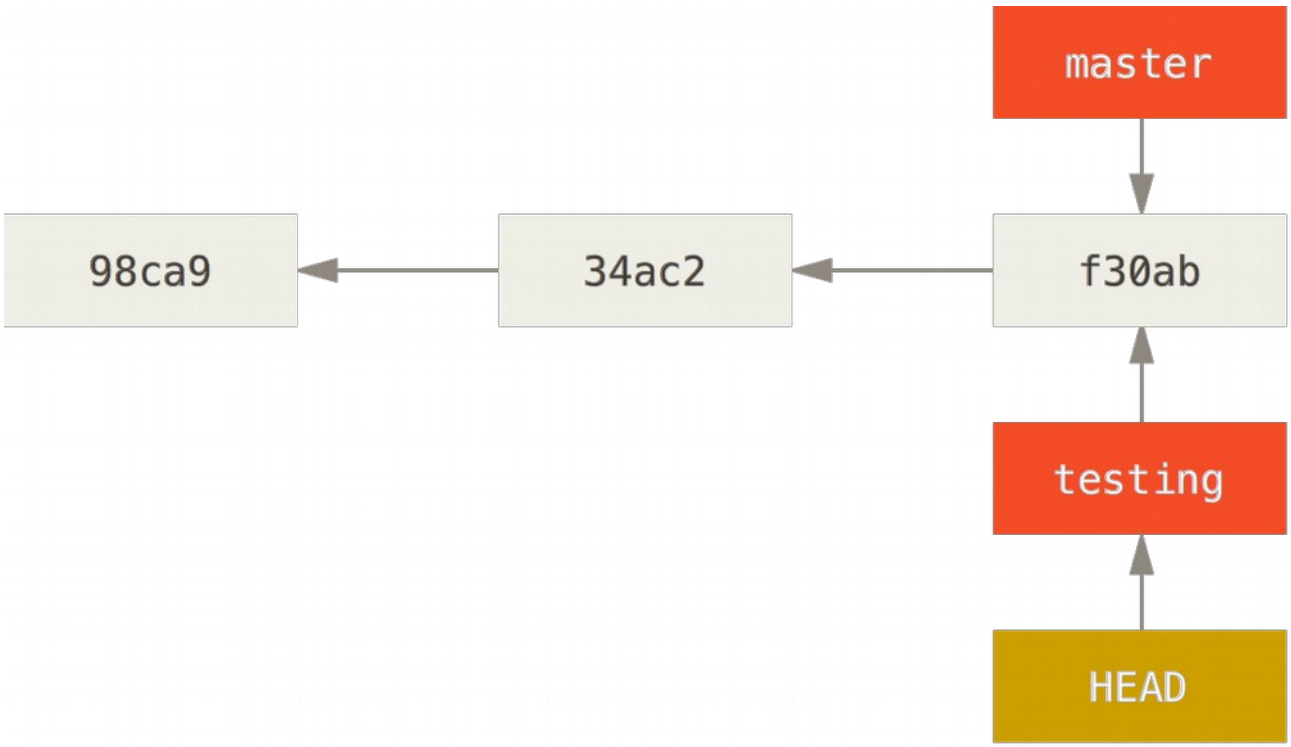
```
git branch testing
```

thats what you usually do if you would create a copy / folder "my_project_v2" in pre git times...

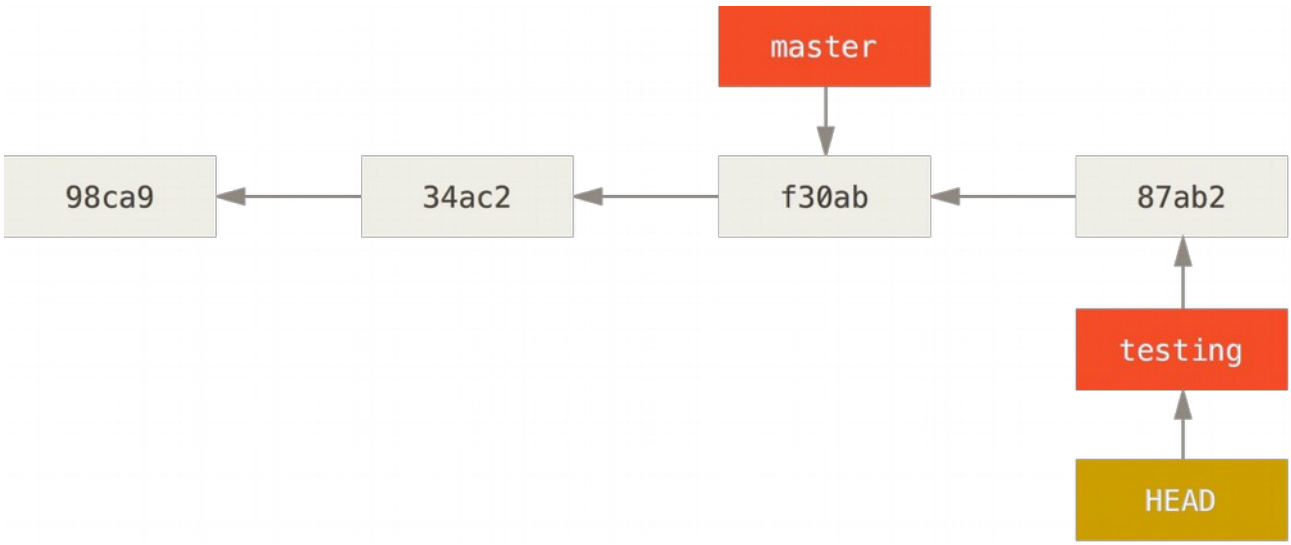


Switch branch

```
git checkout testing
```



edit a file, make a new commit:



you can switch branches at any point:

```
git checkout master
```

Note that ALL your files in the working directory get changed!!

See branches

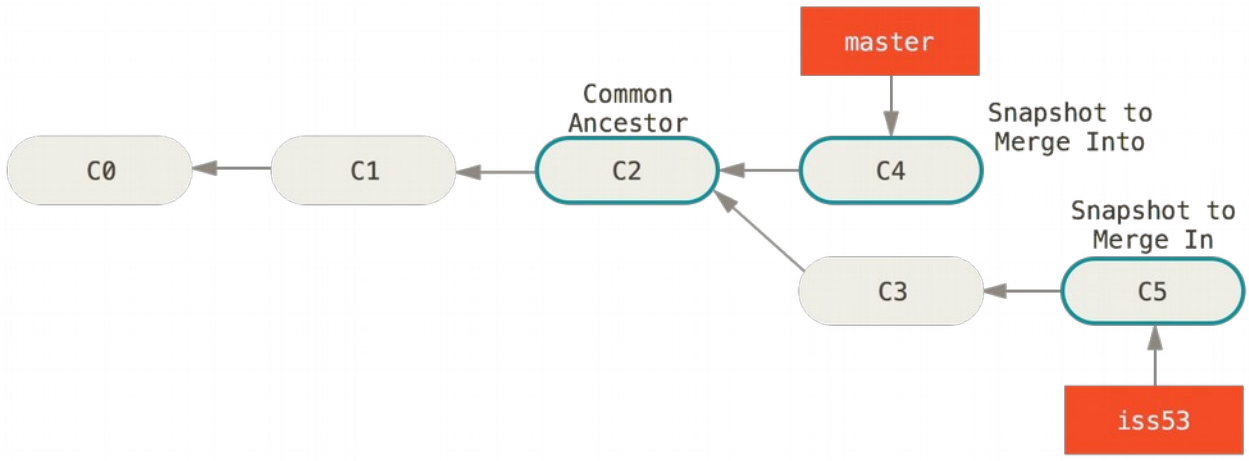
Check the log / branching history of the project in command line / gui

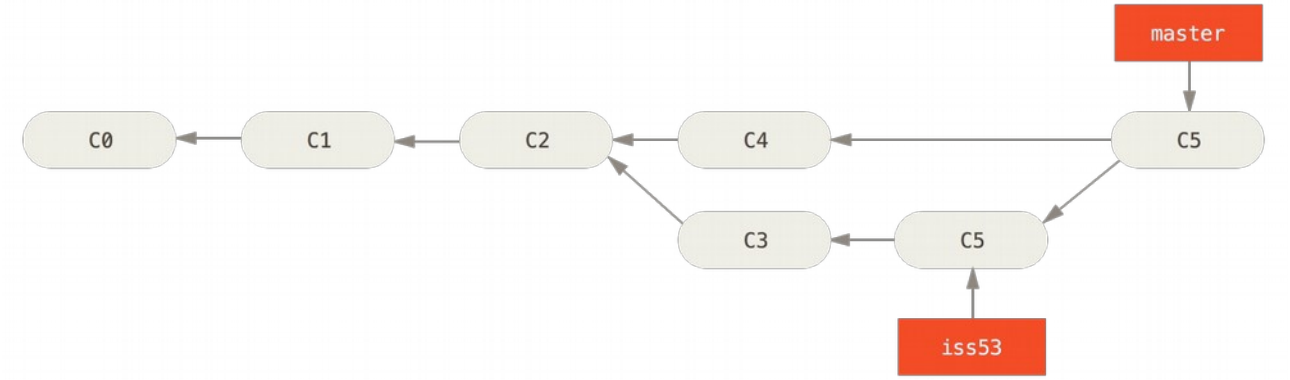
```
git log          # cli
gitk --all       # gui
```

Merging branches

You merge a branch INTO the current one:

```
git checkout master    # change the current
git merge iss53        # merge iss53 into current=master
```





Merge conflict

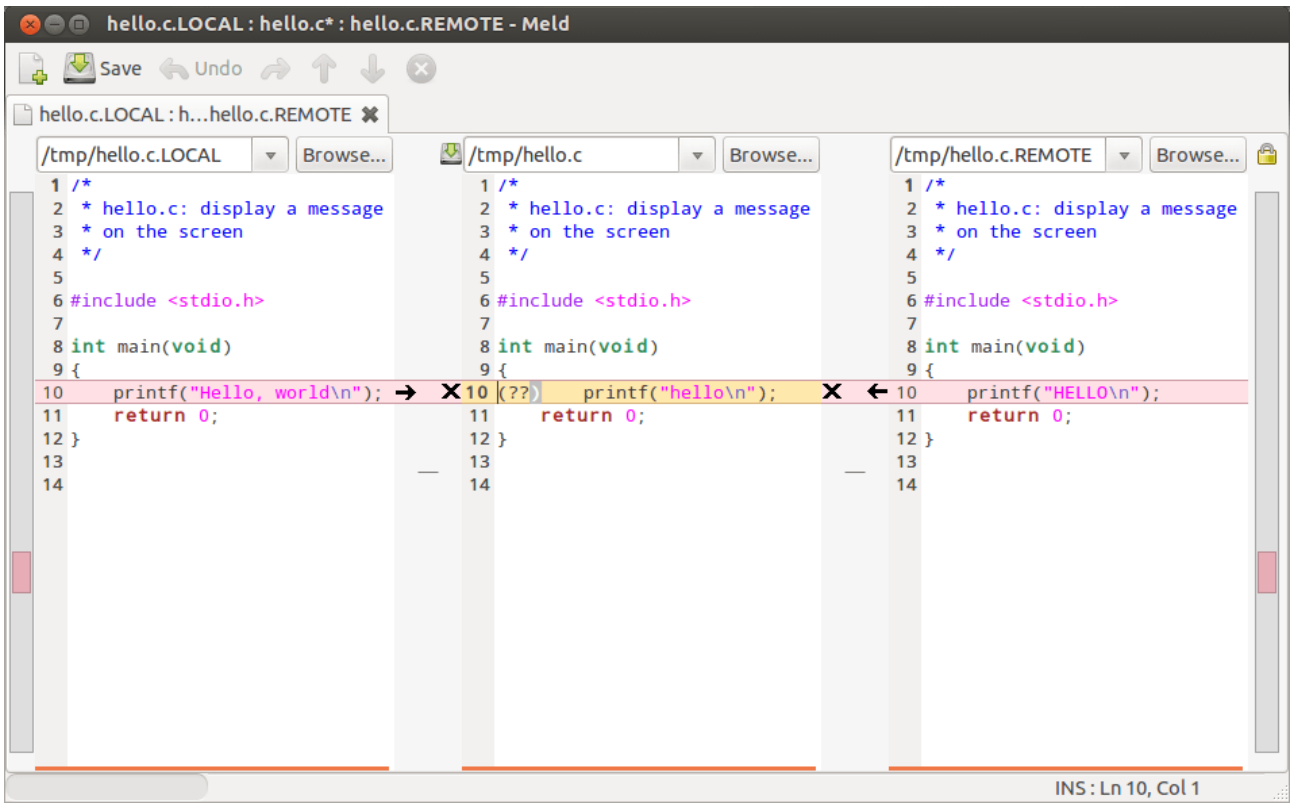
In case of conflicts (both C4 and C5 changed the same file) → merge conflict
start the merge.
Edit the file manually or use a gui mergetool like kdiff3, meld by typing:

```
git mergetool
```

A file with conflict looks like:

```
[...]
<<<<<<< HEAD:index.html
I, user1 wrote this line
=======
I, user2 wrote another line
>>>>>>> iss53:index.html
[...]
```

Choose one version, delete the lines with `<<<<` `====` `>>>>`



Do git status, fix files iteratively until everything is fixed. then commit.

Remove branch

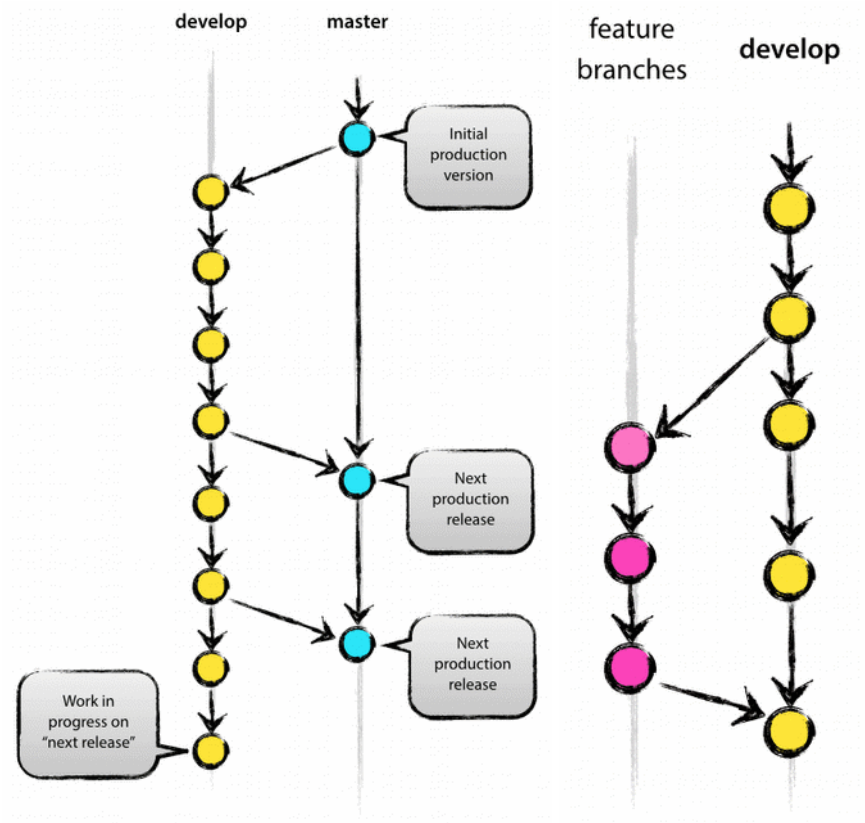
At the end, or if an experiment didn't go well, you can delete old / merged or unused branches:

```
git branch -d iss53
```

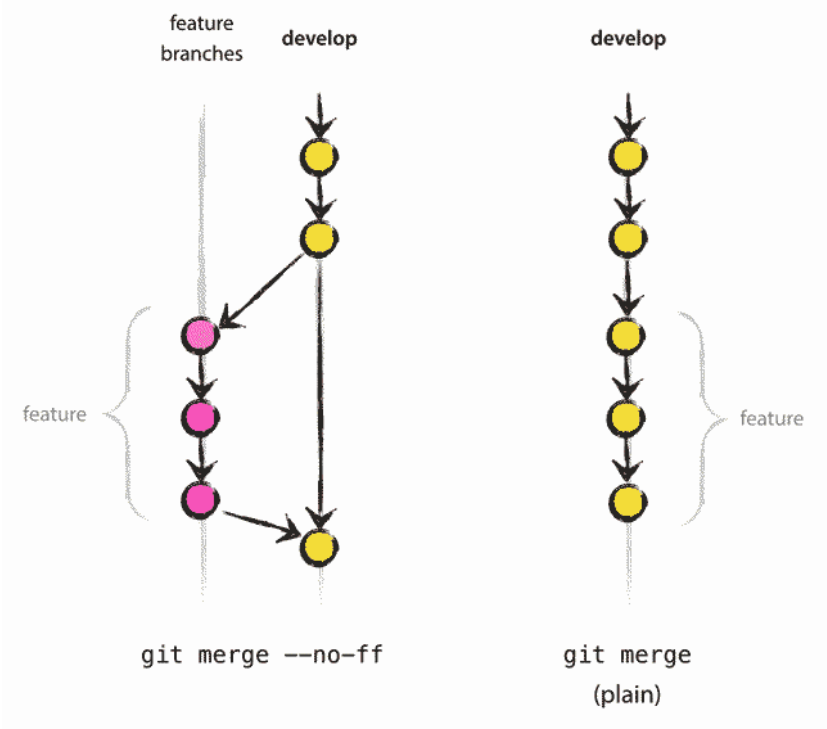
Using branches

from here: http://nvie.com/posts/a-successful-git-branching-model/
Simple model:
Use "master" branch with dynamically created feature branches to try out stuff. if it works, merge the branch. if not, delete it...

More complicated setup: master, develop, feature branches



Using fast forward `--ff` vs not using it – `no-ff` while merging



Undoing / correcting things

Only do changes to the history if you didn't share (push) anything yet.
Best to NEVER rewrite the history...

Quick fix if you forgot to add one file for a commit or want to change the commit message:

```
git commit -m 'initial commit'
git add forgotten_file
git commit --amend -m "new commit message"
```

Revert a file:

```
git checkout -- CONTRIBUTING.md
```

and ALL your local changes are gone!

Using remotes

Branches can be in your local repo, but also on someones else computer:
Starting with a copy of a remote repository:

```
git clone https://github.com/<username>/<project>
    <local_folder>
```

Git clone makes a local copy, and create a remote named `origin`.

Or add a remote repro after you created your own:

```
git remote
git remote -v

git remote add [remote_name]
    https://github.com/<username>/<project>
```

Get the updates from a remote, don't do anything

```
git fetch [remote_name]
```

Get the updates from a remote, and automatically merge the branch:

```
git pull [remote_name] [branch_name]
```

Publish your changes to a remote

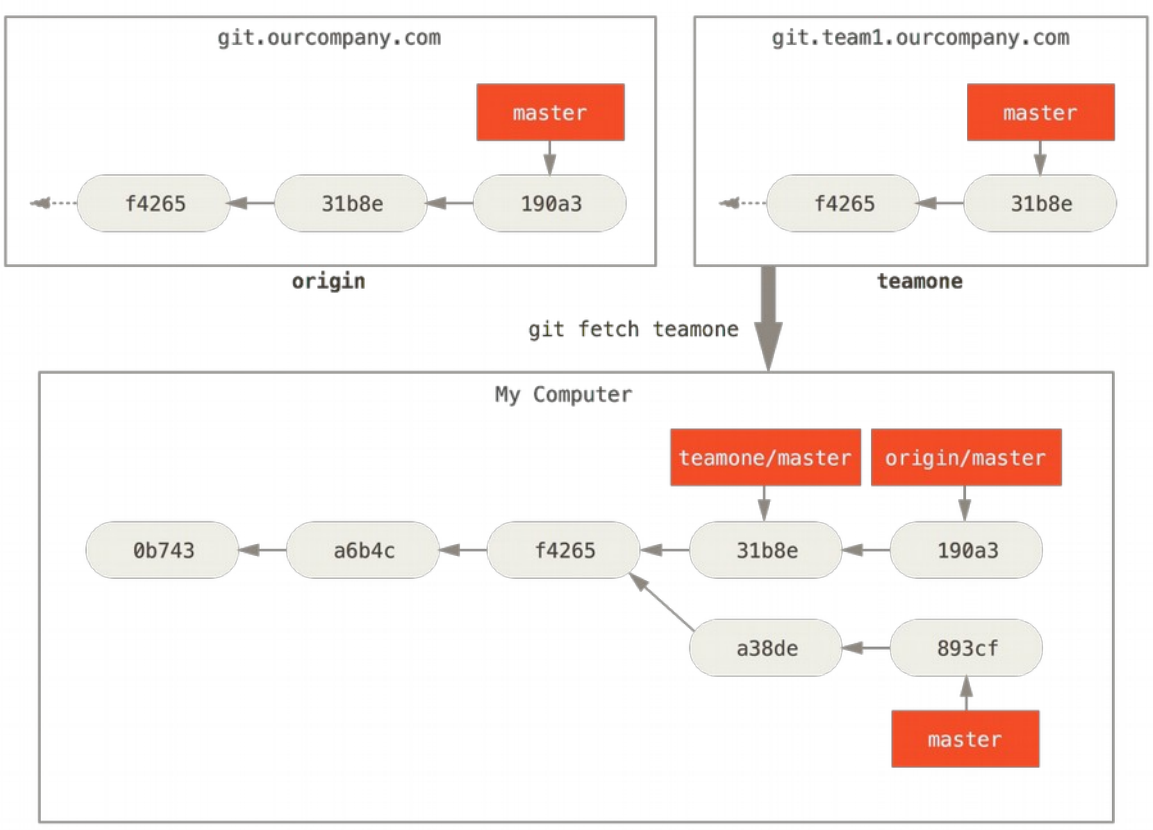
```
git push [remote-name] [branch-name]
```

Inspect a remote:

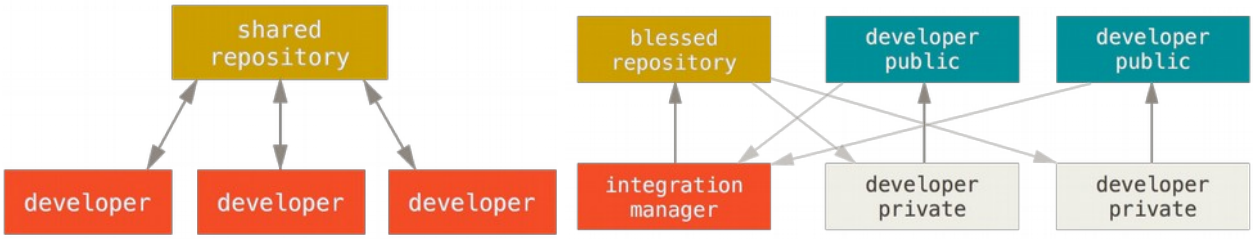
```
git remote show [remote]
```

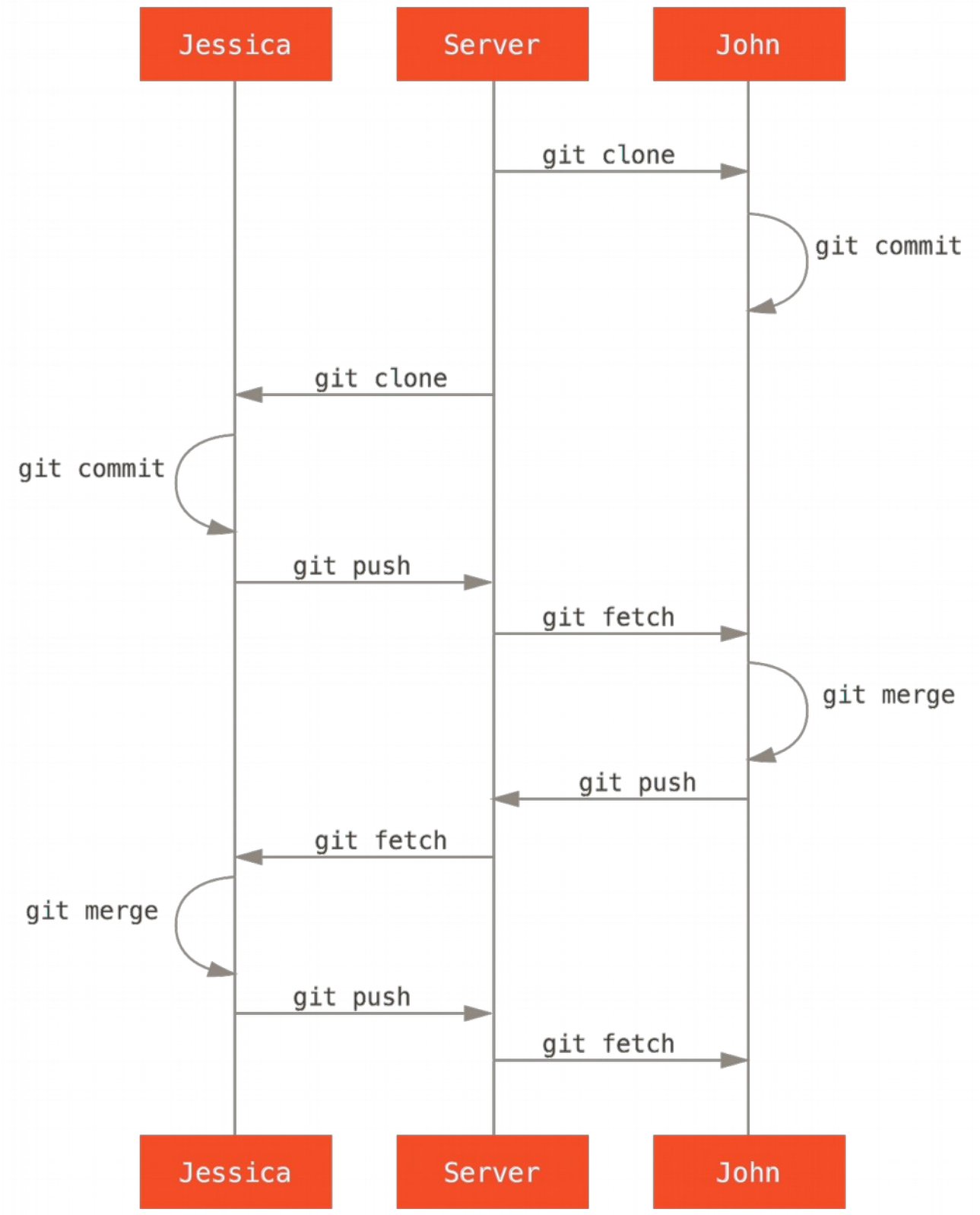
Track a remote branch:

```
git checkout --track [remote]/[branch]
```



Distributed workflows





GitHub Setup

Central repro vs Forks / Pull request vs remotes

General Hint / get help

RTFM!

```
https://git-scm.com/book/en/v2
```

and for allem use:

```
git status
```

and read what's written on the screen!!!

get help!

```
git help <command>
git <command> --help
```

First steps

Make config

```
git config --global user.name "<name>"
git config --global user.email <email>
```

change the default text editor (for example use `kate` for a simple graphical)
and disable fast forwarding:

```
git config --system core.editor <editor>
git config --global --add merge.ff false
```

The configs are saved in

```
<my_project>/.git/config  # project specific config
~/.gitconfig              # User-specific settings
                                 (--global flag)
$(prefix)/etc/gitconfig   # System-wide settings
     (hardly ever used)
```

Set mergetool (use `kdiff3` or `meld`):

```
git config --global merge.tool kdiff3
```

start from scratch

local first

start a new project on your computer / local:

```
cd ~/my_project
git init
```

Create files. By default you should always have `README.md`, `LICENSE`, `.gitignore`

```
echo "#Mein Project" > README.md
echo "This work is under WTFPL" > LICENSE
nano .gitignore
```

In the `.gitignore` file, you can list files, that should be ignored by the version tracking. You

should list there all files that are artificially generated out of other files. (For example compiled programs, pdf and intermediate files out of latex documents… Google for example for `latex gitignore` to get a template.)

Add the new files to be versoin tracked:

```
git add .
```

Make your first commit:

```
git commit -am "Init of project"
```

then connect to github:

```
[create repro in github]
git
```

Github first

same as "Start from someone else"

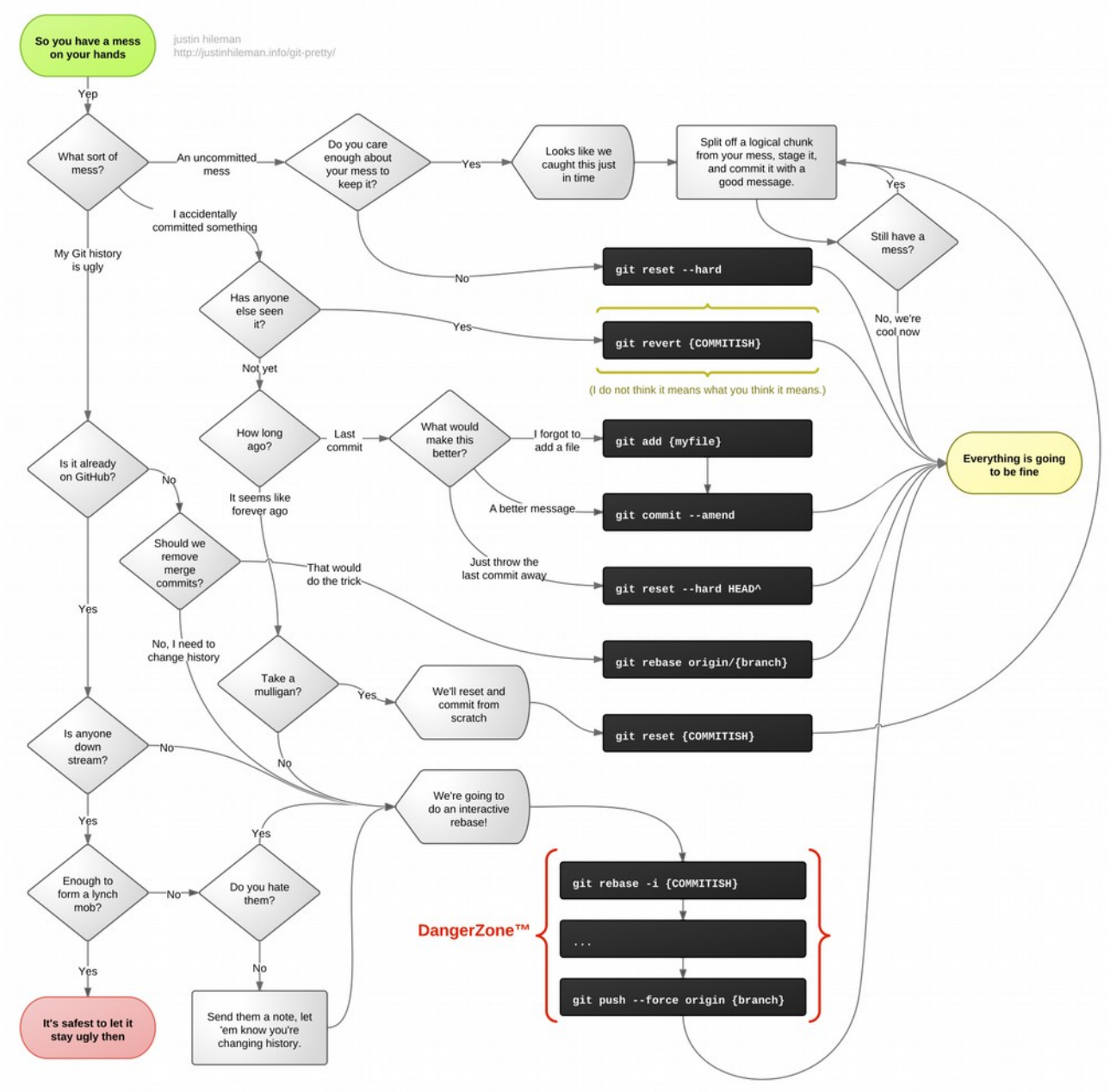
Start from someone else

```
cd <projects>
git clone https://github.com/<username>/<project> <local_name>
```

Cheat sheet:

http://ndpsoftware.com/git-cheatsheet/previous/git-cheatsheet.html

First Aid


So you have a mess on your hands
justin hileman
http://justinhileman.info/git-pretty/
Yep
What sort of mess?
An uncommitted mess
I accidentally committed something
My Git history is ugly
Do you care enough about your mess to keep it?
Yes
No
Looks like we caught this just in time
Split off a logical chunk from your mess, stage it, and commit it with a good message.
Still have a mess?
Yes
No, we're cool now
git reset --hard
Has anyone else seen it?
Yes
git revert {COMMITISH}
(I do not think it means what you think it means.)
Not yet
How long ago?
Last commit
What would make this better?
I forgot to add a file
git add {myfile}
A better message
git commit --amend
Just throw the last commit away
git reset --hard HEAD^
It seems like forever ago
Is it already on GitHub?
No
Should we remove merge commits?
That would do the trick
git rebase origin/{branch}
Yes
No, I need to change history
Take a mulligan?
Yes
We'll reset and commit from scratch
git reset {COMMITISH}
Is anyone down stream?
No
No
We're going to do an interactive rebase!
Yes
Yes
Enough to form a lynch mob?
No
Do you hate them?
Yes
No
DangerZone™
git rebase -i {COMMITISH}
...
git push --force origin {branch}
It's safest to let it stay ugly then
Send them a note, let 'em know you're changing history.
Everything is going to be fine