

# git gud

tek

may 9, 2019

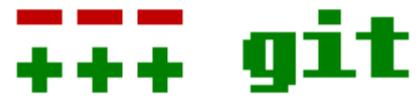
phil loctaux

# whoami

- phil
- (trying to) use  for 4 years
- not an expert
- the internet is your best friend when you don't have an answer

**what i'm about to show  
you is what i know of git  
as of right now**

# back in my day...



*where does git come from?*

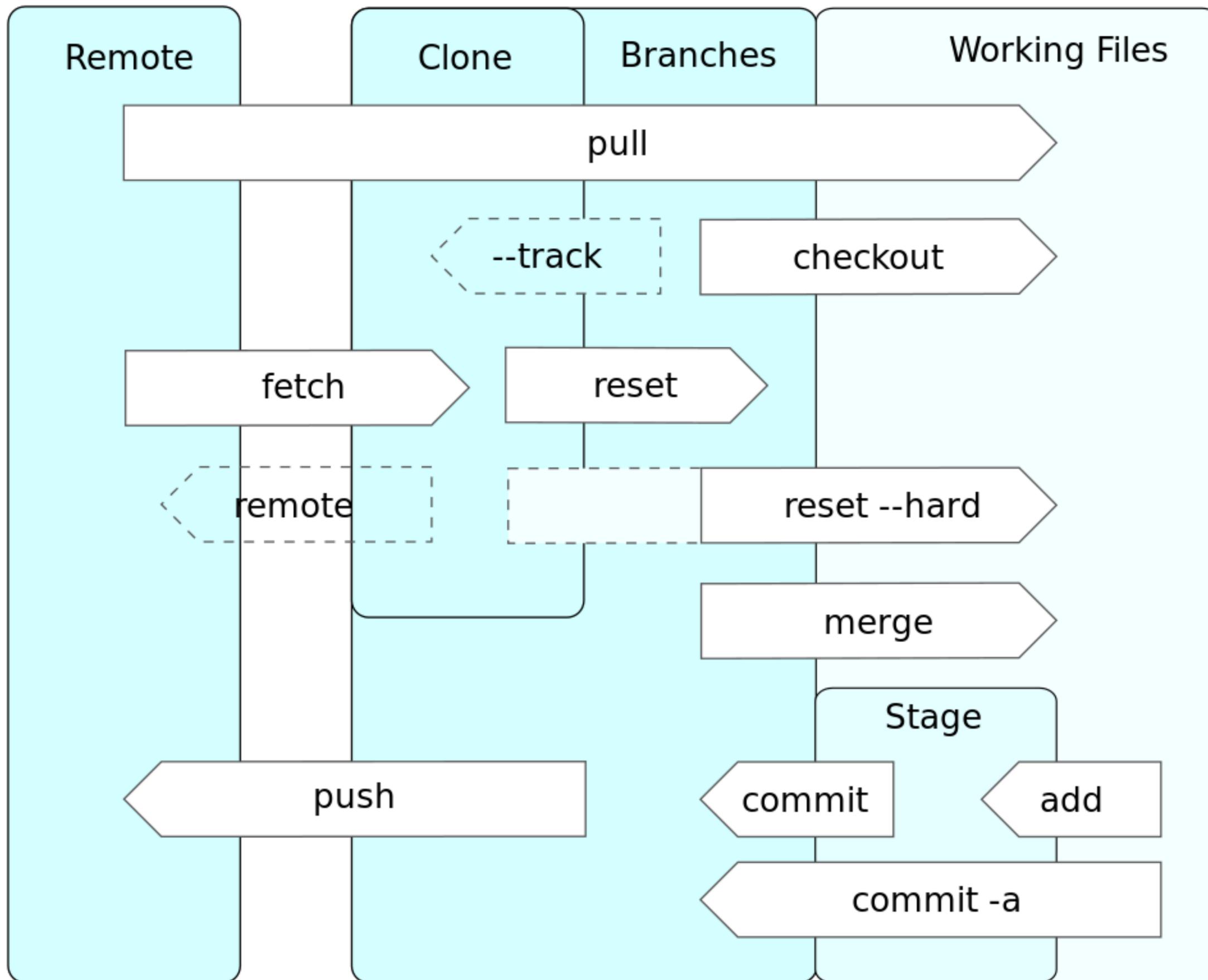


git is made by and for linux kernel development  
before git, patches, tarballs and emails were used (still used as of today)

*april 2005*

Linus Torvalds started to write git  
took 2 weeks to get something stable  
self hosted in couple of days [0]

[0] <https://github.com/git/git/tree/e83c5163316f89bfbde7d9ab23ca2e25604af290>



# let's go custom

**~/.gitconfig**

**[user]**

name = Philippe Loctaux

email = p@philippeloctaux.com

**[core]**

editor = vim

**[color]**

ui = auto

status = auto

diff = auto

branch = auto

[git book reference](#)

[config man page](#)

# let's get started

**git clone**

when you already have a remote repository

**git init**

when you don't have a remote repository

or

just want git to start tracking your work

**[clone man page](#)**

**[init man page](#)**

# afraid to commit?

**regular** commits and good **messages**  
are important to know what's what

commits help you **keep track** of your work

with a nice message you know what you did,  
useful if you need to go back in time to fix an nasty bug

**[BUG]** or **[FEATURE]** commits headers  
can be useful to quickly find a specific commit

# forgot something in your commit?

## forgot a file

add your files using *git add xyz* followed by  
*git commit --amend* will add work on the tip of the commit

## typo in commit message

[add man page](#)

*git commit --amend* and fix your commit message

[commit man page](#)

# show changes (aka diffs)

*git diff* between local modifications and last commit

*git diff master..feature* between **master** and **feature** branches

*git diff 33b1ea1..134ccf5* between commits

*git diff --staged* to show staged changes

# remove local modifications

*git checkout -- filename.xyz* will revert the local modifications to the latest commit

[diff man page](#)

[checkout man page](#)

# branch out

*git checkout -b mybranch* creates branch **mybranch** and switches to it

*git checkout master* switches to branch **master**

*git branch -d mybranch* deletes branch **mybranch**

push your work on that branch on your remote: **git push origin mybranch**

[checkout man page](#)

[push man page](#)

# stash

useful when you need to change branch  
but don't want to commit your work

*git stash* to save current directory

*git stash pop* to resume working

# rebase

apply commits from a branch to the current branch

*git rebase mybranch* will rebase current branch from **mybranch**

*WARNING* : this often causes conflicts!

# what happened here?

*git log* shows all commits on your current branch

*git show xyz* shows the commit info of **xyz**

*git shortlog --summary* shows the number of commit per person

*protip: git log --oneline* is better when you have a long commit history

[log man page](#)

[show man page](#)

[shortlog man page](#)

# work tree at commit xyz

*git checkout xyz* gets working tree at specified commit

*git checkout -b derive xyz* will create branch derive from commit **xyz**

# go back in time (the hard way)

do this only if you didn't push, or if you know what you're doing

this rewrites history, which may (*will*) break the repos of your co-workers

***git reset --hard xyz*** sets you back to commit **xyz**  
(and deletes all your work since that commit)

[reset man page](#)

# go back in time (the soft way)

this makes a revert commit for every commit to be reverted,  
which may clutter your commit history

for each commit, git will pop out your favorite text editor,  
just save and close the file to complete the revert

## one commit

*git revert xyz* reverts commit **xyz**, by doing the opposite of what **xyz** did

## range of commits

*git revert abc..xyz* will revert from commit **abc** to **xyz**

# multiple remotes

a repo can exist on multiple servers. they are called remotes in git  
the default remote is called **origin**

*git remote* shows remotes registered in the repo

*git remote add name url* adds a new remote with a given **name** and **url**

*git remote remove name* removes the remote **name**

to push on a specific remote, specify its name in the git push command, for example:

*git push myremote mybranch* will push the branch **mybranch** on the remote **myremote**

# ignoring junk

entering “*the gitignore*”

**.gitignore** is a file at the root of a git repo with a list of files that should not be committed to git

useful for binaries, objects, libraries, logs,  
backup files made by text editors, junk system files, etc

use [this cool website](#) **[0]** to generate a gitignore for your project

**[0]** <https://gitignore.io>

[gitignore man page](#)

# who wrote *that*?

*git blame filename* will show information about **filename**

- \* all lines of the file saved in git
- \* who wrote each line
- \* when that line was committed
- \* the commit hash

this way, when someone breaks your project, you *know* who to blame!

# to infinity, and beyond!

the purpose of this talk is to **encourage** you to learn more about git

try learn git branching **[0]**

read the git book **[1]**

git problem? go on stackoverflow **[2]!!!**

keep trying new stuff, or just ***rm -rf*** and start again

***man git***

**[0]** <https://learngitbranching.js.org>

**[1]** <https://git-scm.com/book>

**[2]** <https://stackoverflow.com>

**thx!**

<https://x4m3.rocks/talks/git-tek.pdf>

