Version Control Systems

the **diff** command

- Suppose I have a text file **a.txt**
- I make a copy of it: **cp a.txt b.txt**
- With a text editor, I make some changes to **b.txt**

- Running the command **diff a.txt b.txt** will show where the changes were made.

```
yoav-freunds-PowerBook-G4-retina:diff yoavfreund$ diff 03_numpy_vs_native.ipynb 03_numpy_vs_native.ipynb.1
77c77
<         "(array([ 0.45880794,  0.17875314, -0.18561953,  0.51020662,  0.48870627]),\n",
---
>         "(array([ 0.45887094,  0.17875314, -0.18561953,  0.51020662,  0.48870627]),\n",
yoav-freunds-PowerBook-G4-retina:diff yoavfreund$
```

- instead of storing **a.txt** and b.txt. We can store **a.txt** and **diff a.txt b.txt**

diff and version control

- Typical software projects today consist of 100,000 - 10,000,000 lines of code.
- Code changes, big and small, are made by a large and changing team of programmers. This takes place every day for years.
- It is critical to keep track of these changes.
- Instead of storing each version of each file, one can store only the **diff**s. Any version can be reconstructed from these diffs.
- Version control systems use the diff method to allow programmers to collaborate effectively.
- Public version control systems have evolved from RCS,CVS,SVN to the more modern Mercurial and GIT
- GIT was developed by Linus Torvalds to support the thousands of volunteer programmers around the world that contribute to LINUX.
- There are also many commercial systems, that can take care of many more operations such as testing and deployment. We will focus on GIT

Really Quick Intro To Git

Hasan Veldstra
<hasan@hypernumbers.com>

quick poll: Git users, SVN users.

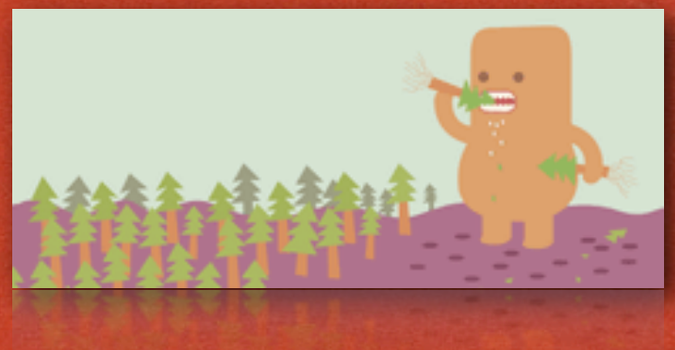
hypernumbers

like CVS or SVN, only much better.
fundamental difference – distributed.

source control system

the logo: git

an artistic impression:



why Git?

why Git?

- makes stuff cheap
  - any workflow
    - fast
- small

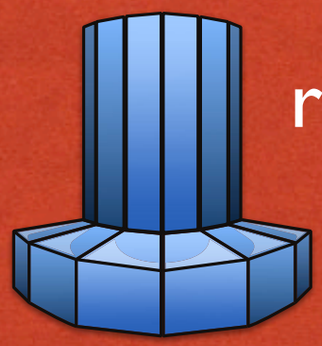
```
$ cd (project directory)
$ git init
$ (create some files)
$ git add .
$ git commit -m 'start the project'
```

distributed

=

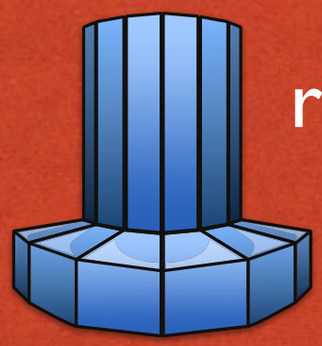
no central repository

centralized (SVN):

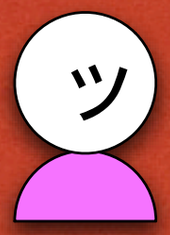


repository

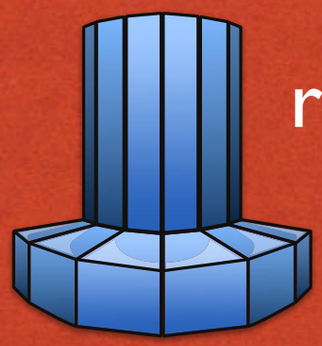
centralized (SVN):



repository

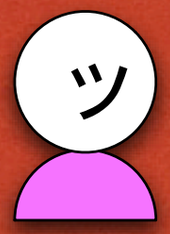


centralized (SVN):

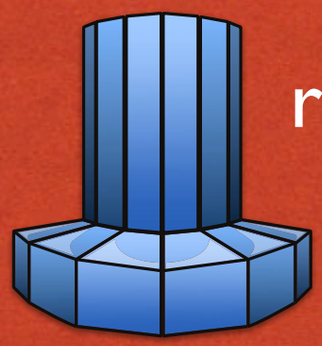


repository

devs “check out” the code to create a “working copy”

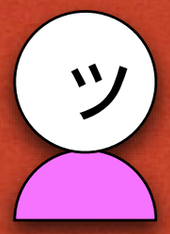


centralized (SVN):

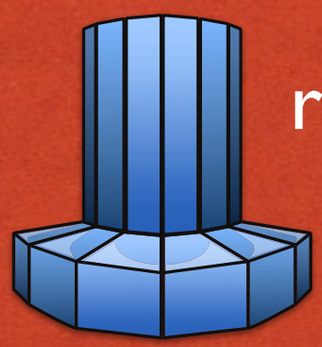


repository

check in



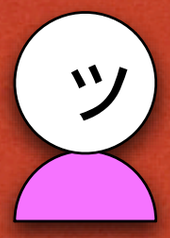
centralized (SVN):



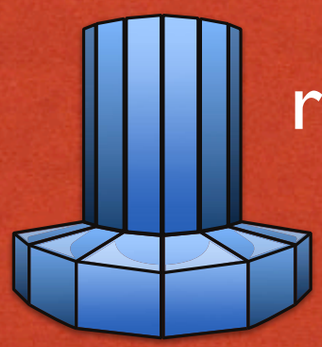
repository

check in

check out



centralized (SVN):

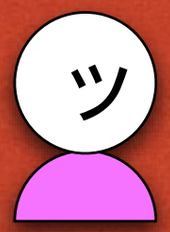


repository

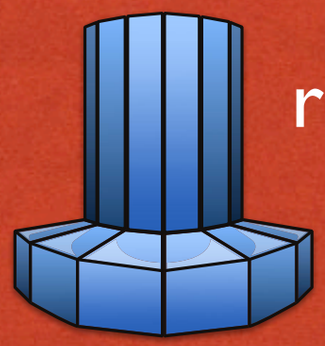
check in

check out

make changes



centralized (SVN):



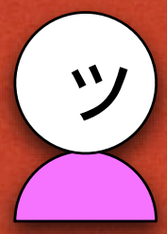
repository

check in

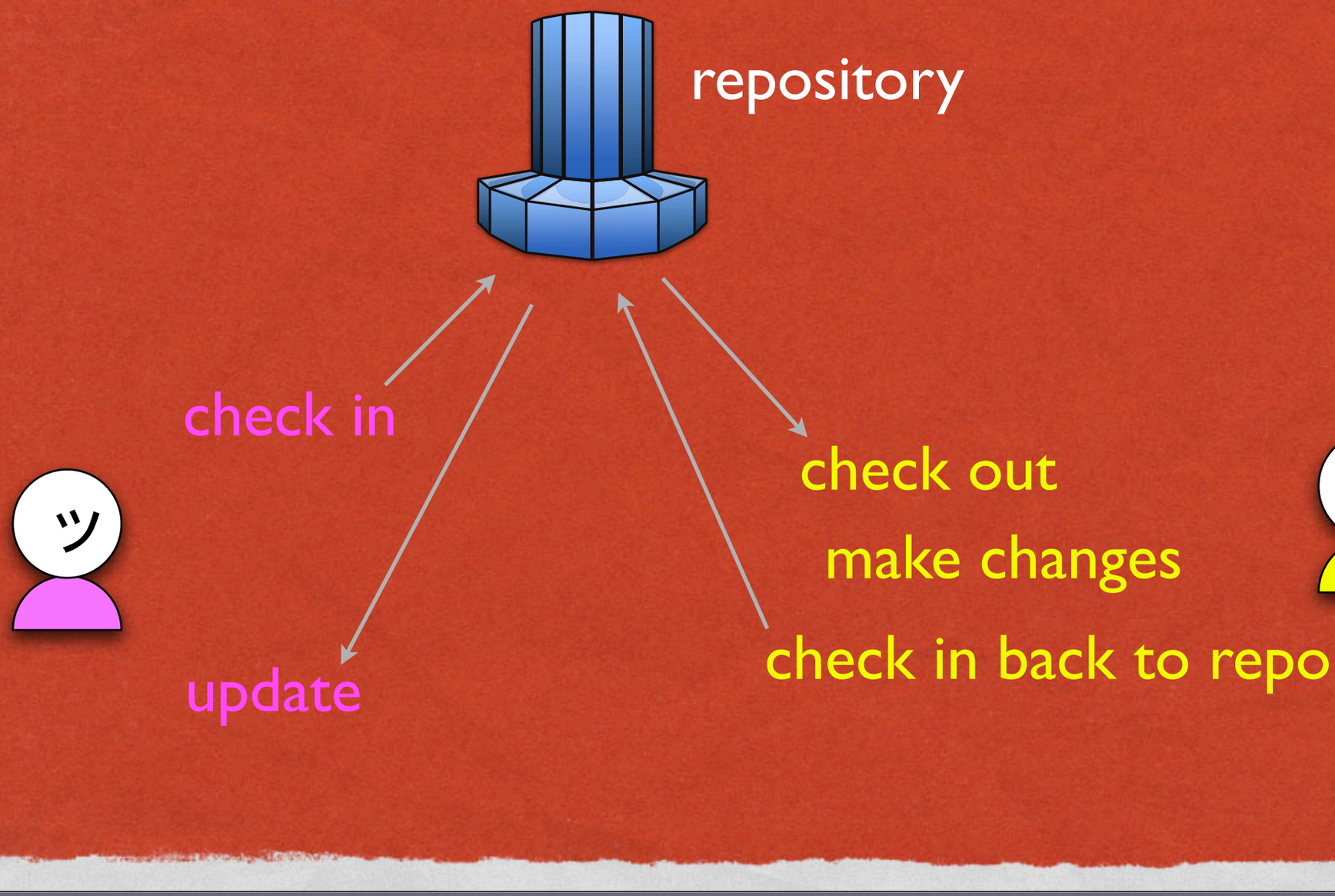
check out

make changes

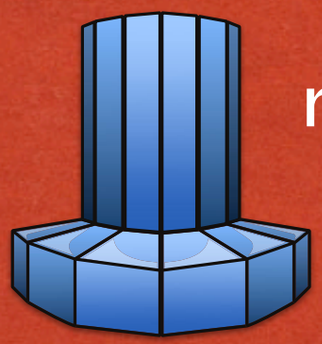
check in back to repo



centralized (SVN):



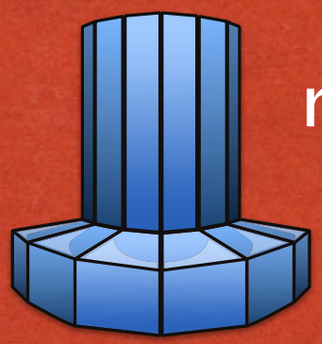
centralized (SVN):



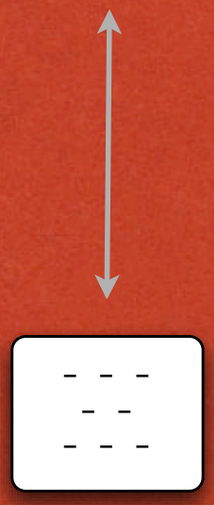
repository

one-to-many relationship

centralized (SVN):

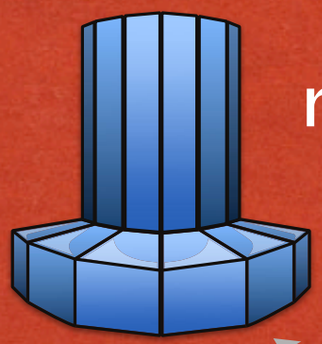


repository

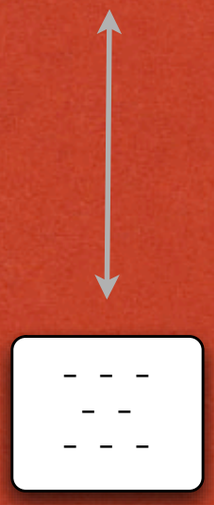


working copy #1

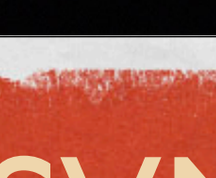
centralized (SVN):



repository

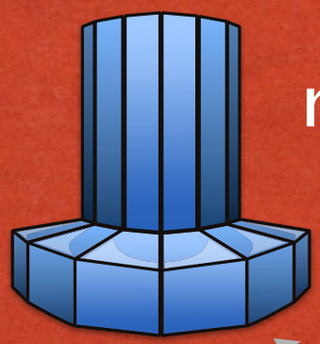


working copy #1

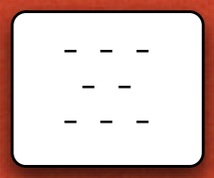


working copy #2

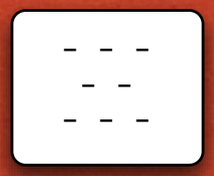
centralized (SVN):



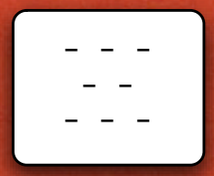
repository



working copy #3



working copy #1



working copy #2

centralized (SVN):

- working copies are inferior

centralized (SVN):

- working copies are inferior
- & can't talk to each other

centralized (SVN):

- working copies are inferior
- & can't talk to each other
- you gotta be online

centralized (SVN):

- working copies are inferior
- & can't talk to each other
- you gotta be online
- single point of failure

distributed (`Git`):

distributed (`Git`):

~~checkout~~

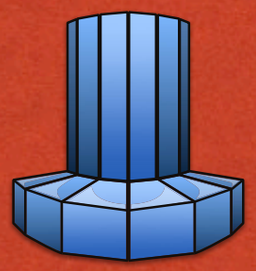
distributed (`Git`):

~~checkout~~

clone

we don’t check out, we clone
clone = full history, can do anything original can

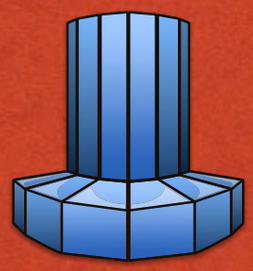
distributed (`Git`):



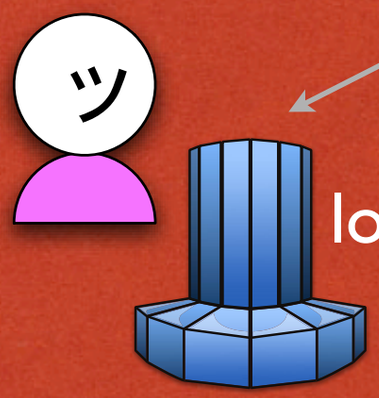
“canonical” repository
For example: GitHub

"canonical" because it's a social convention, not a technical requirement

distributed (`Git`):

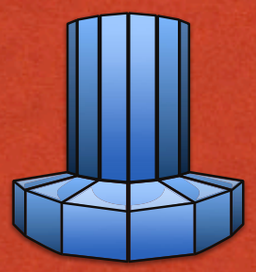


“canonical” repository
For example: GitHub

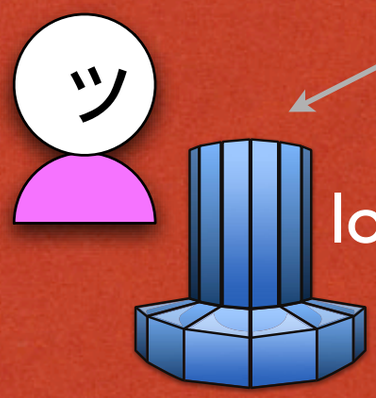


local repository #1

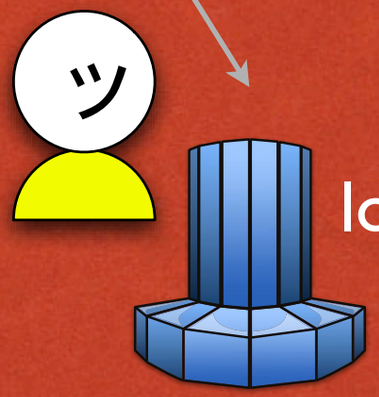
distributed (`Git`):



"canonical" repository

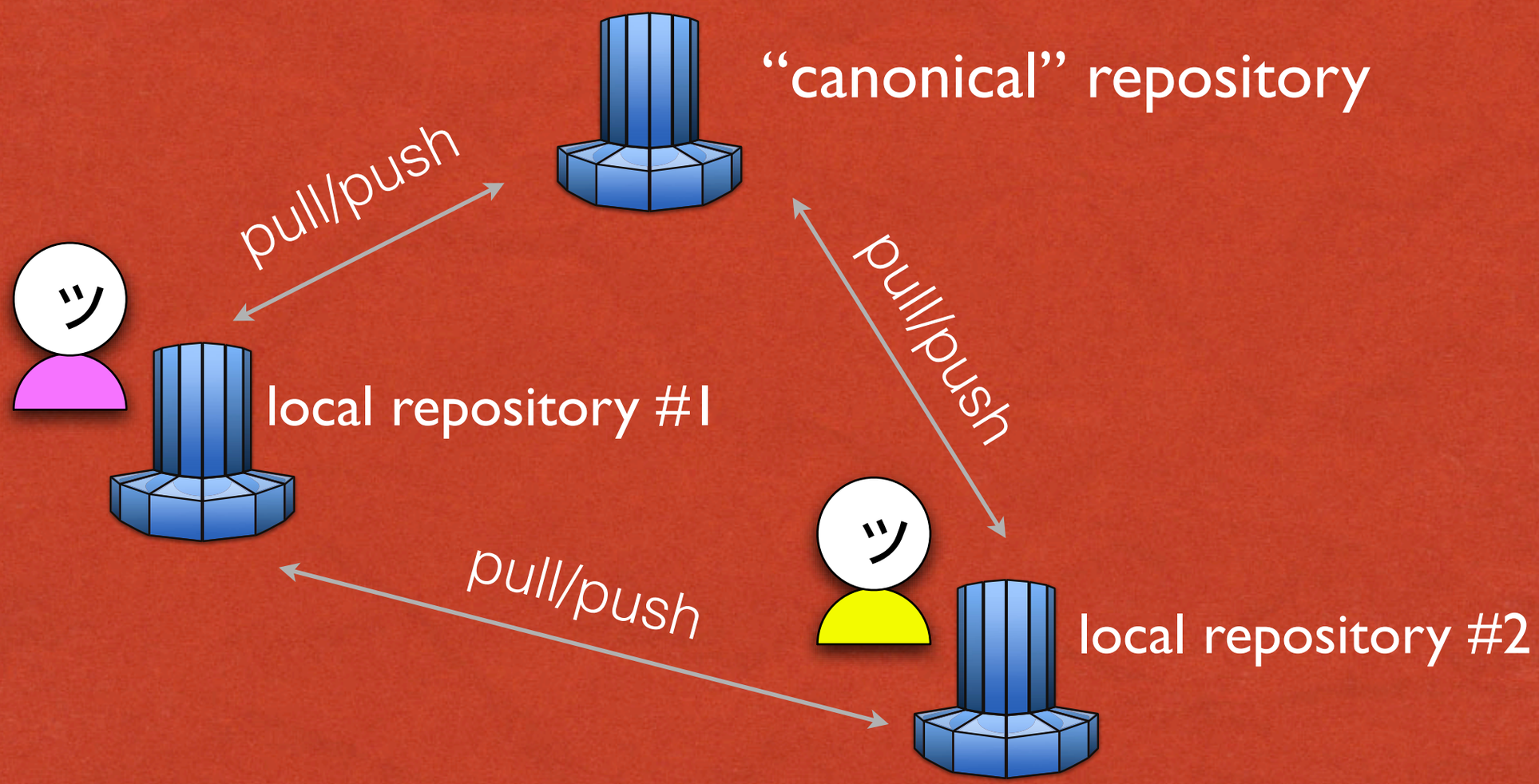


local repository #1

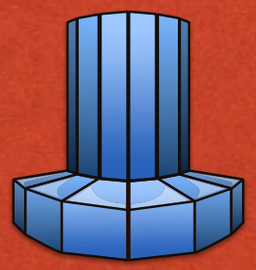


local repository #2

distributed (`Git`):

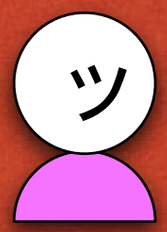


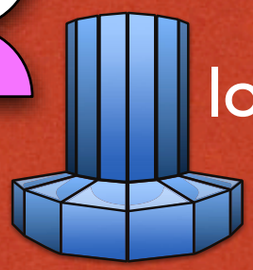
distributed (`Git`):



"canonical" repository

pull/push



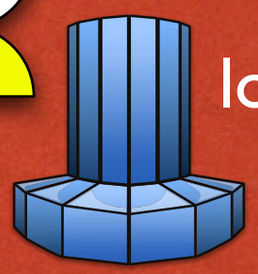


local repository #1

clones are full-fledged repositories => exchange patches and merge between them without touching the main repo.

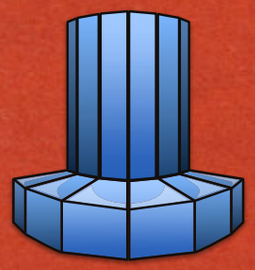
several people can work together on something new amongst themselves, and then push to main when done.

pull/push



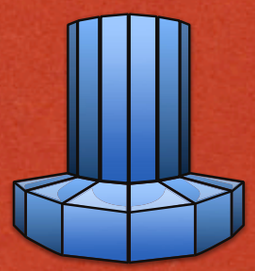
local repository #2

staging area

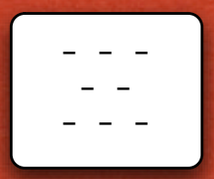


"canonical" repository

like a shelf on which to put things that'll make up the next commit. incremental building of. "chunk commits" = awesome. "tangled working copy" problem gone.

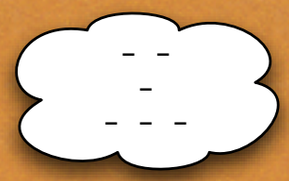


local repository



working copy

object database, trees & blobs that represent the repo (file contents,



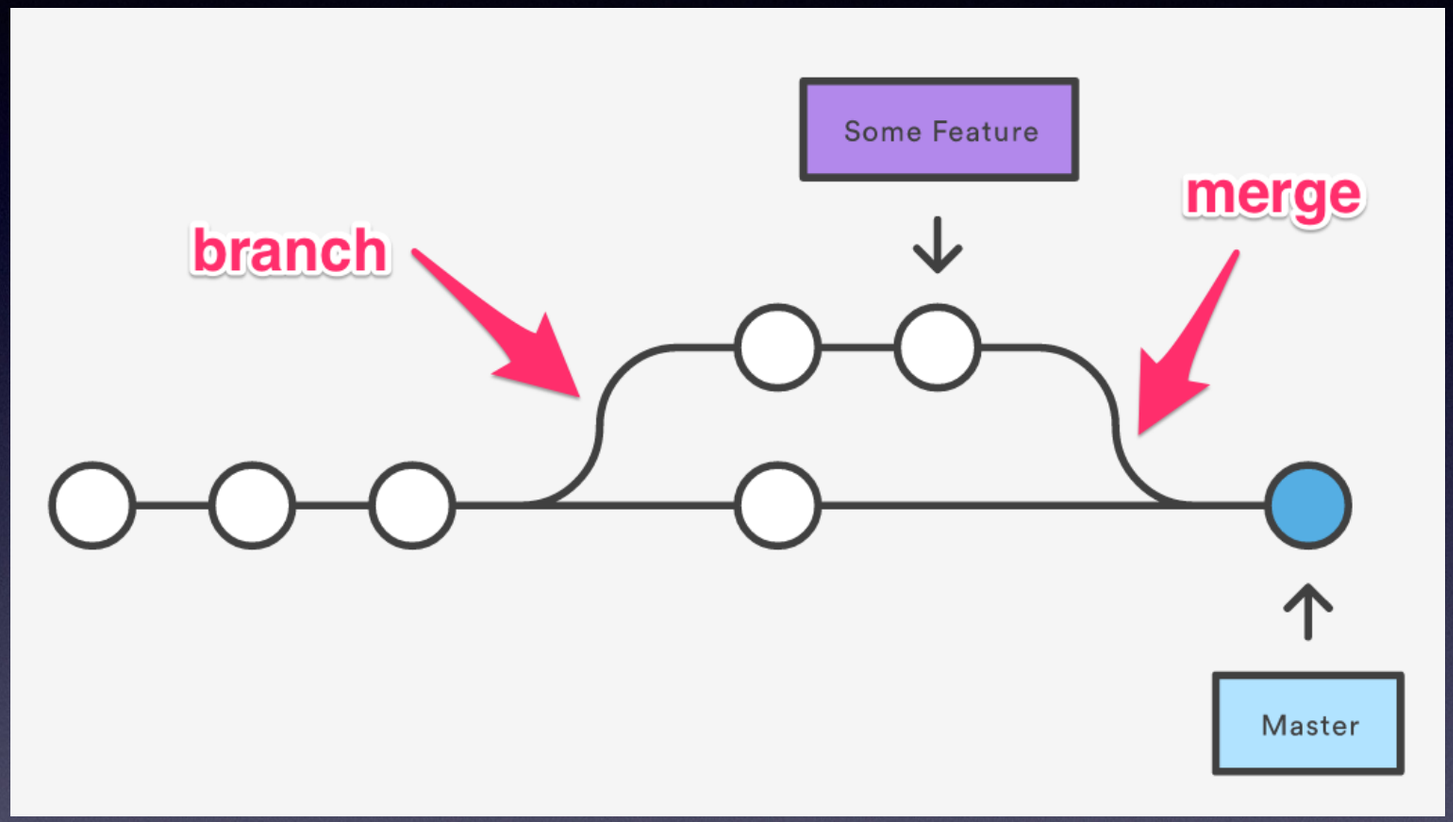
staging area

stuff on your disk tracked by the repo

Branches

- branches are used to isolate work of different people or different projects of the same person (example: test version of the software that uses a different library)
- switching between branches: **git checkout**
- merging branches: **git merge**

Example of branches



Remember: each copy of the repository can hold any and all of the branches

Git is for source-files

- Git makes heavy use of **diff**
- works very well for code/text/markdown
- Any file versions ever committed, can be reconstructed.
  Nothing is ever lost. (Unless the whole repository is deleted)
- Works poorly for machine generated files (compiled source)
  A small change in the source results in big differences after compilations.
- **Not for data files!** Github puts an upper bound of 100MB per repository.

GitHub and iPython notebooks

- GitHub repositories can contain .ipyb files
- **The good news:** Github recognizes the .ipyb format and will render the notebook inside the web site.
- **The bad news:** outputs and plots are large files for which **diff** does not work very well.
    - Solution: Clear all of the cell outputs before committing to Git.
- **.ipynb_checkpoints**: ipython stores each saved version in this directory. This is orthogonal to github. Best to not track this directory in git.
    - Solution: create a file called .gitignore in the notebook directory and put in it the line **.ipynb_checkpoints**

Recommended further reading:
https://www.atlassian.com/git/tutorials/