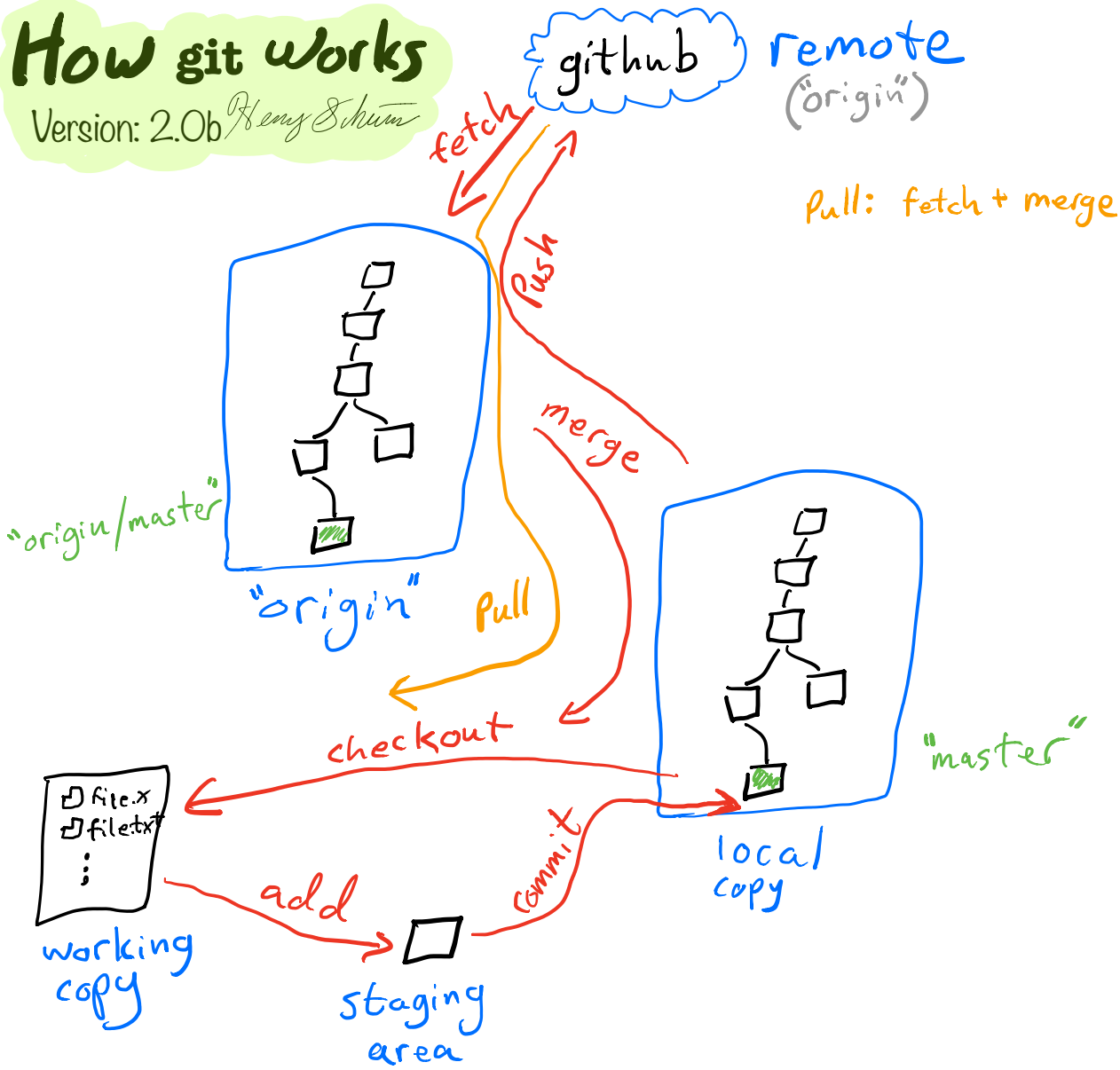
How git works

Version: 2.0b Henry Shum

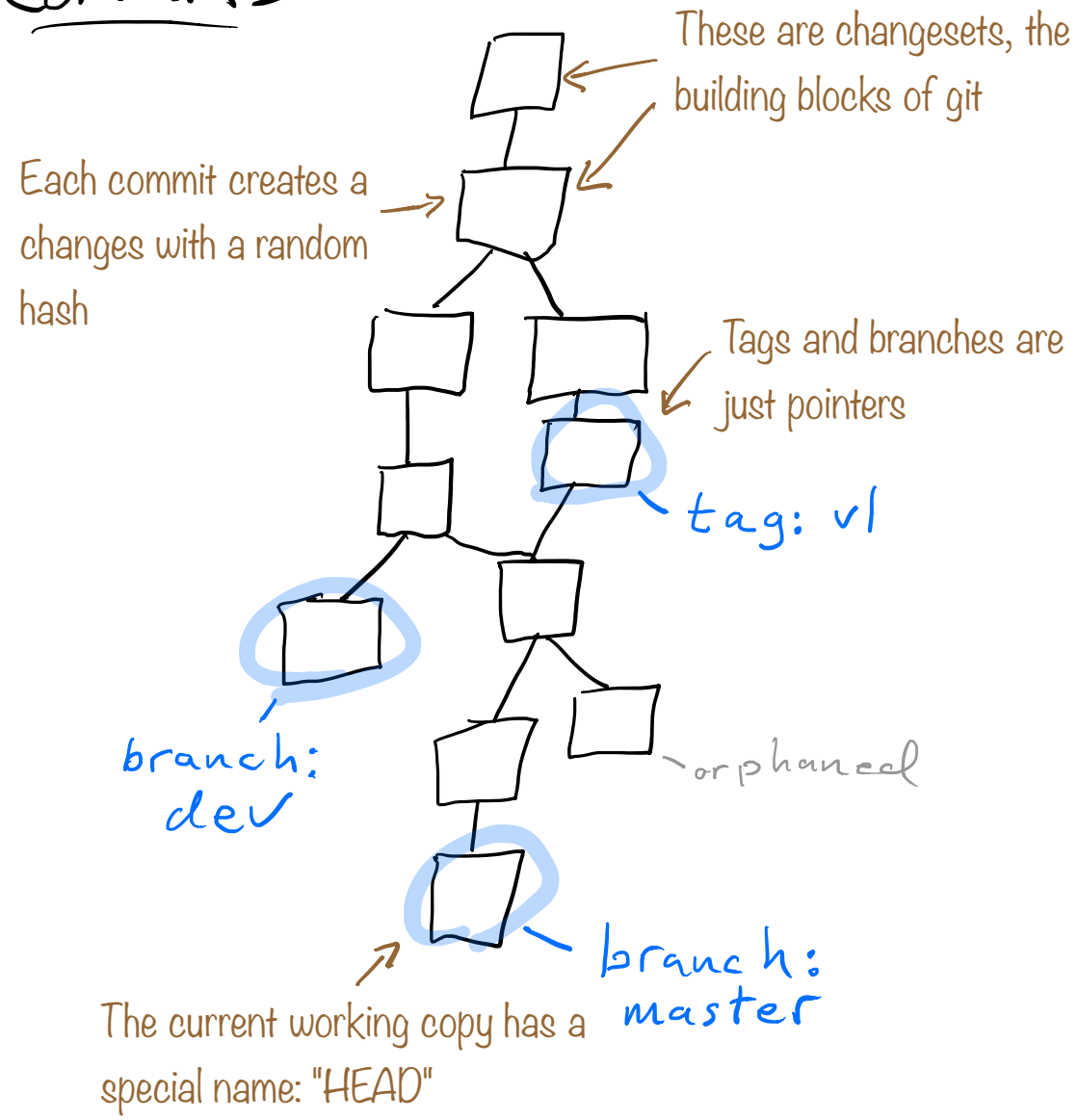


- Merge combines trees, and checks out the result
- Pull does a fetch, then a merge

If you only can remember one command:

git --help	Get common commands and help
git <command> --help	How to use <command>

Commits



Features of changeset design:

- Only changes stored, compressed -> Very efficient
- Random hashes: multiple people can make commits locally

Handy commands:

git status	Show lots of useful details — use often!
git log	Show history (--pretty=oneline)
git add -u	Add all changed files — newer git will only add current dir and subdirs
git diff	Show changes vs. staging area
git diff HEAD	Show changes vs. last commit — or diff <a> <b> for any two things. Can include filename.

Get an existing repository: (see later page for more options)

git clone git@github.com:<username>/<repo>.git — assuming ssh is set up

Make a new repository:

```
git init
git remote add <repository> origin
<add files, commit>
git push -u origin master
```

— or other name

— makes git remember where to push

Standard procedure:

```
git pull
git add <files>
git commit -m "My message"
git push
```

— or leave off, then use editor

Tagging:

git tag	List all tags
git tag <tag name> -a	Make a new tag (annotated)
git push --tags	Push tags to remote

Standard names:

github bitbucket gitlab

- **Remotes:** Computers to push to or pull from — usually origin or upstream
  - **origin:** The default name for the first remote
- **Branches:** A moving pointer to commits
  - **master:** The default name for the first branch — local
  - **origin/master:** Remote branches are also available — copy of remote
    Very rarely used directly!
  - **HEAD:** Special term for the latest commit
- **Tags:** Like branches, but usually stationary
  please don't be ROOT and move them!

Changing to new commit:

`git checkout <existing>`	Checkout commit, tag, or branch
`git checkout -b <branch>`	Make new branch and checkout
`git checkout HEAD^`	Go back one commit

can add more than one

Helpful extra tools:

`git grep "term"`	Search text only in repository (fast)
`git ls-files`	List files in repository

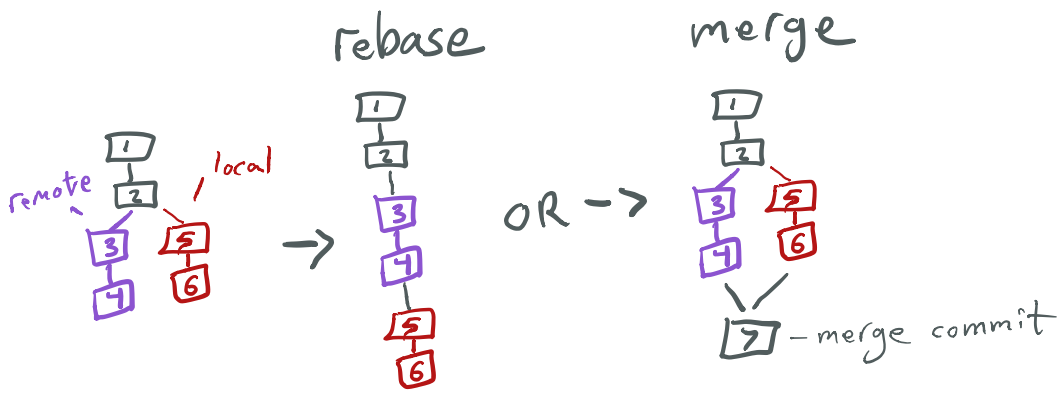
awesome!

Useful but dangerous commands:

`git reset`	Unstage staging area, no change to working copy
`git reset <commit>`	Move current branch pointer
`git reset --hard`	Wipe all working copy changes Forever!
`git stash`	Put all changed files in a local stash
`git stash apply`	Put last stash back in working dir
`git stash pop`	Like above, but also remove stash

Combing changes:

git pull --rebase Rewind history, then replay changes
Much nicer history!



What happens if there is a conflict?

- Different files changed -> both added
- Different parts of one file -> both parts
- Changes to the same line(s) -> "Merge confict", presents diff
  - Use **git mergetool** for graphical solution
  - Or just edit the file and **git add**

Why use **git pull** instead of **git pull --rebase**?

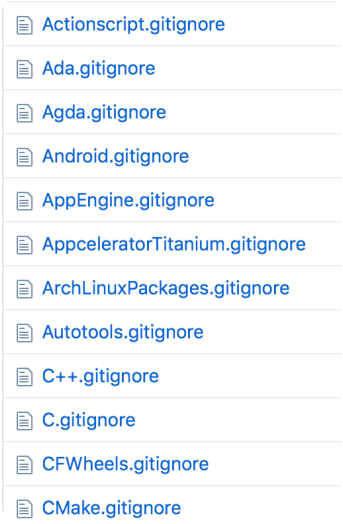
- Less typing
- Slightly easier; rebase will not run if there are working copy changes
  - Just **git stash**, **git pull --rebase**, and then **git stash pop**

Special files:

.git/config	Local configuration (easy to edit)
.gitignore	Any file listed will not be shown or (easily) added
.gitkeep	Undo gitignored files
.gitmodule	Used by git submodule (below)

Git ignore files:

- Can be in any directory (only affects directory and subdirectories)
- Prepared .gitignore files for many languages (LaTeX, C++, Python, etc) are available
- Always add at least editor autosave files!
- Use `git status --ignored` to see ignored files



https://github.com/github/gitignore

Advanced: SubModules

Following commands must not be run in sub directory

`git submodule add ../../<username>/<reponame>.git local_dir`

Adds a git repo as a sub directory

`git submodule update --init --recursive`

Initializes and updates modules (needed after clone)

`git submodule deinit -f .`

Wipe out all submodule checkouts (fixes problems in URLs)

Pretty safe, but will clear changes

All submodules behave like normal repositories when inside them

Adding the submodule like a normal file remembers the git hash of the module

Advanced: Cloning

git clone <url> <local folder>

--depth=N	Only download last N commits — only use sparingly
--recursive	Also get all submodules - always a good idea
--branch=<branch>	Auto-checkout a branch

Advanced: History rewriting

These are safe if you have not pushed changes

very useful if not shared

git commit --amend	Modify last commit (staging area or change msg)
git merge --squash ...	See online for usage, combines commits

If you are working on your own branch, this can be used:

git push -f	Push changed history

Online:

Fork: A copy of a git repository you own

Pull request or Merge request: Merge your branch or fork to original repository

Issues: A place to ask or report things

Mentions: Use @username or #number to mention user or issue/pull request

Gitisms: (how one works in git)

Make a branch, work in it, merge with rebase or squash, throw away branch

First line of a commit message is overview, and shown in logs/online lists

Commit often, but each commit should run/compile