Git

8/26/2010

What's Git

- Distributed version control system
- Started by Linus Torvalds
    - “I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.”
- In use by several major projects
    - Linux kernel, perl, gnome, qt, android, ruby on rails, wine
- … and tons of smaller OS projects
    - http://github.com/repositories

How does it work?

- Forget everything you know about version control systems
  - Git is not CVS's SVN
  - An entirely different beast

What if all we had was a filesystem, a text editor and some basic commands?

Building a git-like system from the ground up

(paraphrased from http://tom.preston-werner.com/2009/05/19/the-git-parable.html)

Snapshots

- Start project in a directory named `working`
- Write one feature at a time. When a feature is complete, copy all files into a snapshot directory
  - Named `snapshot-1`, `snapshot-2`, ...
  - Add a `message` file to the snapshot directory with
    - Summary of change
    - Date

Branching

- Some snapshot becomes a release (e.g., `snapshot-85`)
  - … and development goes on: `snapshot-86`, `snapshot-87`, ..., `snapshot-110`
- You need to fix a bug in the released version
  - Copy `snapshot-85` to working, fix issue and create a new snapshot: `snapshot-111`

Branching

- Implicit relationship between snapshots:
  - `snapshot-(N+1)` follows `snapshot-N`
- Assumption no longer valid
  - `snapshot-111` follows `snapshot-85`, not `snapshot-110`
- Easy fix!
  - Record the id of the parent snapshot in the `message` file

Branch names

- Identify branches
  - Name
  - Keeping track of latest snapshot within a branch
- `branches` file with name → snapshot pairs
  - Every time we create a new snapshot, update the corresponding pointer

Tags

- Label specific snapshots
- Similar to branches (just a pointer), but they don't move as new snapshots are created
- `tags` file

Sharing work with others

- Share all your `snapshot-xyz` directories, `branches` and `tags` files
- Both make changes to the main branch and create a snapshot … with the same name, but different contents!
  - How to share each other's changes?

Sharing work with others

- Solution: use hashes to name snapshots
  - SHA-1 of the contents of the `message` file
- Also, add name/email of author to `message` file
- Snapshots created by different people can be merged together without fear of collisions

Merging

- A new snapshot is created to record the changes needed to make both branches identical
    - Special snapshot that contains a pointer to both parent snapshots in the `message` file

Staging area

- Sometimes, you get sidetracked and add more than one unrelated change to the working copy
  - Introduce a `staging` directory
  - Snapshots are now created from the staging directory
  - Pick and choose which changes from `working` are applied to `staging`

Duplication

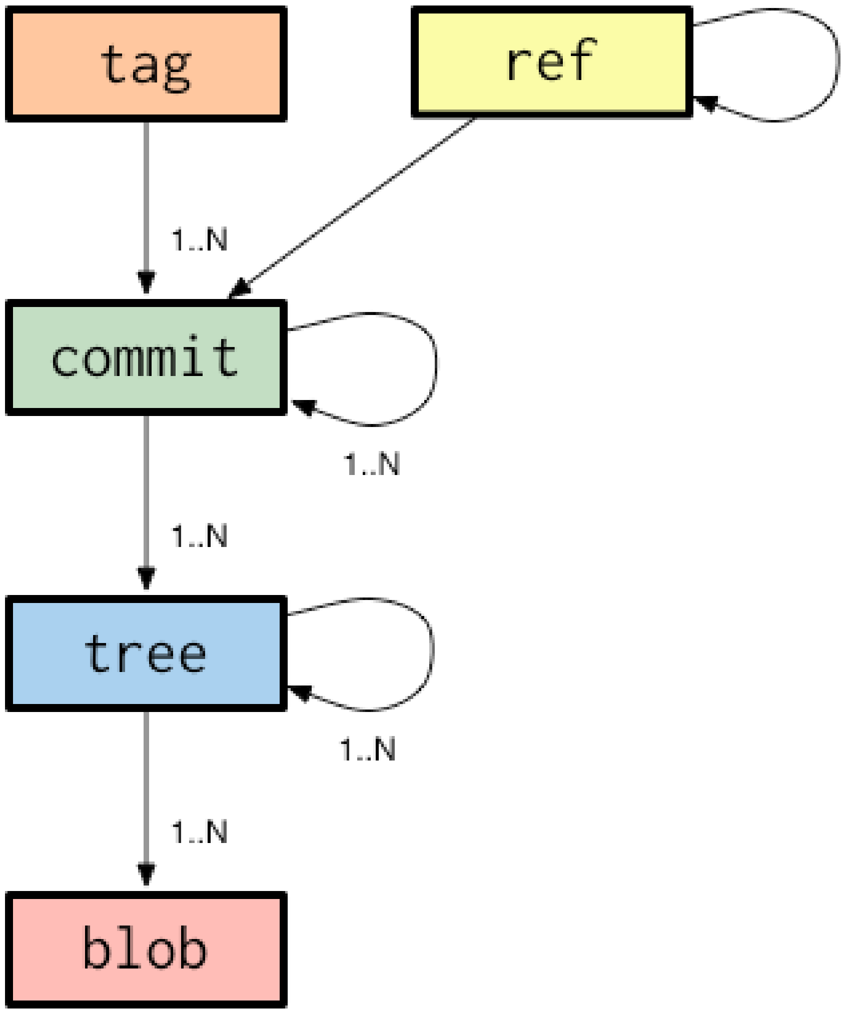
- Each snapshot will likely differ from others in just a handful of files
  - Lots of duplicated data across snapshots
- Idea:
  - “there's no problem in CS that can't be solved by adding one more level of indirection”
  - replace contents of each snapshot directories with a single file mapping filename → id
  - compute id of each file by applying SHA-1 to its contents
  - store files in an `objects` directory, using SHA-1 as filename
- Bonus: compress files

What's Git (revisited)

- A distributed, replicated, content-addressable, snapshotting, nonlinear, hierarchical content management system
  - Just a bunch of tools to facilitate managing snapshots, files, references, etc.
- Not far from the model we just described
  - but a lot smarter
  - many more features

A tour of Git

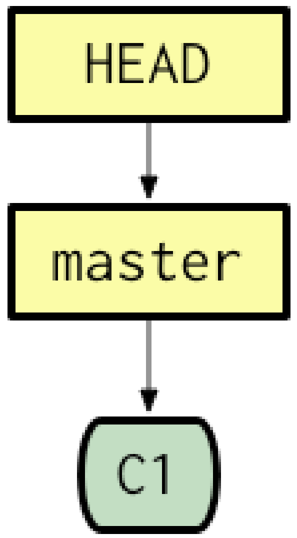
- 4 types of **immutable** entities
  - tag
  - commit
    - a “snapshot”
  - tree
    - represents a directory
  - blob
    - a “file”



A tour of Git

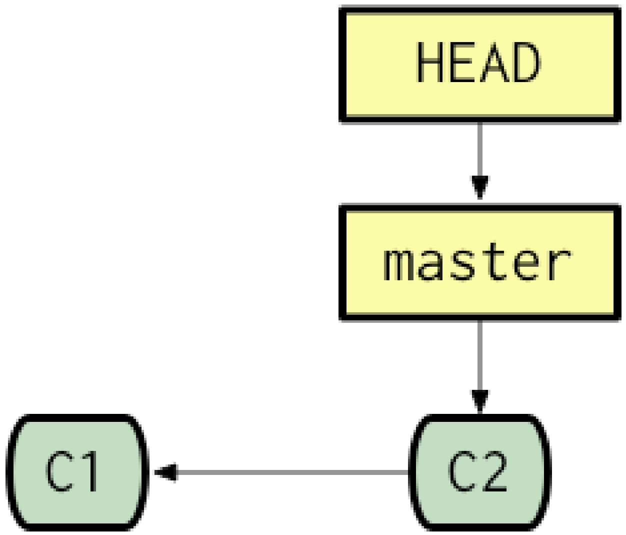
- refs
  - Moveable pointers to a commit
    - branches
    - HEAD
      - Pointer to the currently checked out commit
    - remotes
      - Pointers to branches in remote repositories

Init and first commit


$$\begin{tikzcd}\boxed{\text{HEAD}} \arrow[d] \\ \boxed{\text{master}} \arrow[d] \\ \text{C1}\end{tikzcd}$$

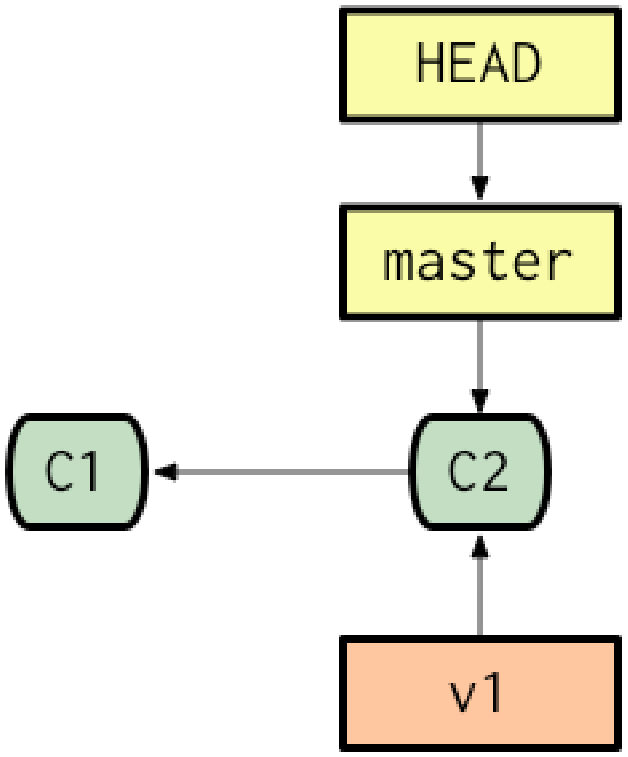
```
git init

echo 1 > file.txt

git add file.txt

git commit -m “first”
```

Making changes



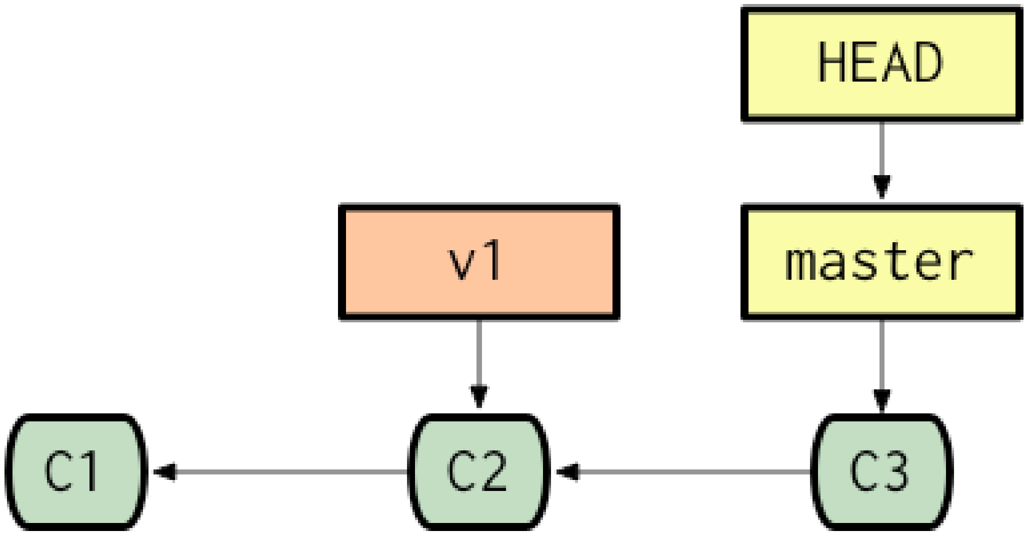
```
echo 2 >> file.txt

git add file.txt

git commit -m “second”
```

Tagging



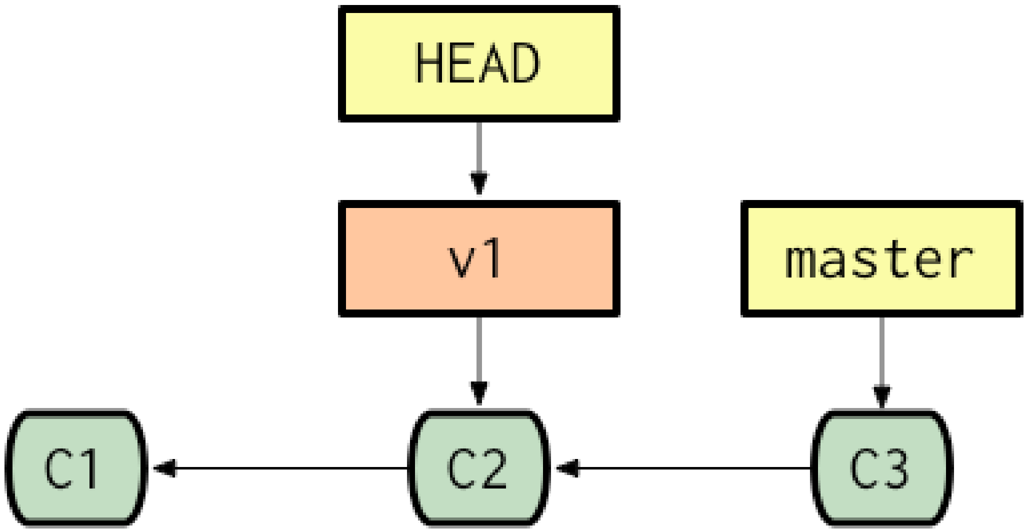
```
git tag v1
```

More changes



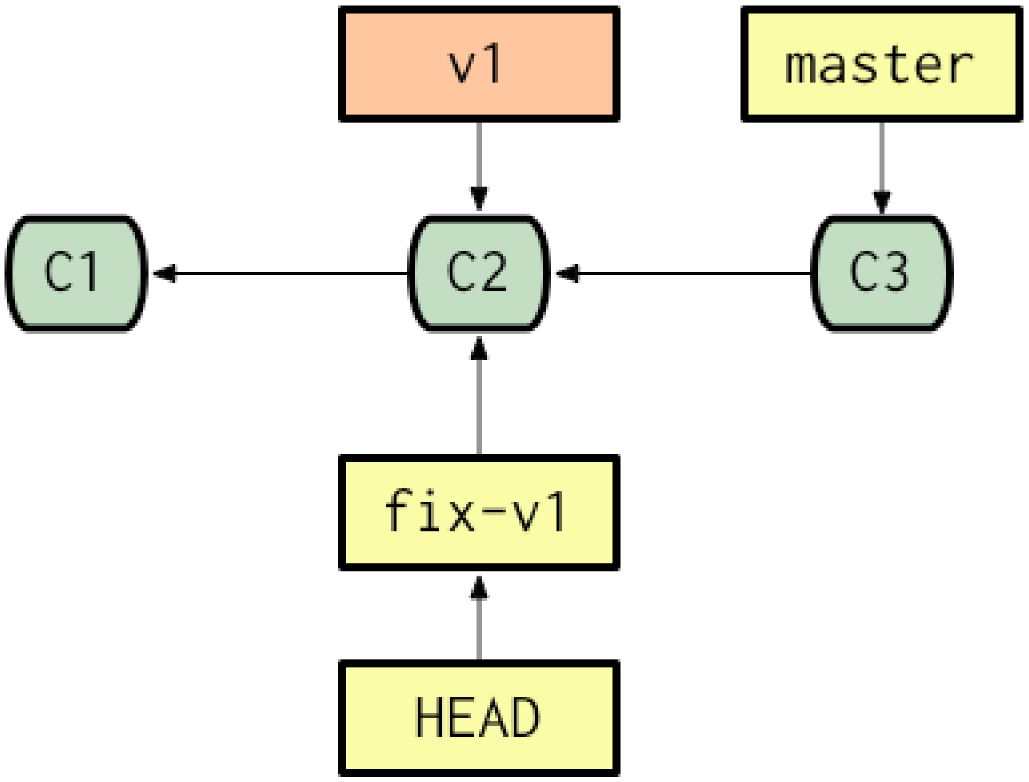
```
echo 3 >> file.txt
git add file.txt
git commit -m “third”
```

Checking out based on a tag



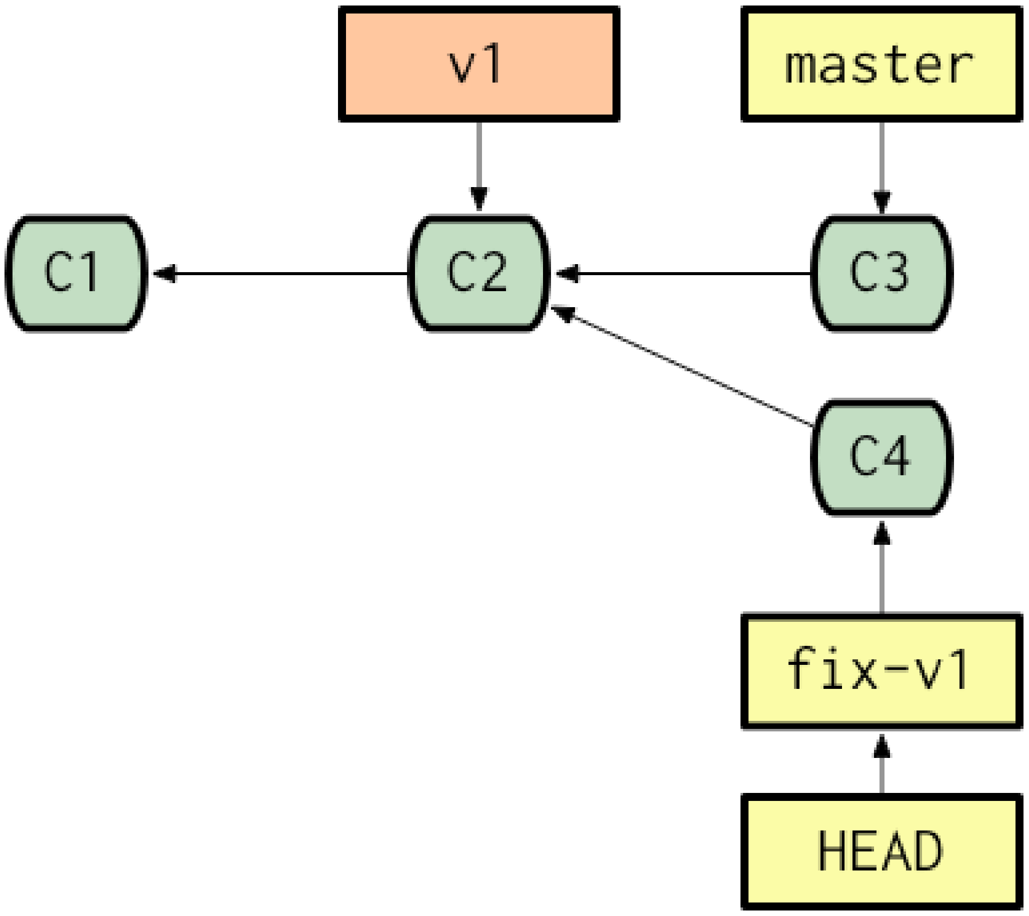
```
git checkout v1
```

Creating a branch



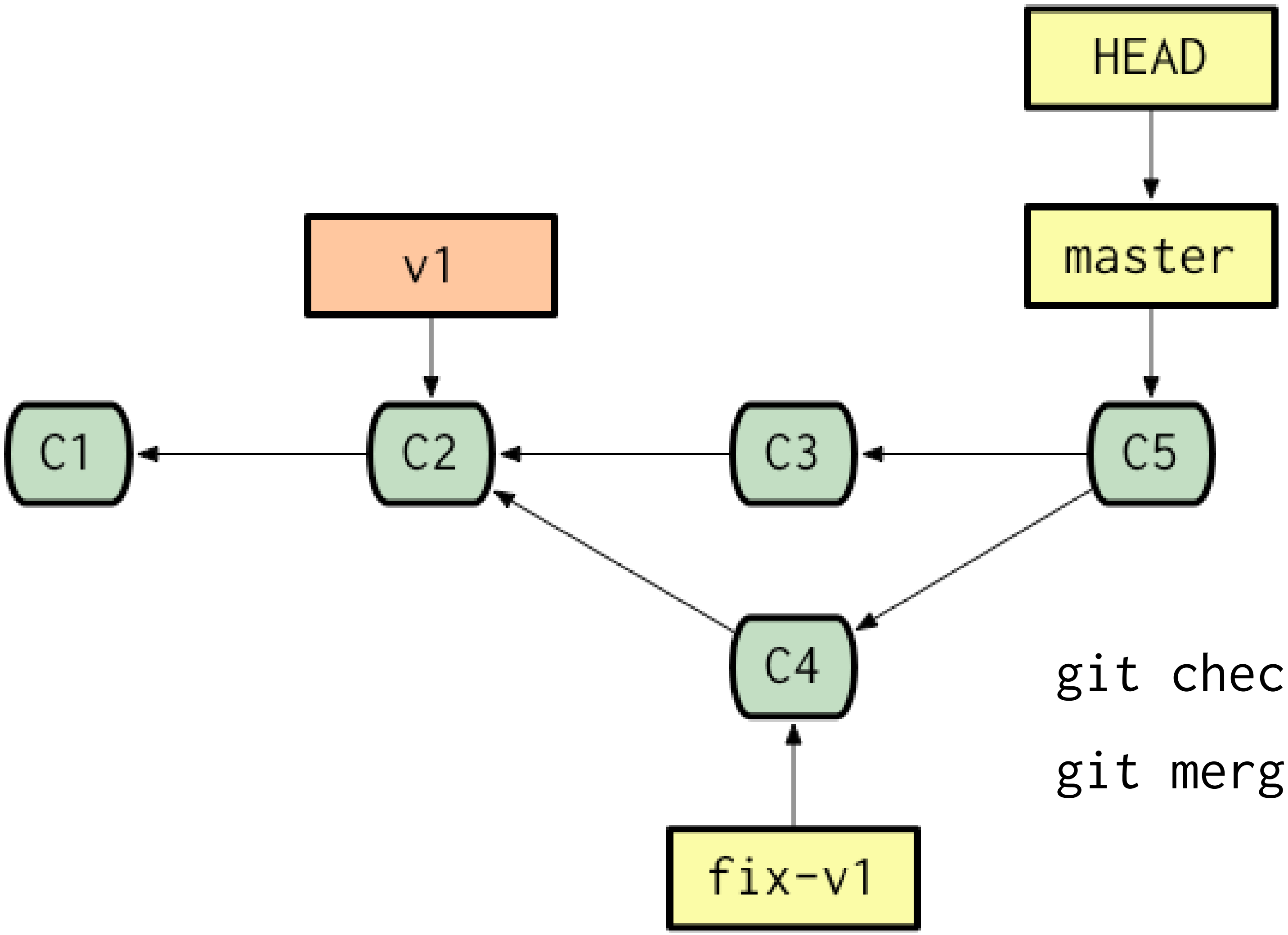
```
git branch fix-v1

git checkout fix-v1
```

Changes in new branch



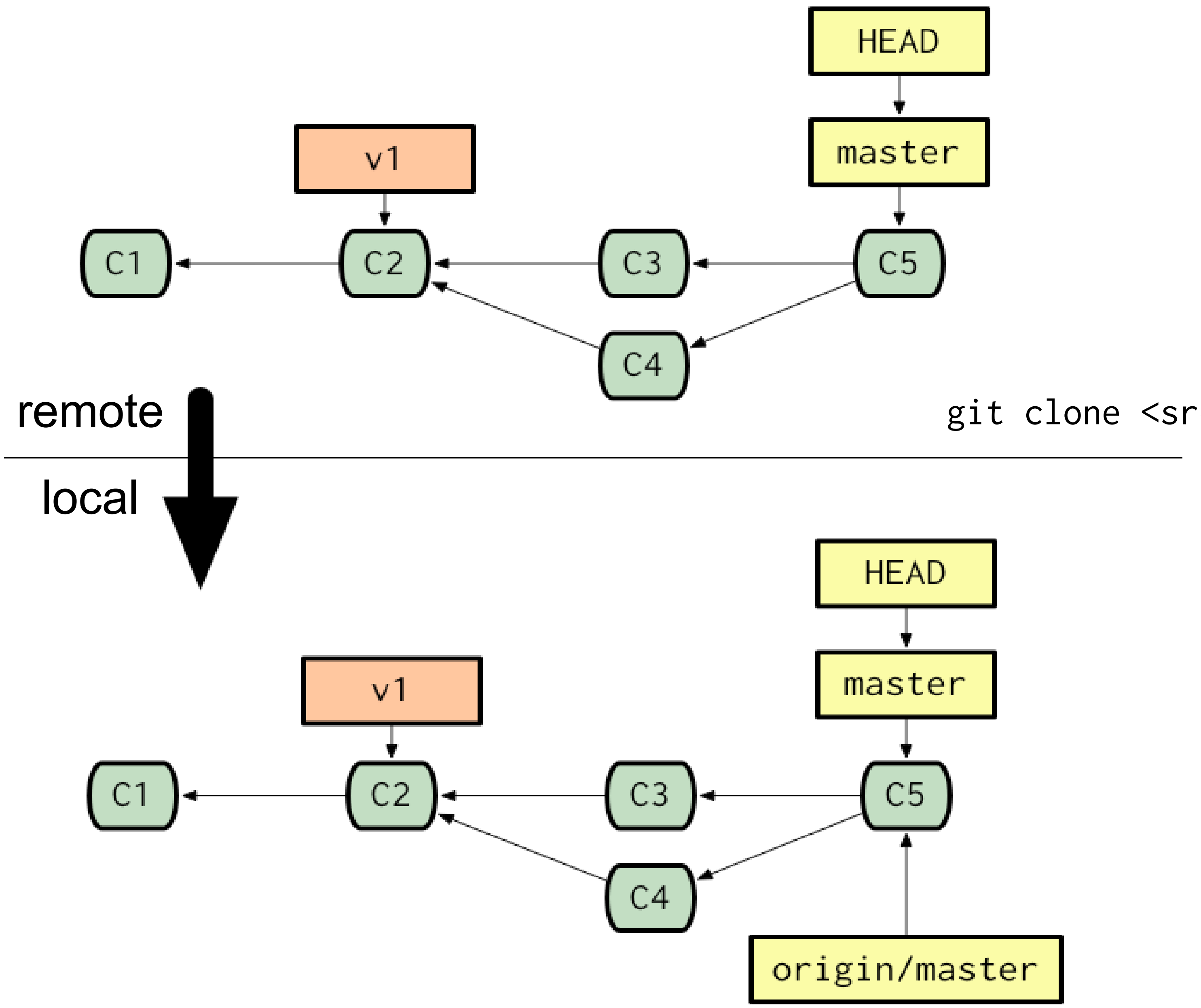
```
echo 4 >> file.txt

git add file.txt

git commit file.txt
```

Merging



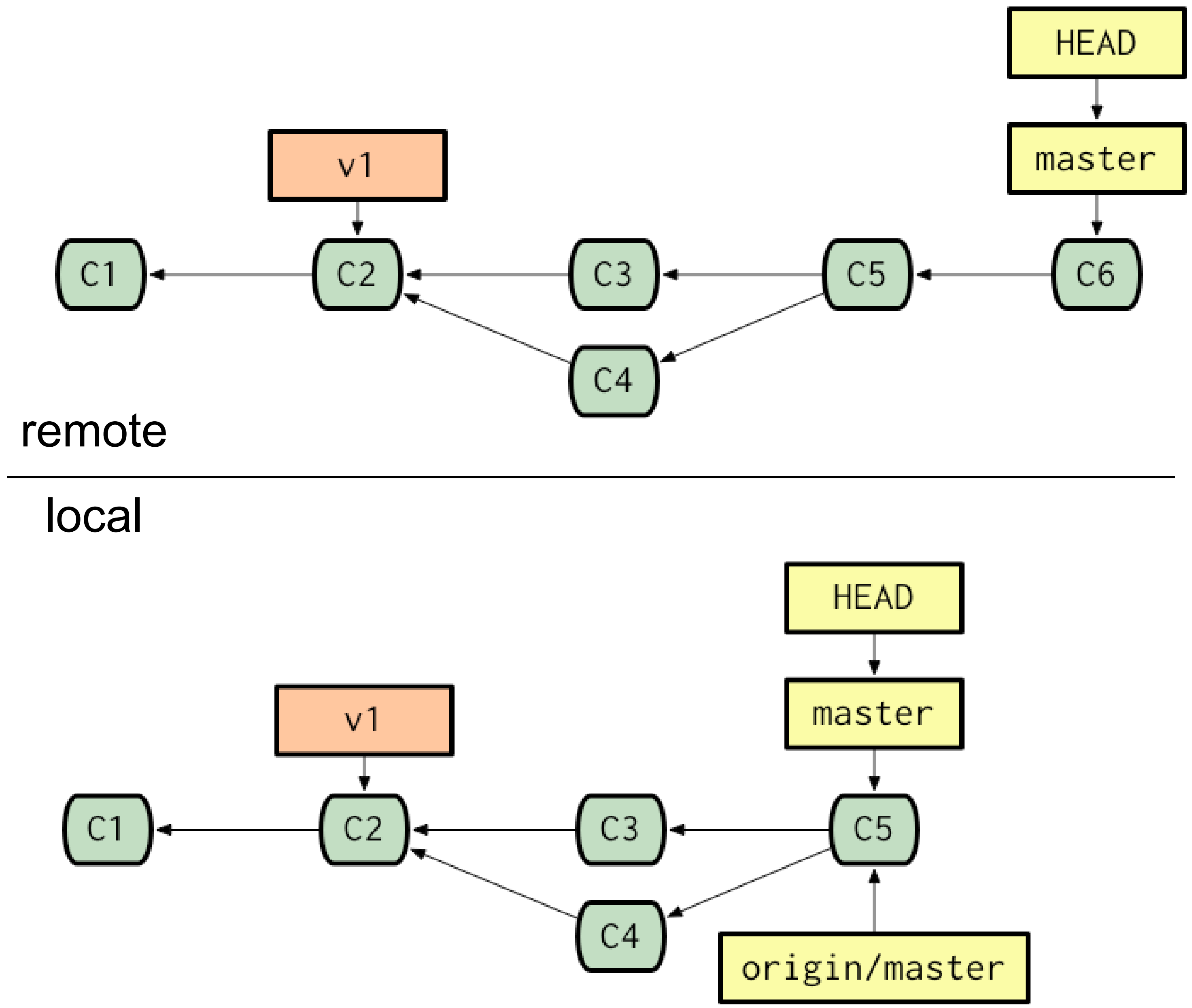
```
git checkout master

git merge fix-v1
```

Cloning a repository

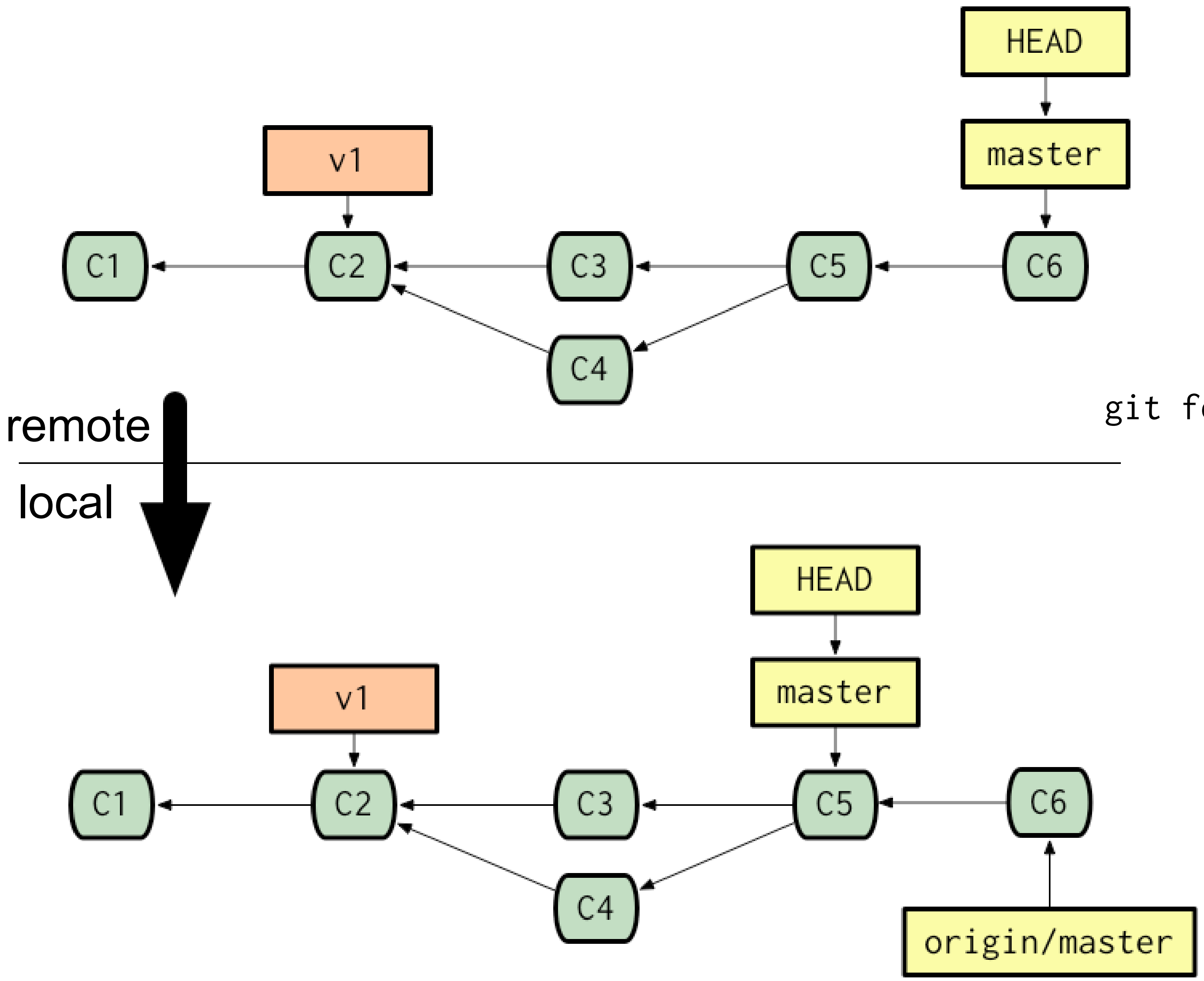


```
git clone <src url> <local path>
```

Changes in origin repository

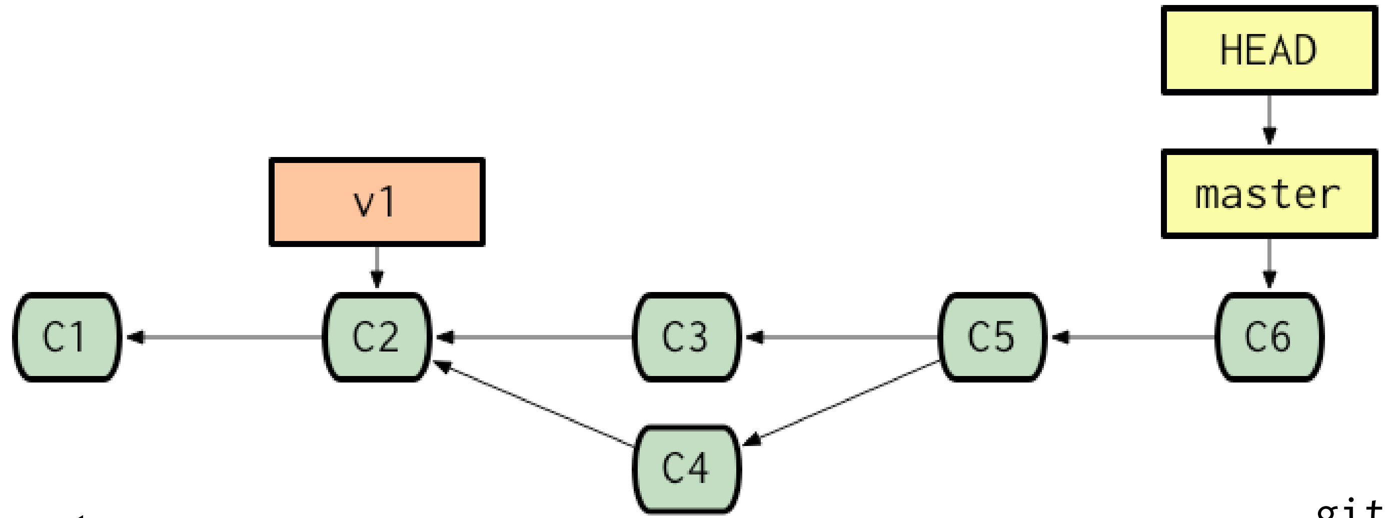


Fetching changes



```
git fetch origin
```

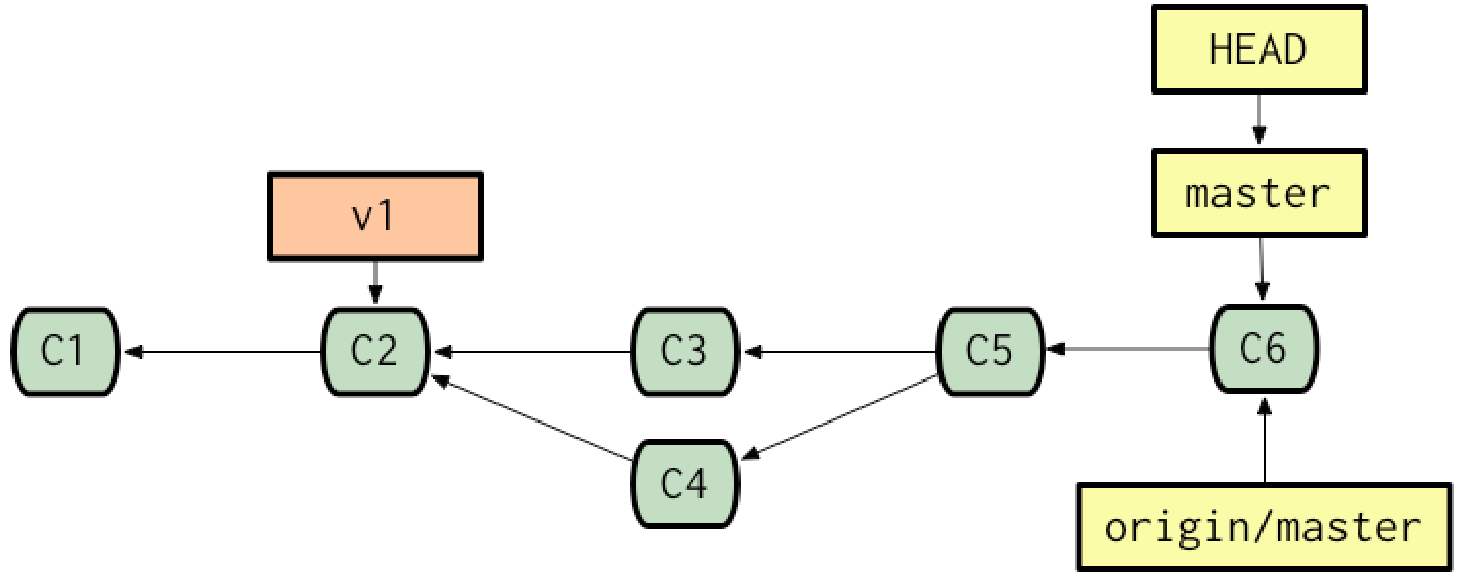
Merging changes



```
git merge origin/master
```

remote

local



Other useful commands

- `git status`: show the working tree status
- `git log`: display commit logs
- `git diff`: display changes between commits
- `git reset`: change the state of the HEAD pointer
- `git clean`: remove untracked files from working directory
- `git rm:` mark files for deletion
- `git stash`: stash uncommitted changes away
- `git pull`: fetch changes from a remote repository and merge into current branch
- `git push`: push local changes to a remote repository
- `git remote`: manage references to remote repositories

Ways to refer to objects

- Full SHA-1: `67a2456566bb217e9d3d0a5e5ed063b978432291`
- Partial SHA-1: `67a245656`
- Branch, tag: `master`, `v1`
- Remote: `origin/master`
- Date: `master@{1 week ago}`
- Ordinal (nth previous value of ref): `master@{8}`
- nth parent: `master^2`
- nth generation grandparent: `master~5`

Ranges

- Everything between two commits

  2e308b51..ca695e2
- Everything since a commit

  2e308b51..
- Remember: any of the ways to refer to an object can be used to bound the ranges

  `master@{1 week ago}..master^`
- Can be used as input to `git log` and `git diff`

Fun stuff

- You've been coding like mad and have committed dozens of changes
  - … but it's been a while since you last ran your unit tests
- Before you push your changes, you decide to run them and … BOOM!
  - Where's Waldo?

Fun stuff

- Options:
  - Analyze the error, stacktraces, etc. Try to deduce what might have caused the problem
  - Run the units tests for every commit since last time the tests succeeded until you find the culprit
    - Binary search?
    - By hand? You might be able to automate it...
    - But wait, there's a git command for this!

```
git bisect
```

git bisect

Demo

Git with SVN

- Git as a better SVN client
- Commands
  - `git svn clone <svn url>`
  - `git svn rebase`
  - `git svn dcommit`
- Caveats
  - Need to “linearize” your commits before pushing them back to SVN

Why Git?

- Fast checkouts
- Fast branching
- Fast merging
- Fast diff
- Fast history browsing
- Did I mention it's fast?
  - It enables a different style of development
    - commit early, commit often
    - separate “commit” from “making changes available to others”
    - branches even for minor changes

Why Git?

- Detached operation
  - Most commands can operate locally
- Flexible workflows
  - Centralized for corporate
  - Tiered for large-scale open source (e.g, linux kernel)
  - Decentralized for small projects and ad-hoc development. E.g.,
    - smaller open source projects
    - synchronizing shell config & scripts across laptop, desktop, etc.

Installing

- Mac OS X with MacPorts

  ```
  port install git-core +bash_completion+doc+svn
  ```

- Ubuntu/Debian

  ```
  apt-get install git-core
  apt-get install git-svn
  apt-get install git-completion
  ```

- Windows

  http://code.google.com/p/msysgit/

- Other useful tools
  - GitX, gitk, qgit, gitg, tig

Resources

- Official Git website

  http://git-scm.com/
- Git User's Manual

  http://www.kernel.org/pub/software/scm/git/docs/user-manual.html
- Pro Git book, by Scott Chacon

  http://progit.org/book/
- Git reference site

  http://gitref.org/
- Git tips & tricks

  http://www.gitready.com/
- Man pages

  git help [<command>]