Pro Git

Grasping it Conceptually

by Kim Seungwon
(seungzzang@gmail.com)

http://git-scm.com/

Grasping Git Conceptually - Part 1

Grasping Git Conceptually - Part 2

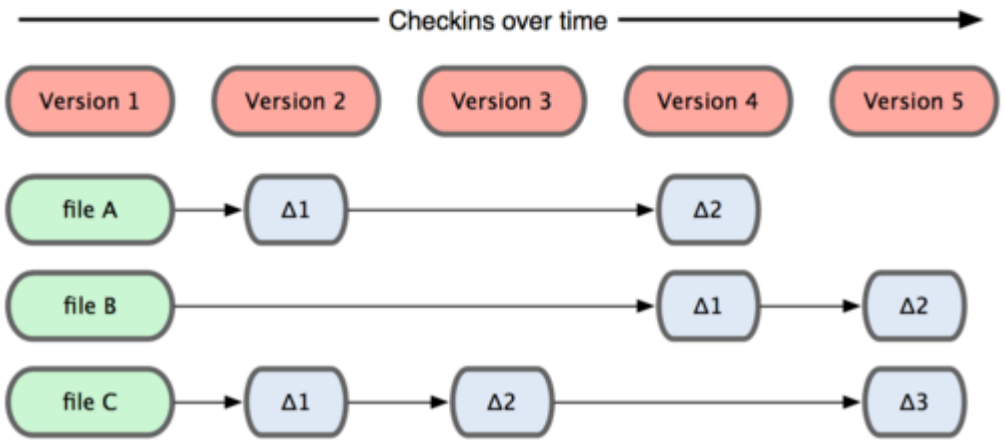
Git Basics - The Way Git Stores History

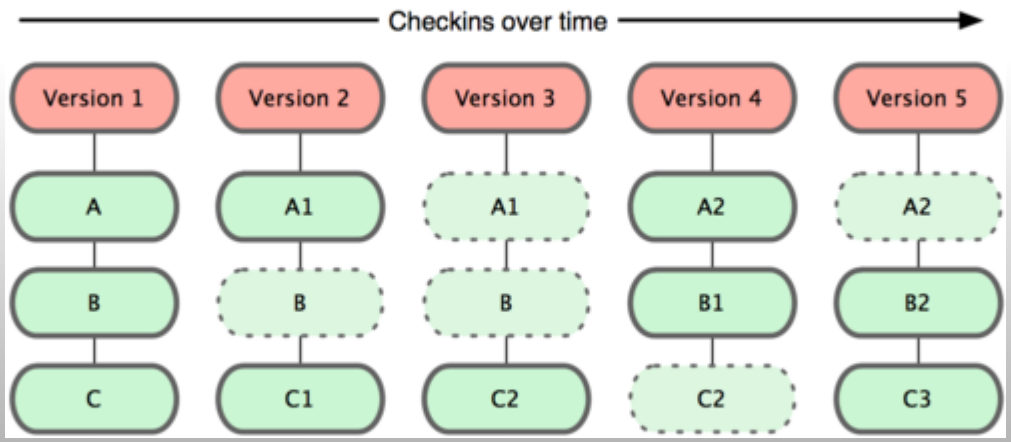
Git initial settings

- /etc/gitconfig, --system
- ~/.gitconfig, --global
- .git/config, per repository

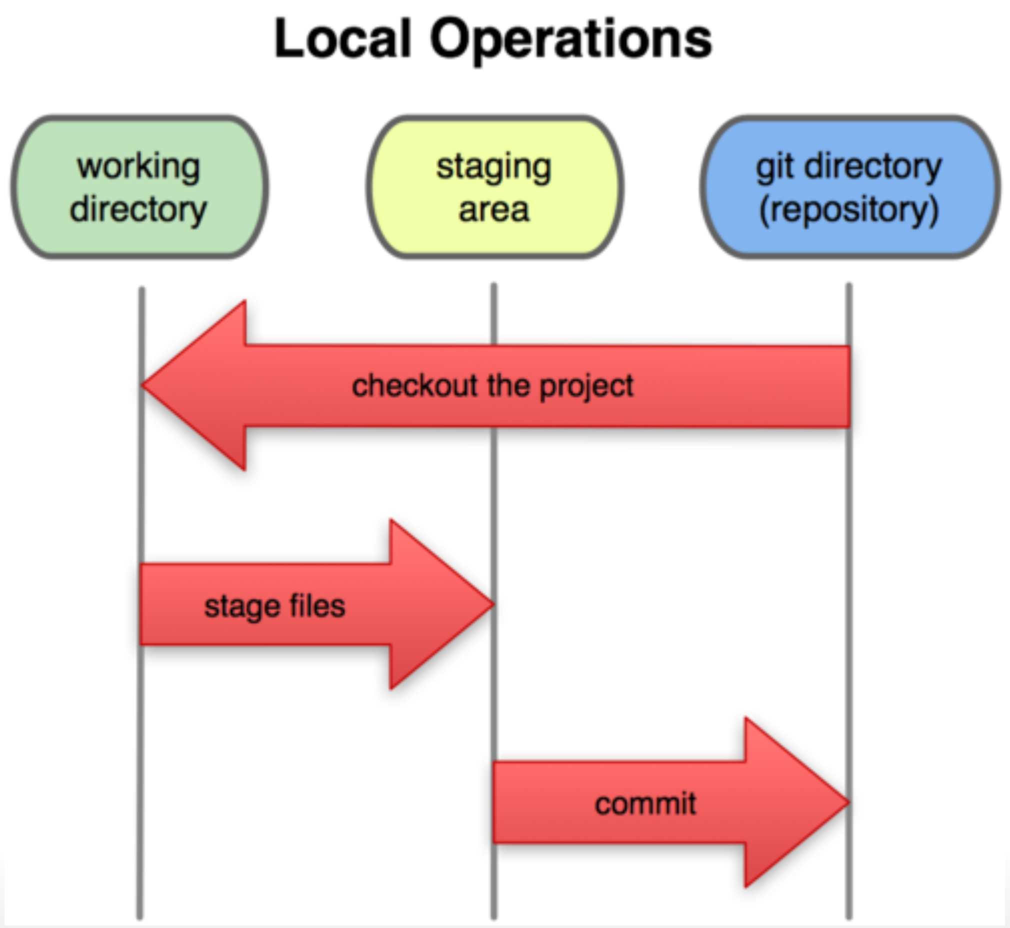
```
$ git config --global user.name "seungzzang"
$ git config --global user.email seungzzang@gmail.com
```

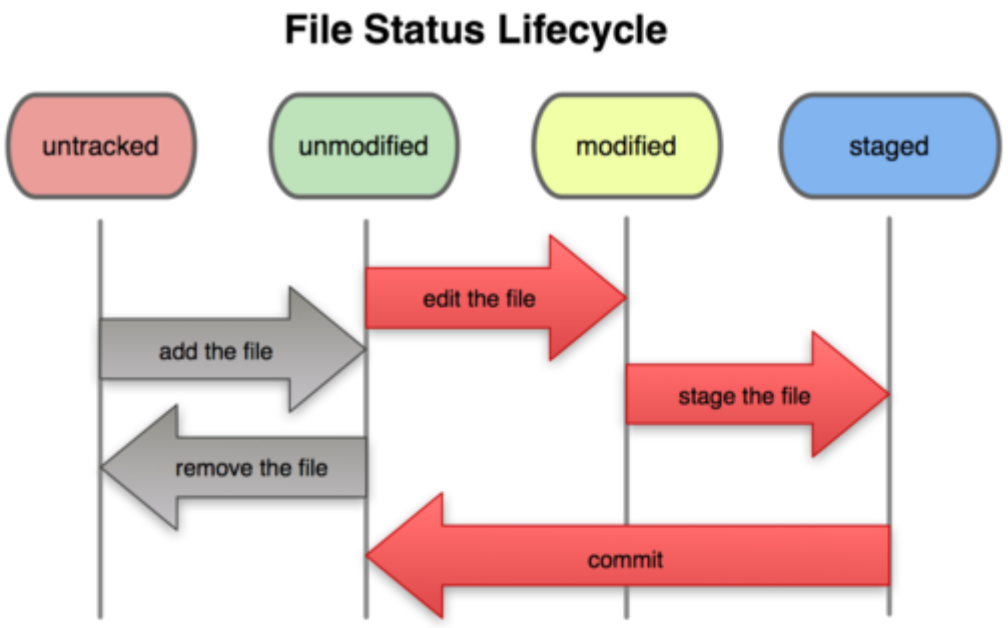
Thw Way Git Stores History : Snapshots, Not Differences





Git Basics - File Status / Committed, Modified and Staged





```
$ vim benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   README
#   modified:   benchmarks.rb
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#   modified:   benchmarks.rb
#
```

Git Basics - Ignoring Files

Ignoring Files - .gitignore

- .gitignore

```
# a comment - this is ignored
# no .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the root TODO file, not subdir/TODO
/TODO
# ignore all files in the build/ directory
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .txt files in the doc/ directory
doc/**/*.txt
```

: Git since version 1.8.2.

Viewing your **staged and unstaged changes**

- git diff, **git diff --staged**

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#   modified:   benchmarks.rb
#
$ git diff
diff --git a/benchmarks.rb b/benchmarks.rb ...

$ git diff --cached
diff --git a/README b/README ...
```

Git Basics - Recording Changes to the Repo.

Committing changes

- `git commit`
- `git commit -v` (put the diff in the editor)
- `git commit -m '...'`

Skipping the Staging Area

- `git commit -a -m 'Add new benchmarks'`

Removing Files

- rm : Changes not staged for commit
- git rm : Changes to be commit
- `git rm --cached readme.txt` (remove it from staging area, don't track any more)

Moving Files

- `git mv file_from file_to`

```
$ git mv README.txt README
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    README.txt -> README
#
```

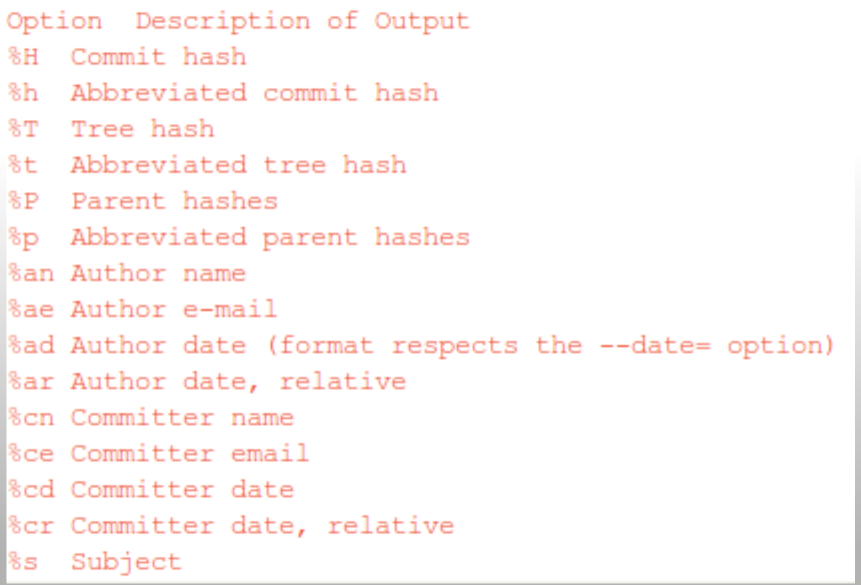
```
$ mv README.txt README
$ git rm README.txt
$ git add README
```

Git Basics - Viewing Commit History

```
git log
     git log -p -2
     git log --stat
     git log --pretty=oneline
     git log --pretty=format:"%h - %an, %ar : %s"
     git log --pretty=format:"%h %s" --graph
     git log --pretty="%h - %s" --author=gitster --since="2008-10-01" --before="2008-11-01" --no-merges -- t/
```

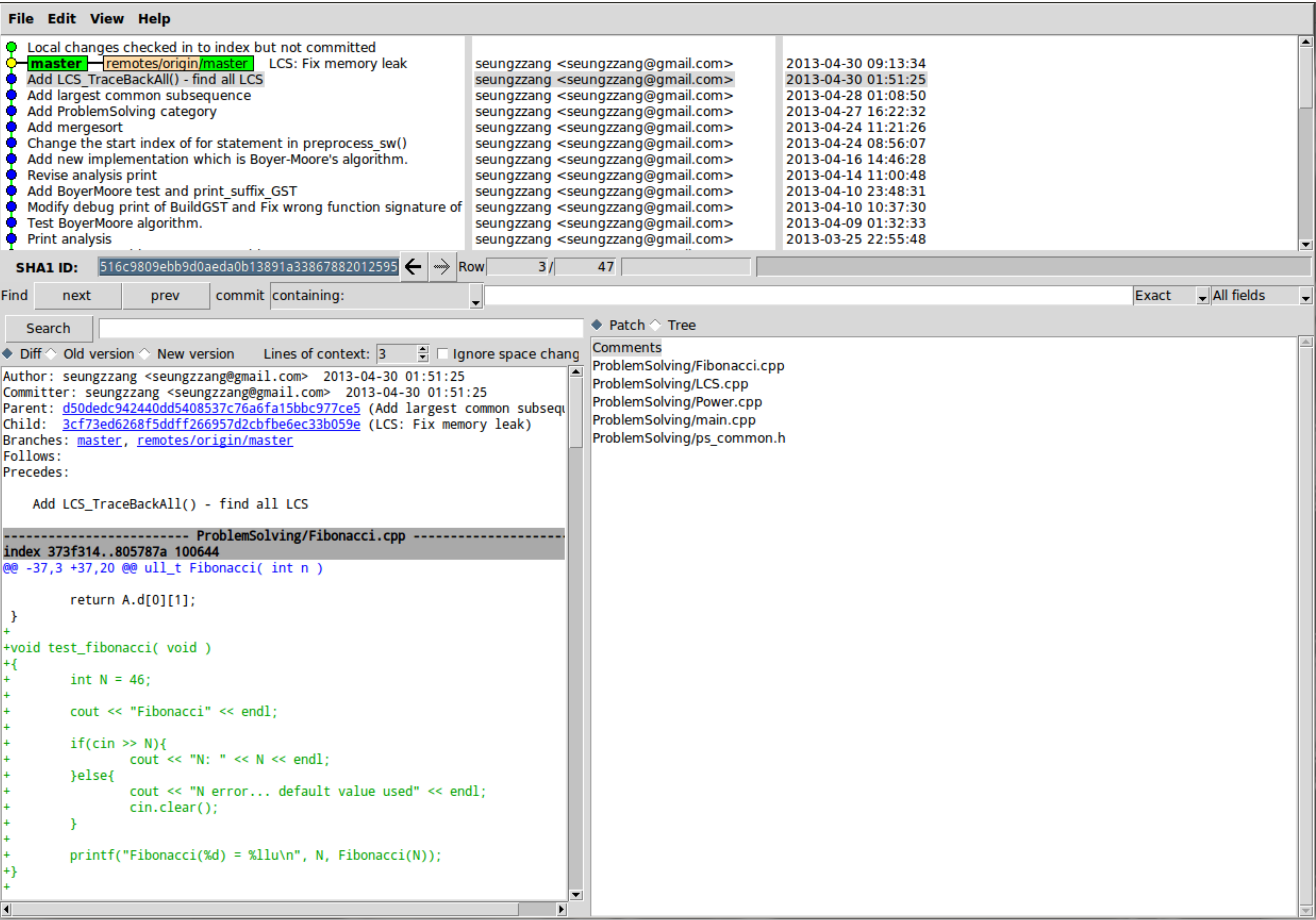
-p	Show the patch introduced with each commit.
--word-diff	Show the patch in a word diff format.
--stat	Show statistics for files modified in each commit.
--shortstat	Display only the changed/insertions/deletions line from the --stat command.
--name-only	Show the list of files modified after the commit information.
--name-status	Show the list of files affected with added/modified/deleted information as well.
--abbrev-commit	Show only the first few characters of the SHA-1 checksum instead of all 40.
--relative-date	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
--graph	Display an ASCII graph of the branch and merge history beside the log output.
--pretty	Show commits in an alternate format. Options include oneline, short, full, fuller, and format (where you specify your own format).
--oneline	A convenience option short for `--pretty=oneline --abbrev-commit`

```
     git log -7 --pretty=format:"%h %s" --graph
     git log -7 --pretty=oneline --abbrev-commit --graph
     git log -7 --oneline --graph
```



Option	Description of Output
%H	Commit hash
%h	Abbreviated commit hash
%T	Tree hash
%t	Abbreviated tree hash
%P	Parent hashes
%p	Abbreviated parent hashes
%an	Author name
%ae	Author e-mail
%ad	Author date (format respects the --date= option)
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cd	Committer date
%cr	Committer date, relative
%s	Subject

Viewing Commit History - gitk



File Edit View Help
SHA1 ID:
Row 3 / 47
Find next prev commit containing:
Exact All fields
Search
Patch Tree
Diff Old version New version Lines of context: 3 Ignore space chang

```
Local changes checked in to index but not committed
master  remotes/origin/master  LCS: Fix memory leak          seungzzang <seungzzang@gmail.com>  2013-04-30 09:13:34
Add LCS_TraceBackAll() - find all LCS                        seungzzang <seungzzang@gmail.com>  2013-04-30 01:51:25
Add largest common subsequence                               seungzzang <seungzzang@gmail.com>  2013-04-28 01:08:50
Add ProblemSolving category                                  seungzzang <seungzzang@gmail.com>  2013-04-27 16:22:32
Add mergesort                                                seungzzang <seungzzang@gmail.com>  2013-04-24 11:21:26
Change the start index of for statement in preprocess_sw()   seungzzang <seungzzang@gmail.com>  2013-04-24 08:56:07
Add new implementation which is Boyer-Moore's algorithm.     seungzzang <seungzzang@gmail.com>  2013-04-16 14:46:28
Revise analysis print                                        seungzzang <seungzzang@gmail.com>  2013-04-14 11:00:48
Add BoyerMoore test and print_suffix_GST                     seungzzang <seungzzang@gmail.com>  2013-04-10 23:48:31
Modify debug print of BuildGST and Fix wrong function signature of  seungzzang <seungzzang@gmail.com>  2013-04-10 10:37:30
Test BoyerMoore algorithm.                                   seungzzang <seungzzang@gmail.com>  2013-04-09 01:32:33
Print analysis                                               seungzzang <seungzzang@gmail.com>  2013-03-25 22:55:48
```

SHA1 ID: 516c9809ebb9d0aeda0b13891a33867882012595

```
Author: seungzzang <seungzzang@gmail.com>  2013-04-30 01:51:25
Committer: seungzzang <seungzzang@gmail.com>  2013-04-30 01:51:25
Parent: d50dedc942440dd5408537c76a6fa15bbc977ce5 (Add largest common subsequ
Child:  3cf73ed6268f5ddff266957d2cbfbe6ec33b059e (LCS: Fix memory leak)
Branches: master, remotes/origin/master
Follows:
Precedes:

    Add LCS_TraceBackAll() - find all LCS

------------------------- ProblemSolving/Fibonacci.cpp --------------------
index 373f314..805787a 100644
@@ -37,3 +37,20 @@ ull_t Fibonacci( int n )

        return A.d[0][1];
 }
+
+void test_fibonacci( void )
+{
+       int N = 46;
+
+       cout << "Fibonacci" << endl;
+
+       if(cin >> N){
+               cout << "N: " << N << endl;
+       }else{
+               cout << "N error... default value used" << endl;
+               cin.clear();
+       }
+
+       printf("Fibonacci(%d) = %llu\n", N, Fibonacci(N));
+}
+
```

```
Comments
ProblemSolving/Fibonacci.cpp
ProblemSolving/LCS.cpp
ProblemSolving/Power.cpp
ProblemSolving/main.cpp
ProblemSolving/ps_common.h
```

Git Basics - Basic Undoing and Remote Repo Commands

Changes the last commit

- `git commit -m '...'`
- `git add forgotten_file`
- `git commit --amend`

Unstaging a staged file

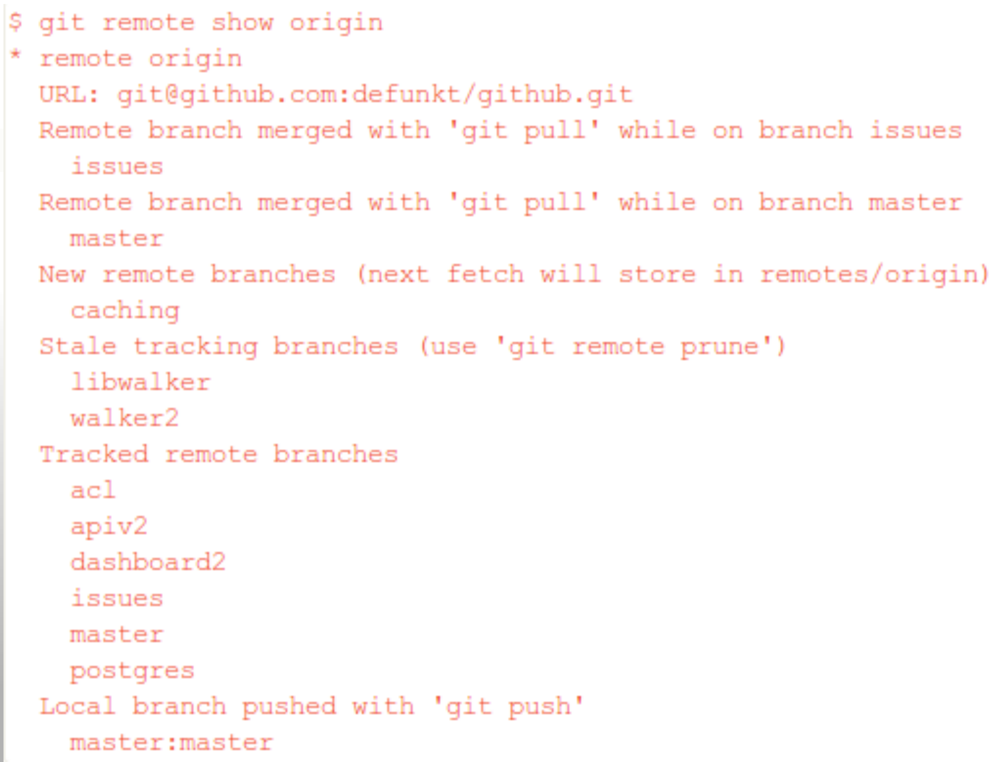
- `git reset HEAD <file>`
- `git config --global alias.unstage 'reset HEAD'`
- `git unstage <file>`

Unmodifying a file

- `git checkout -- <file>`

Remote repository

- `git remote -v`
- `git remote add <shortname> <url>`
- `git fetch <remote-name>`
- `git pull`
- `git push <remote-name> <branch-name>`
- `git remote show <remote-name>`
- `git remote rename pb paul`
- `git remote rm paul`

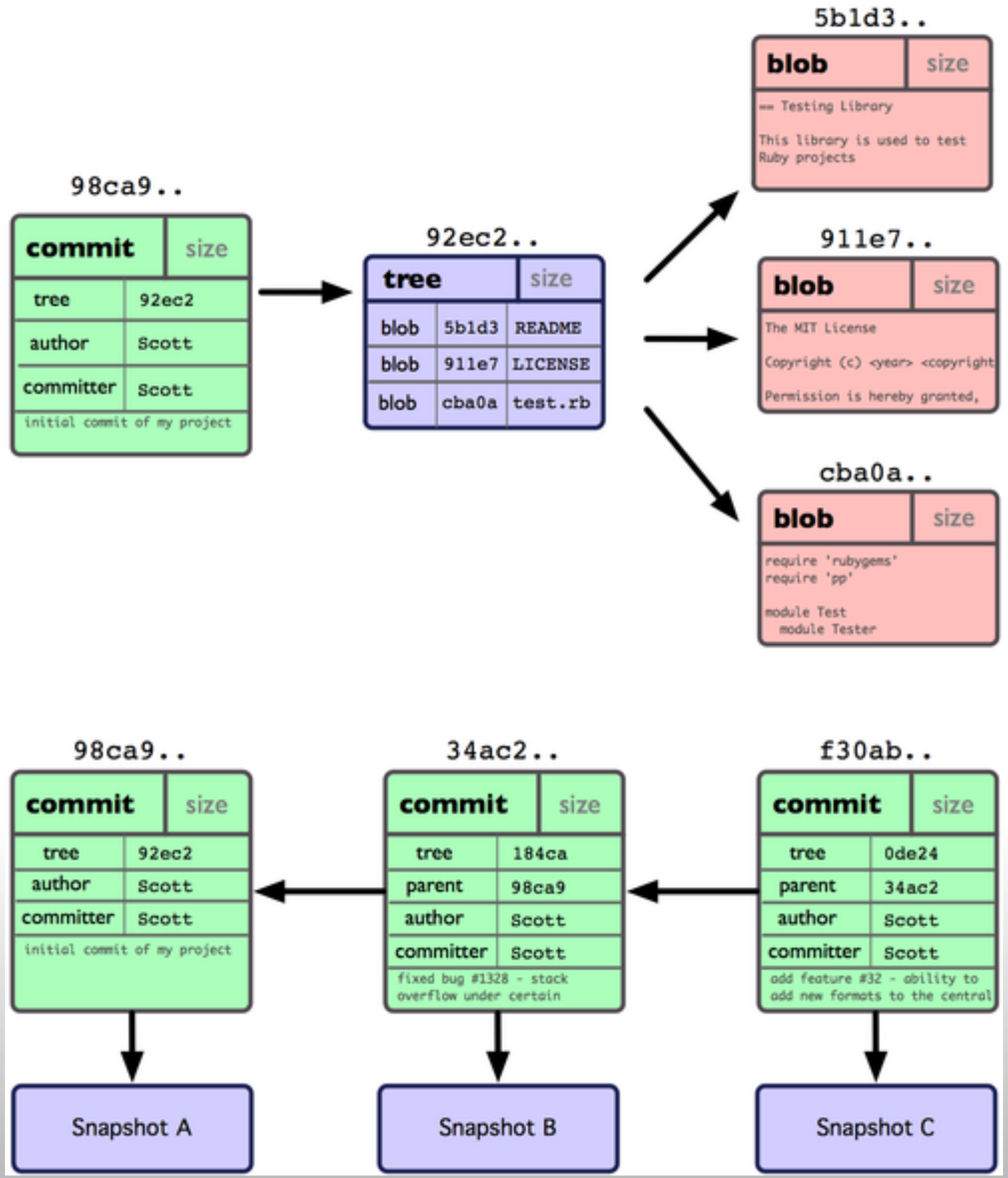


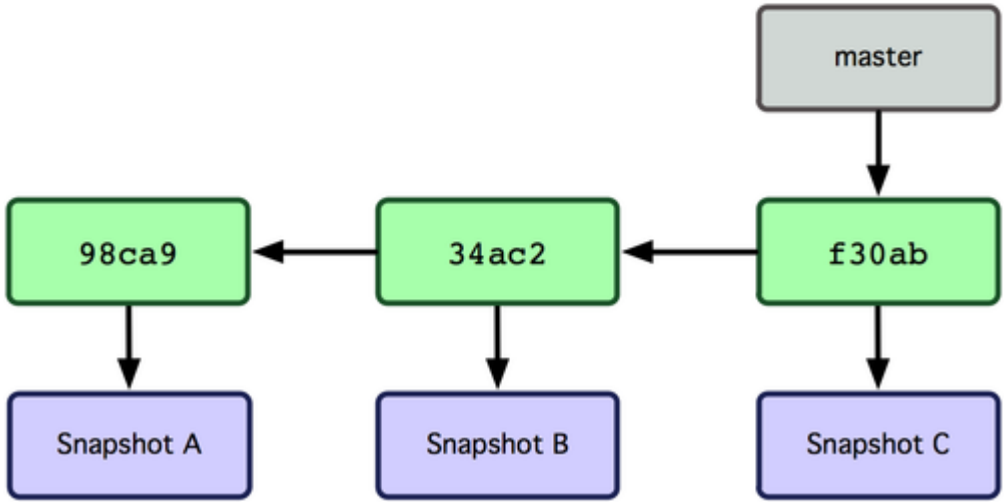
```
$ git remote show origin
* remote origin
  URL: git@github.com:defunkt/github.git
  Remote branch merged with 'git pull' while on branch issues
    issues
  Remote branch merged with 'git pull' while on branch master
    master
  New remote branches (next fetch will store in remotes/origin)
    caching
  Stale tracking branches (use 'git remote prune')
    libwalker
    walker2
  Tracked remote branches
    acl
    apiv2
    dashboard2
    issues
    master
    postgres
  Local branch pushed with 'git push'
    master:master
```

Git Branching - What a Branch Is

Branch is a pointer.

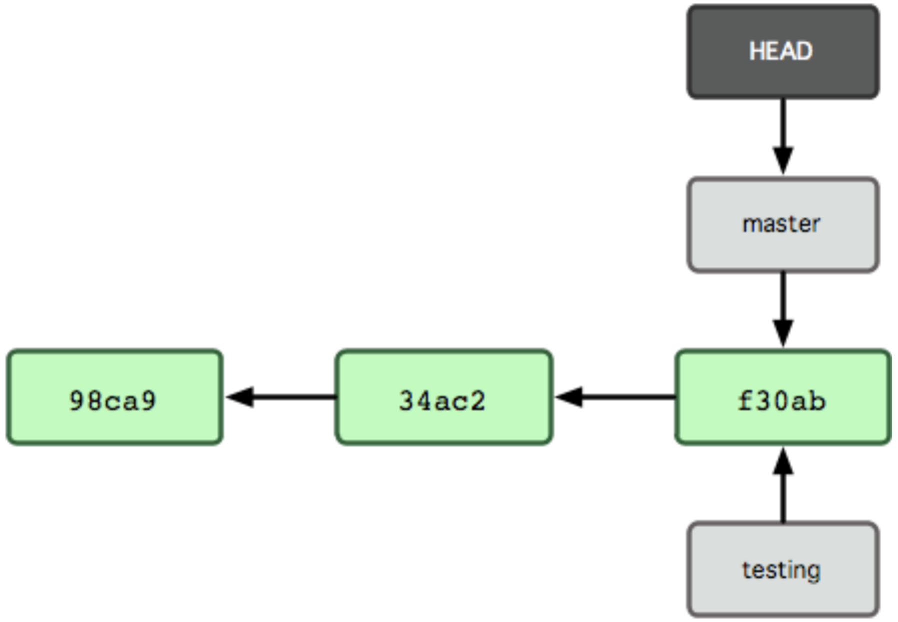
- Simple file that contains the SHA-1 checksum of the commit it points to



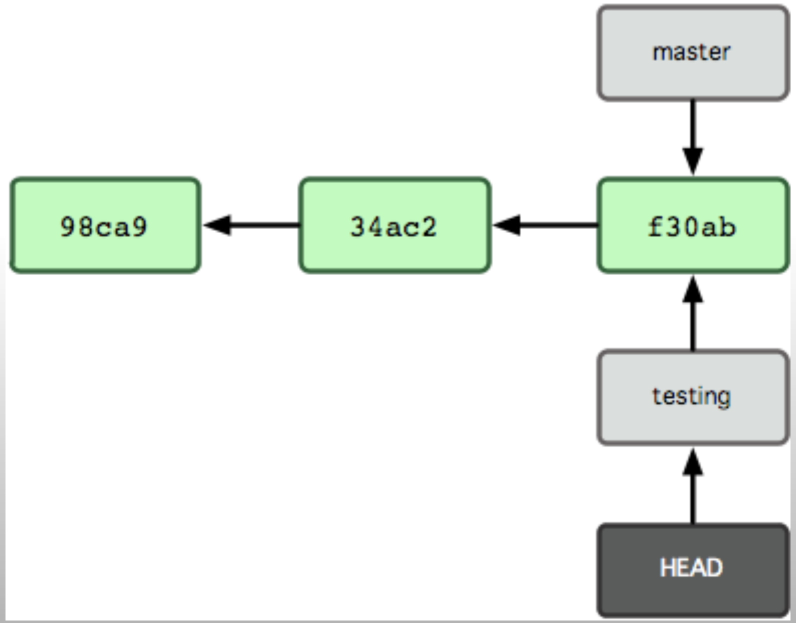


Git Branching - What a Branch Is

- `git branch testing`

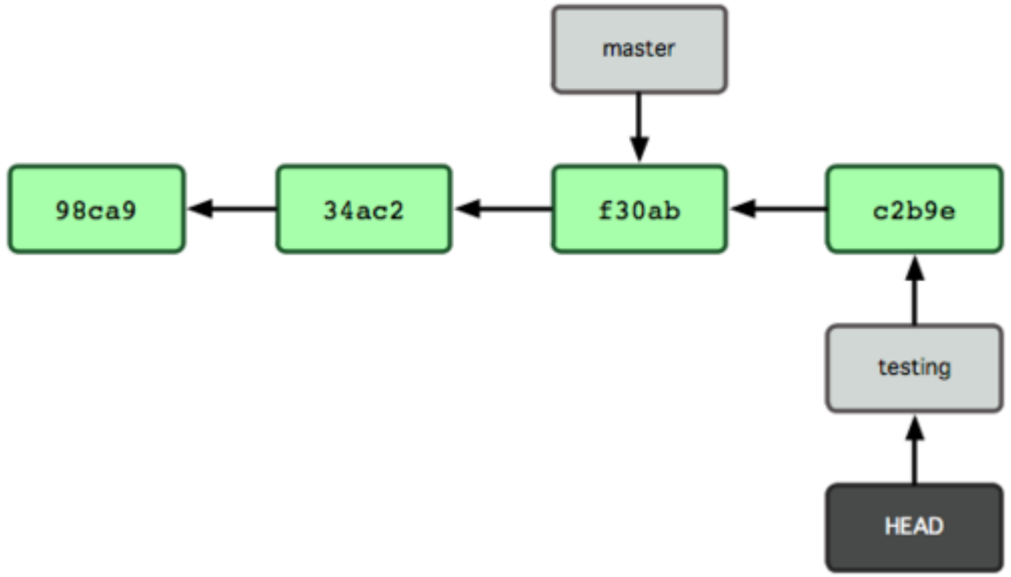


- `git chechout testing`

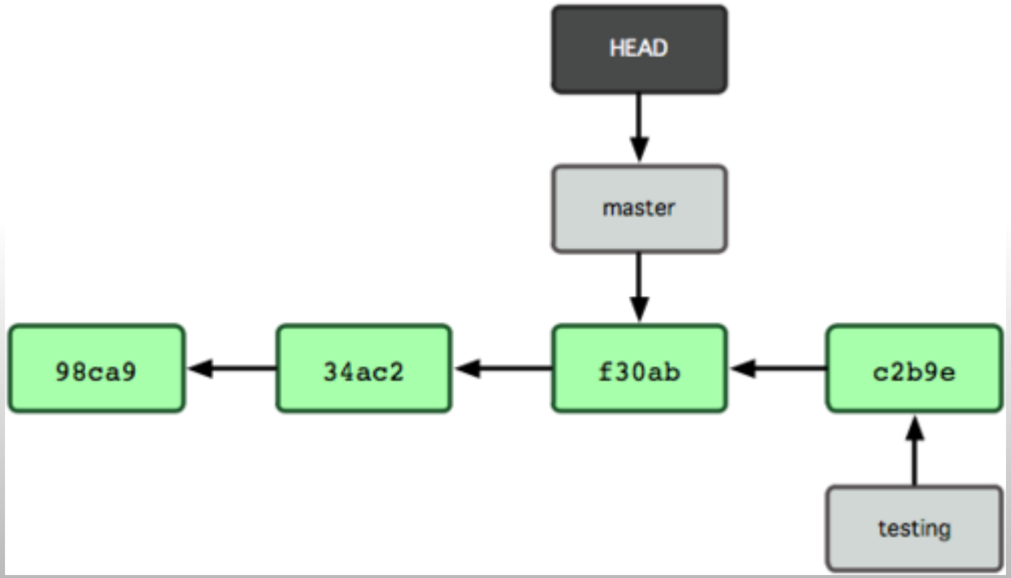


Git Branching - What a Branch Is

- `git commit -a -m 'a change'`

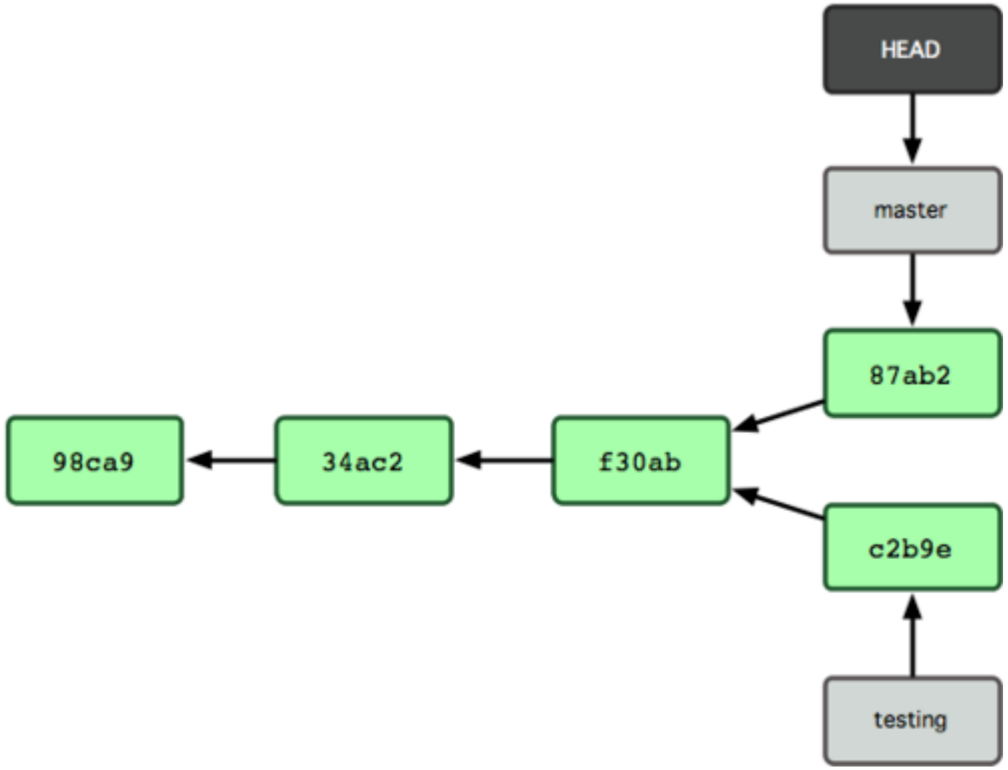


- `git chechout master`



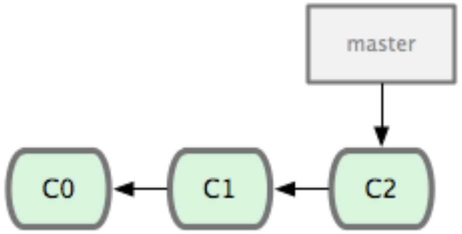
Git Branching - What a Branch Is

- `git commit -a -m 'made another change'`

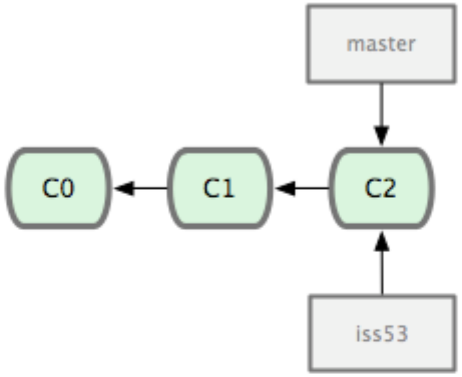


Git Branching - Basic Branching and Merging

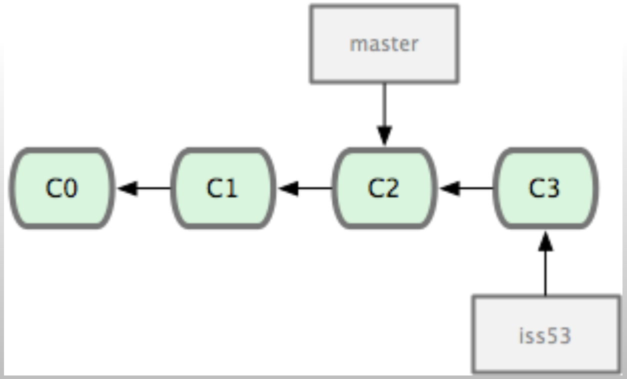
- `Current state`



- `git chechout -b iss53`

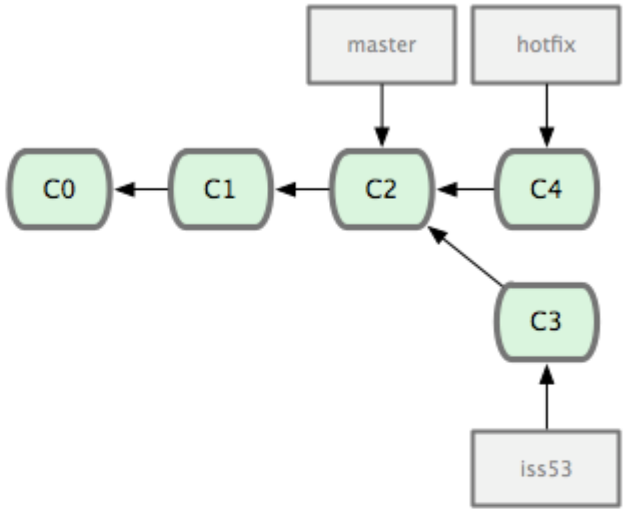


- `git commit -a -m 'new footer [issue 53]'`

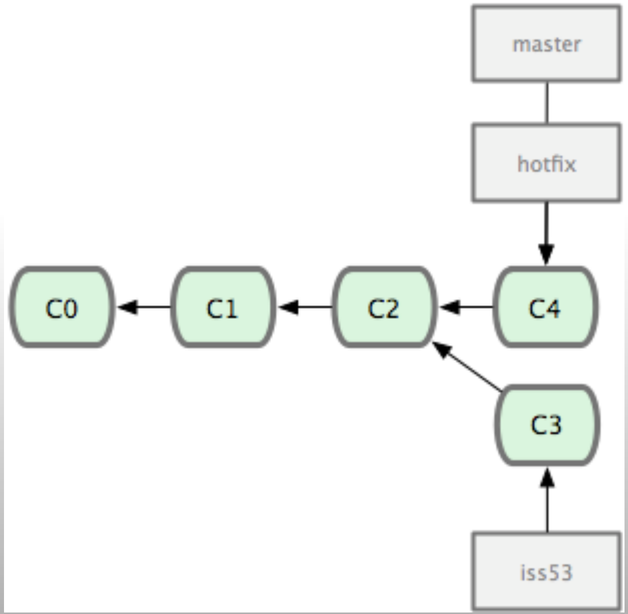


Git Branching - Basic Branching and Merging

- `git checkout master`
- `git checkout -b 'hotfix'`
- `git commit -a -m 'fixed broken email'`

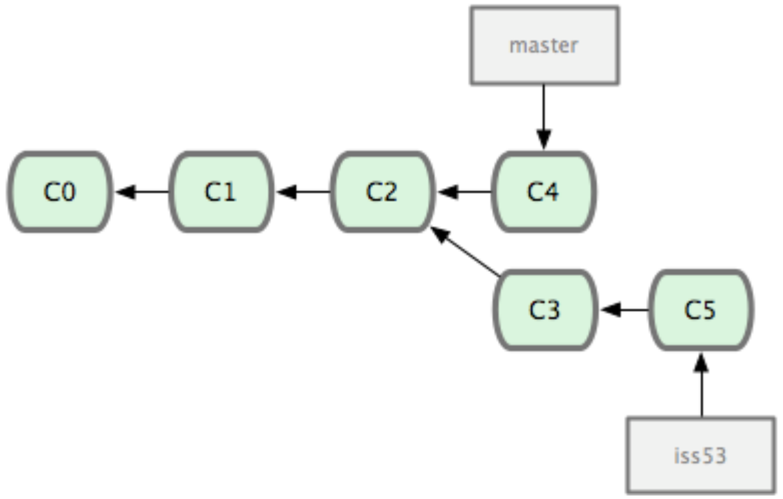


- `git checkout master`
- `git merge hotfix --> fast forward`

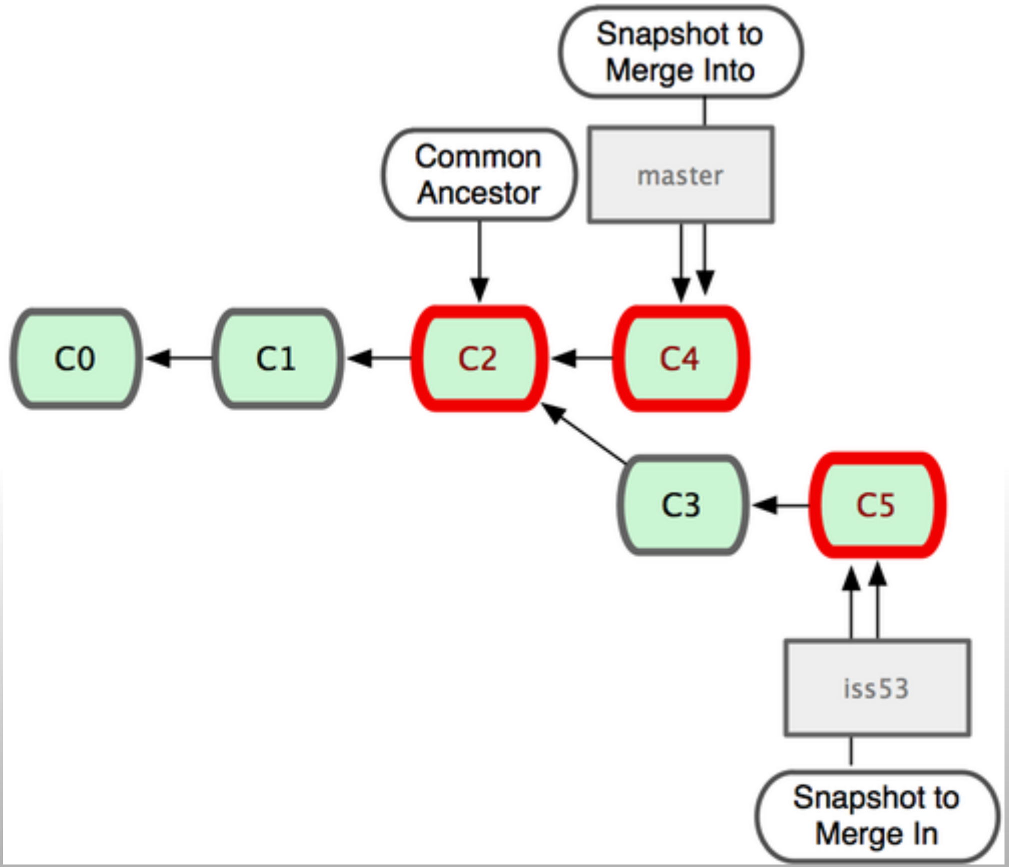


Git Branching - Basic Branching and Merging

- `git branch -d hotfix`
- `git checkout iss53`
- `git commit -a -m 'finished new footer [issue 53]'`

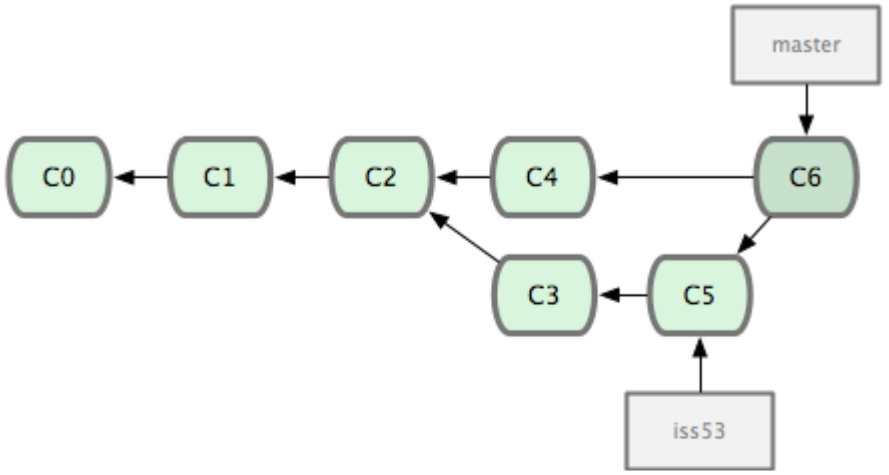


- `git checkout master`
- `git merge iss53`



Git Branching - Basic Branching and Merging

- `git checkout master`
- `git merge iss53`



Git determines the best common ancestor to use for its merge base.

- `git branch -d iss53`

Git Branching - Basic Branching and Merging

- `git merge iss53`

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

$ git status
index.html: needs merge
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   unmerged:   index.html
#
```

```
vi index.html
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

- `git add index.html`
- `git commit`

```
Merge branch 'iss53'

Conflicts:
  index.html
#
# It looks like you may be committing a MERGE.
# If this is not correct, please remove the file
# .git/MERGE_HEAD
# and try again.
#
```

Git Branching - Branch Commands

- `git branch`

```
  iss53
* master
  testing
```

- `git branch -v`

```
  iss53       93b412c  fix javascript issue
* master      7a98805  Merge branch 'iss53'
  testing     782fd34  add scott to the author list in the readmes
```

- `git branch --merged`

```
iss53                        : can be deleted
* master
```

- `git branch --no-merged`

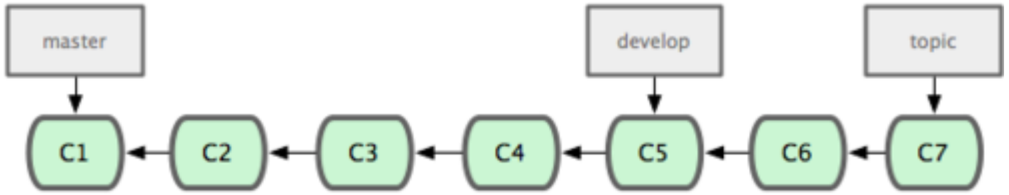
```
testing                      : can not be deleted
```

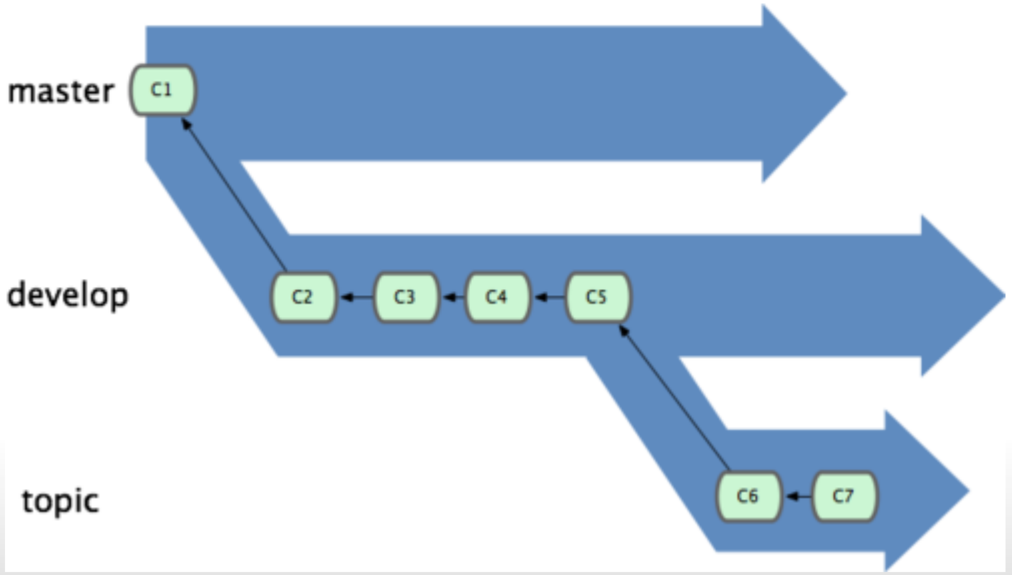
```
$ git branch -d testing
error: The branch 'testing' is not an ancestor of your current HEAD.
If you are sure you want to delete it, run 'git branch -D testing'.
```

You can't delete the branch which is not referenced by any other branches.

Git Branching - Branching Workflows

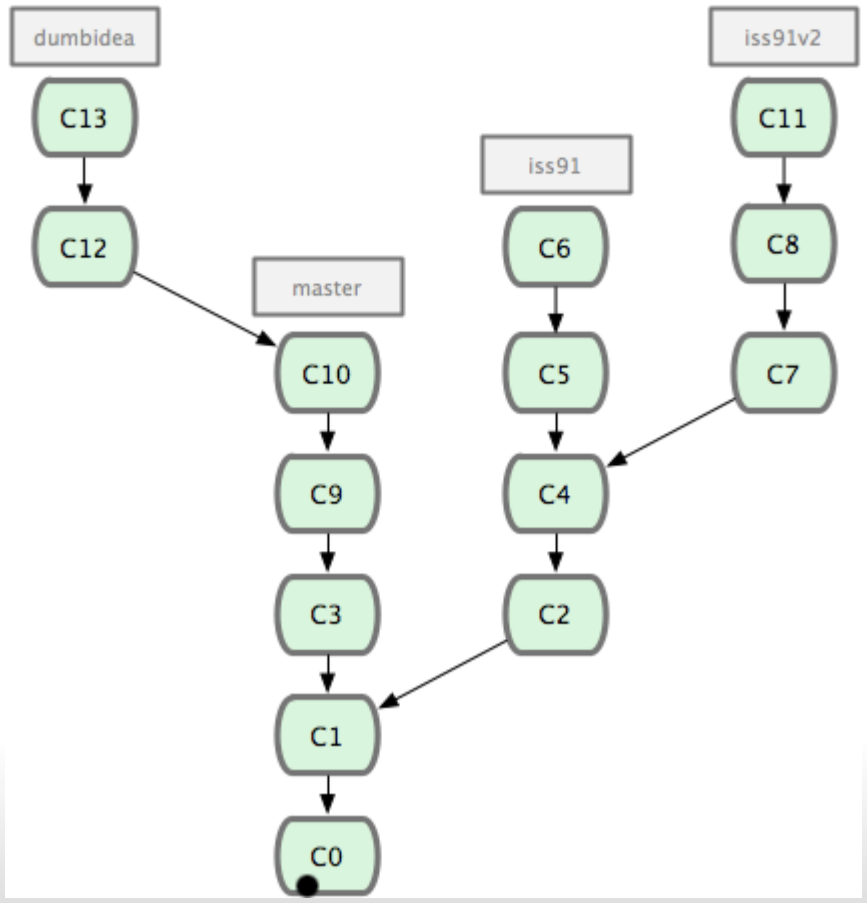
- `Long-Running Branches`

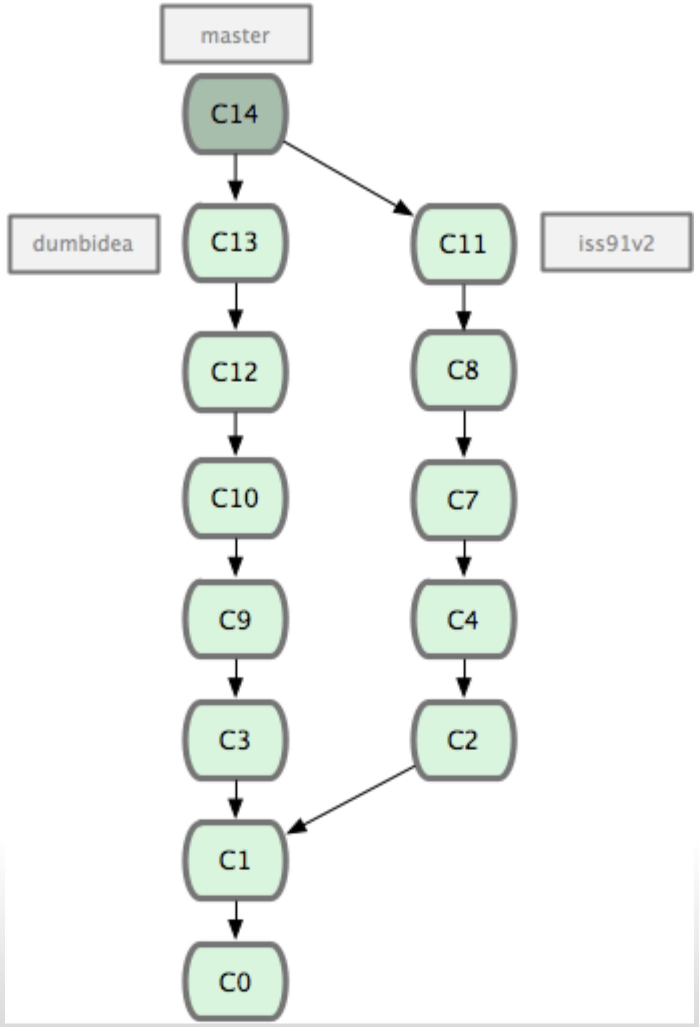




Git Branching - Branching Workflows

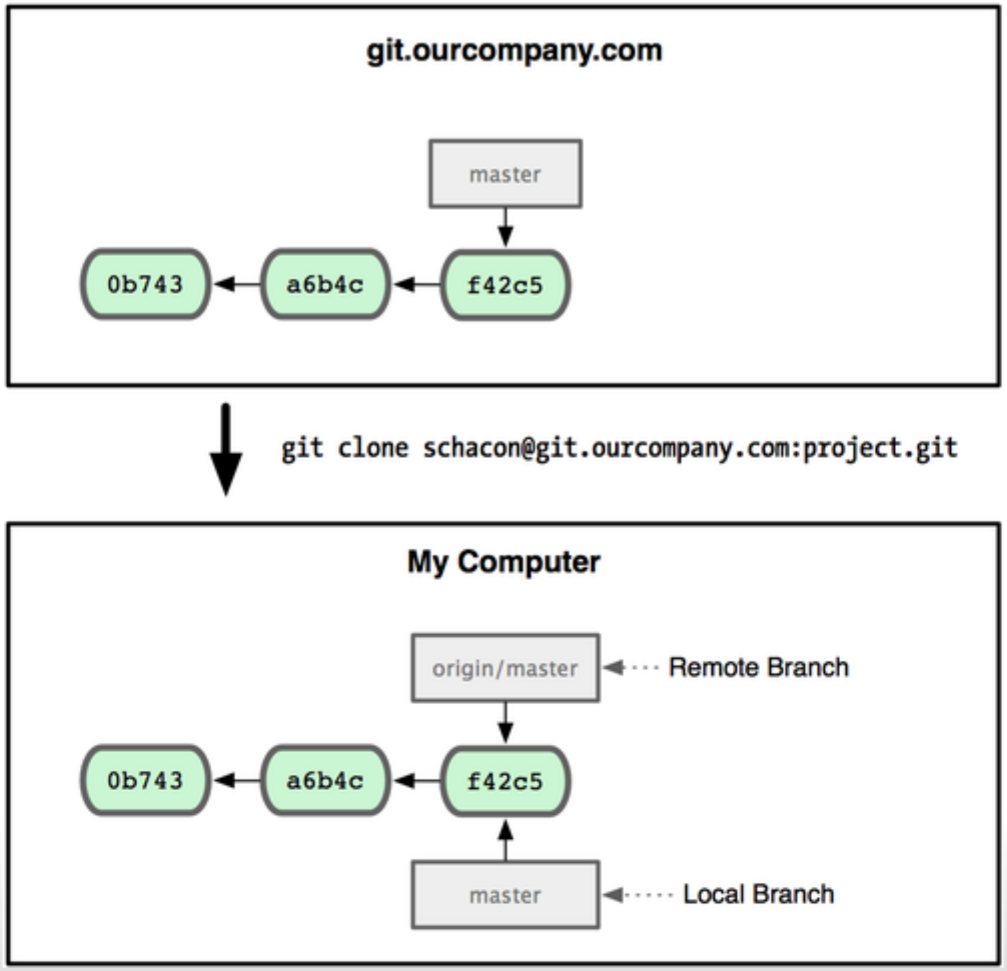
- Topic Branches





Git Branching - Remote Branch

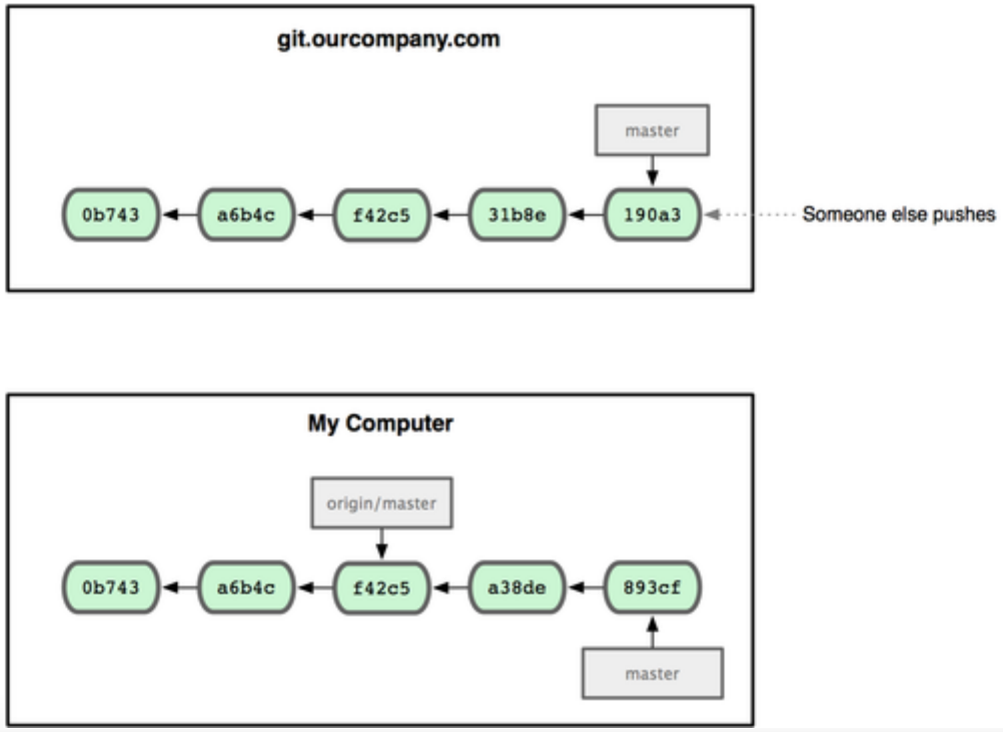
- `git clone`

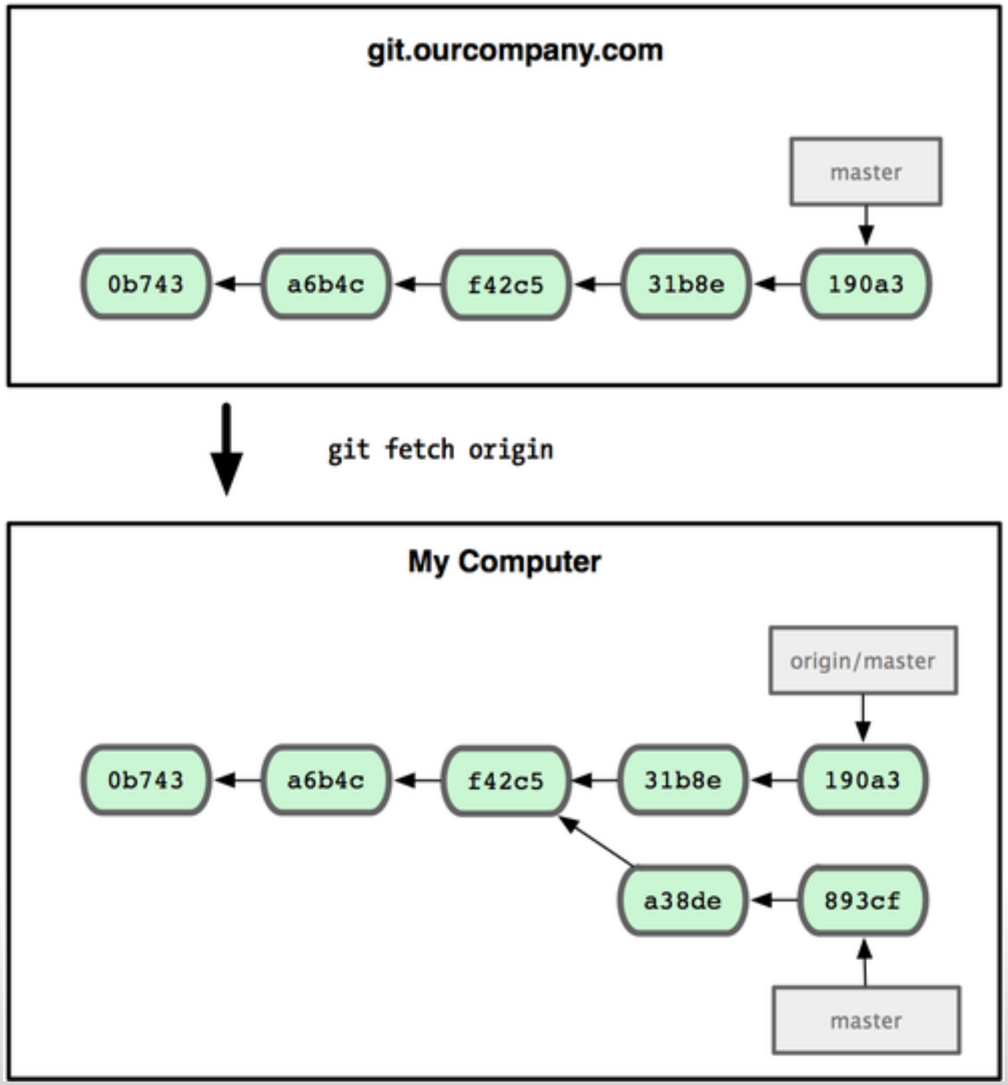


```
git clone schacon@git.ourcompany.com:project.git
```

Git Branching - Remote Branch

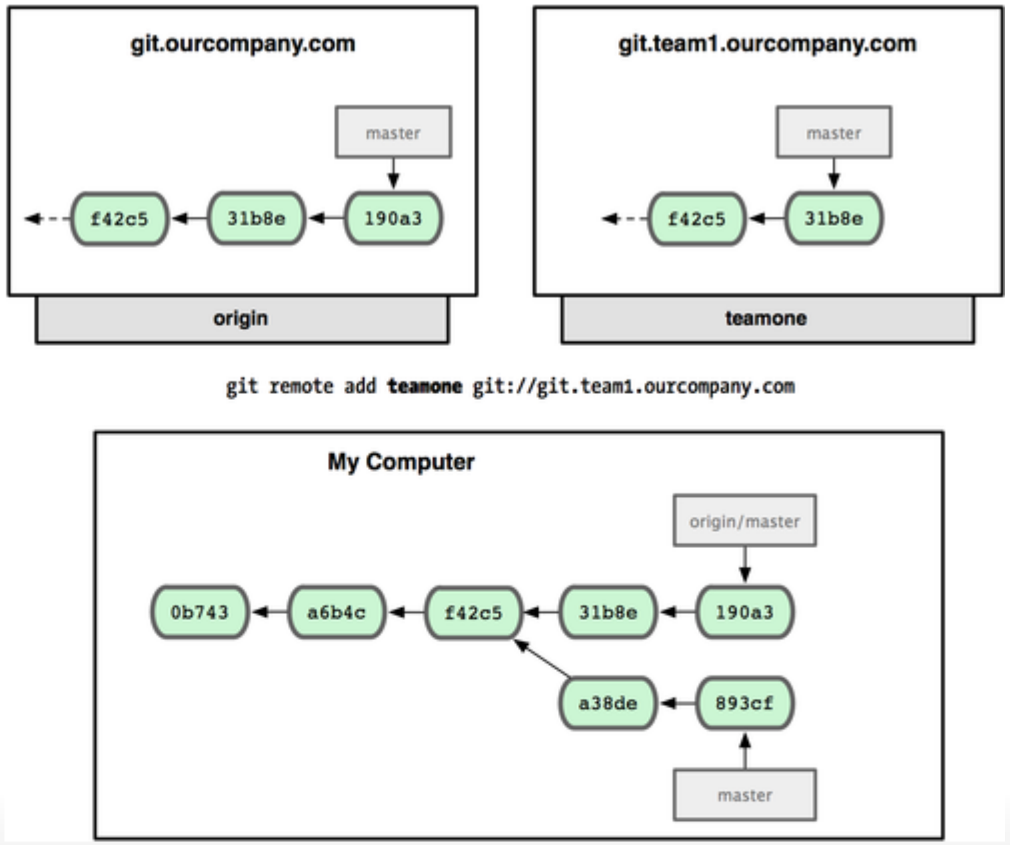
- Work with server

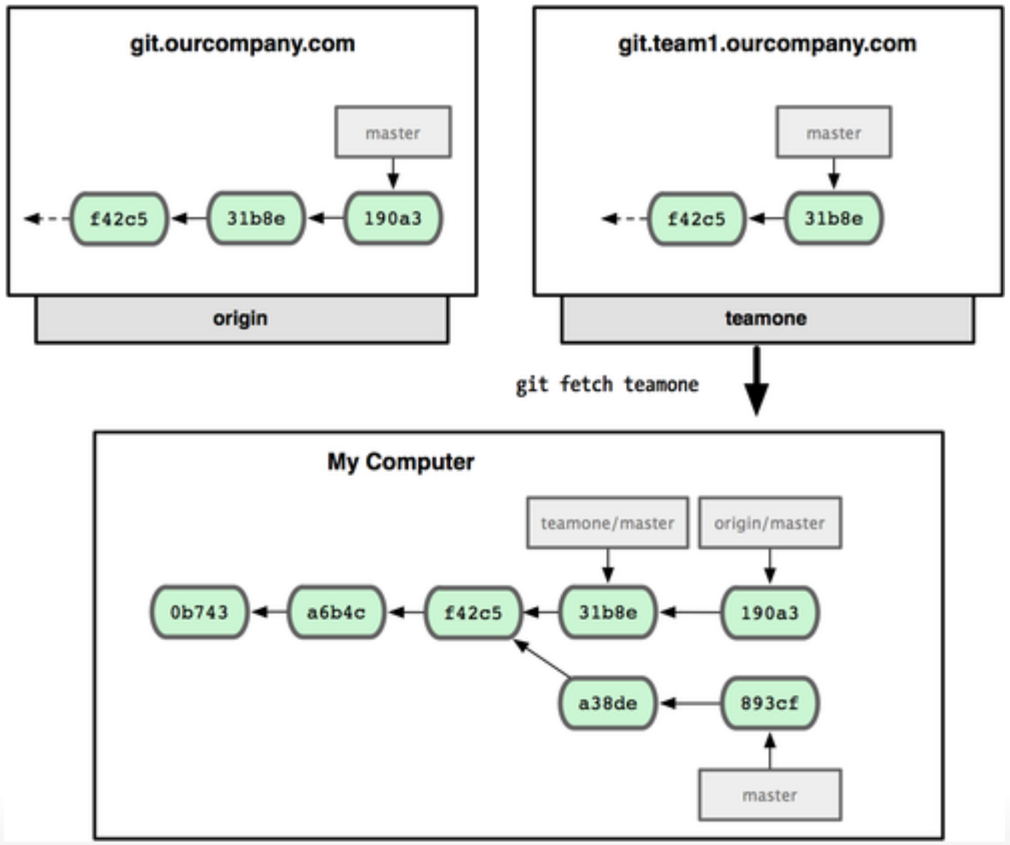




Git Branching - Remote Branch

- Multiple remote repositories





Git Branching - Remote Branch

- Push to the remote

```
$ git push origin serverfix
Counting objects: 20, done.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.74 KiB, done.
Total 15 (delta 5), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new branch]      serverfix -> serverfix
```

- git push origin serverfix = git push origin serverfix:serverfix

- Another person fetch origin

```
$ git fetch origin
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 5), reused 0 (delta 0)
Unpacking objects: 100% (15/15), done.
From git@github.com:schacon/simplegit
 * [new branch]      serverfix    -> origin/serverfix
```

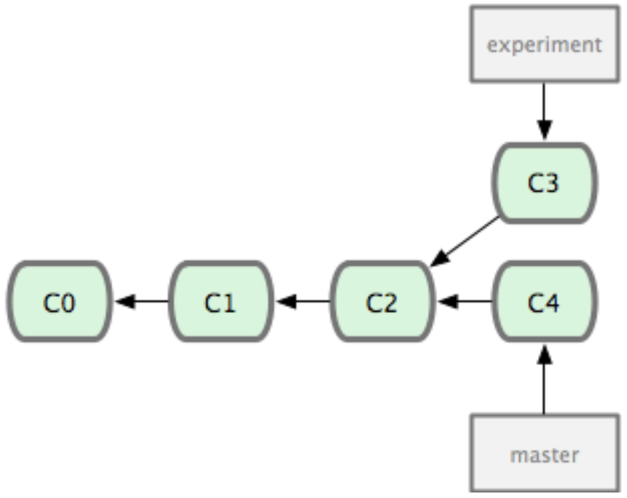
  - **git checkout -b sf origin/serverfix**

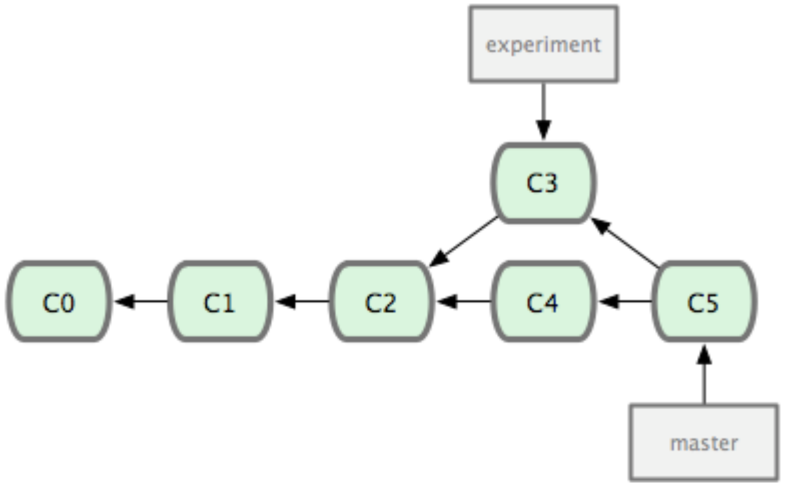
- Tracking Branches
  - Tracking branches are based on git push/pull.
  - git checkout --track origin/serverfix =
    git checkout -b serverfix origin/servierfix
  - **Running git pull while on one of tracking branches fetches all the remote references and then automatically merges in the corresponding remote branch.**

- Deleting remote branches
  - **git push origin :serverfix**

Git Branching - Rebasing

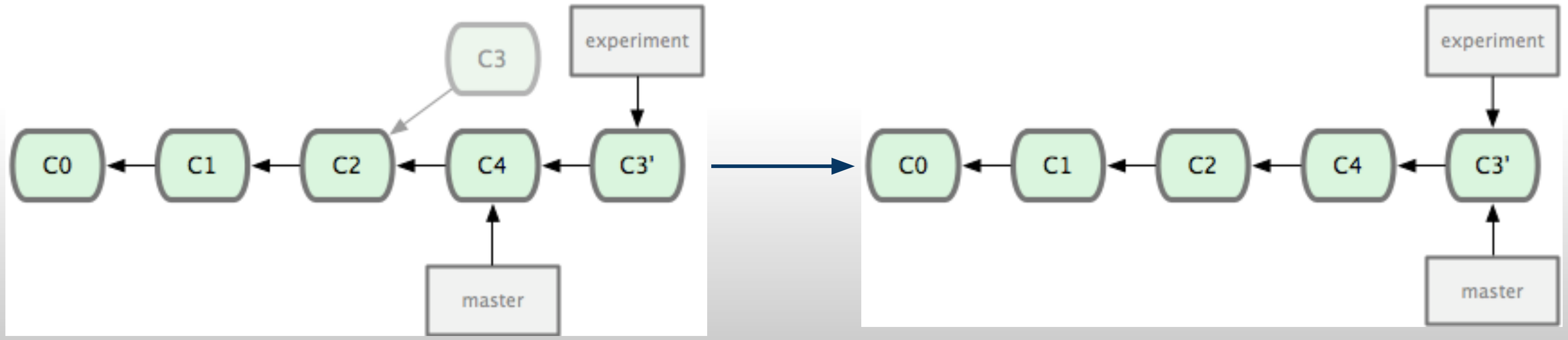
- `Basic rebase`





```
git checkout experiment
```
`git rebase master`

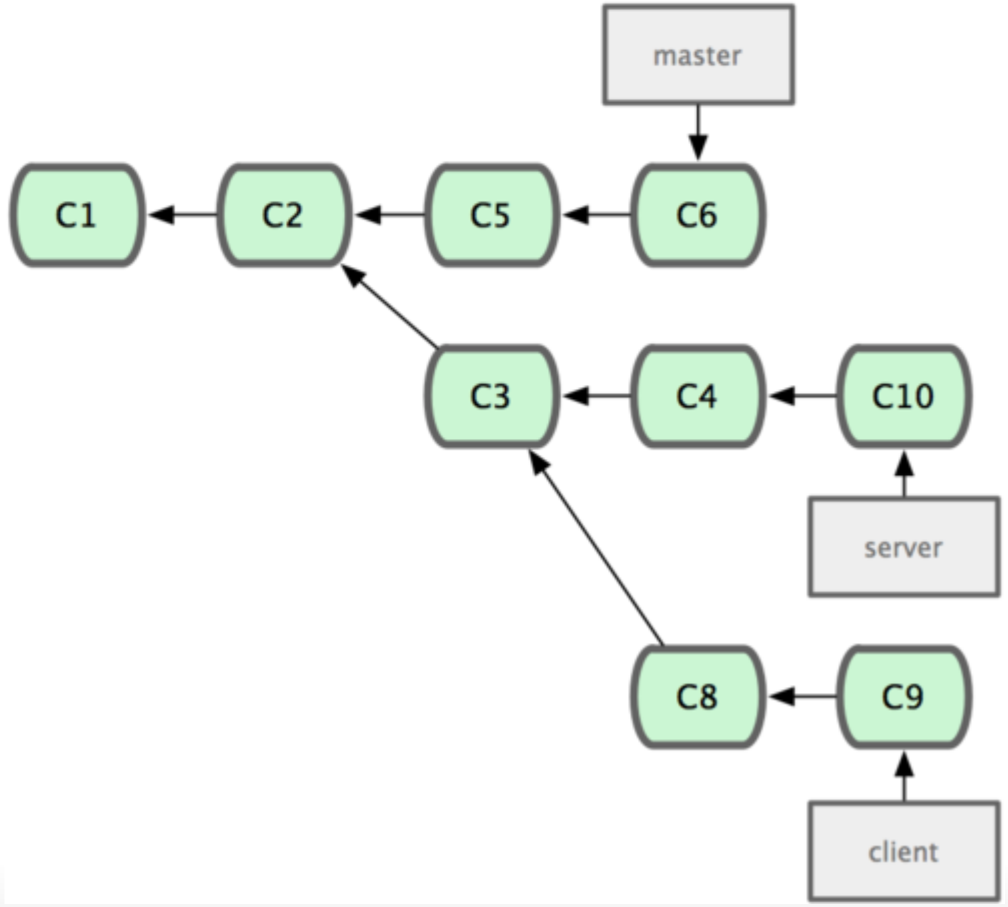
```
git checkout master
git merge experiment
```

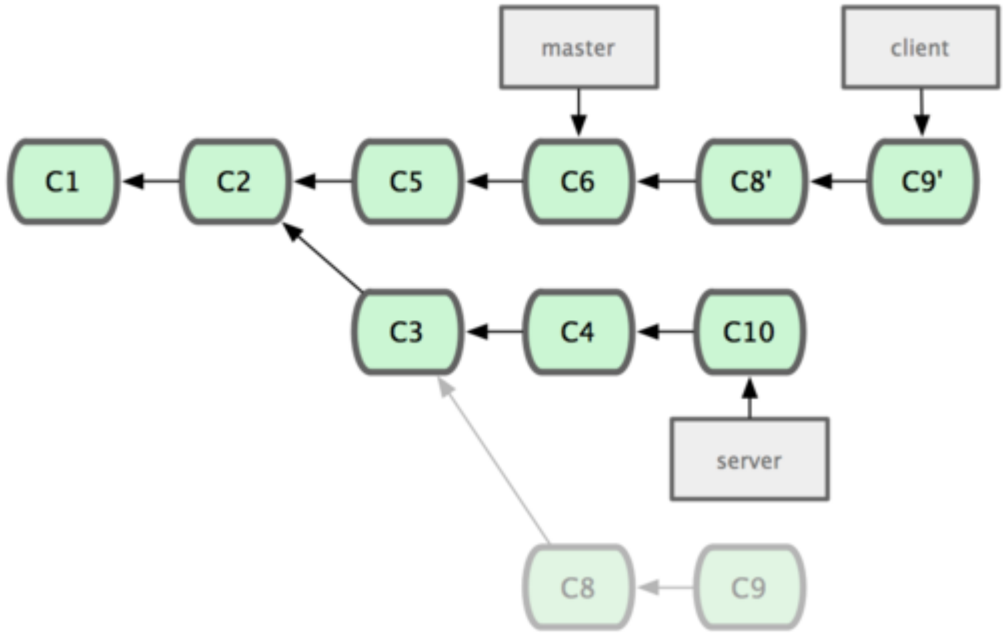


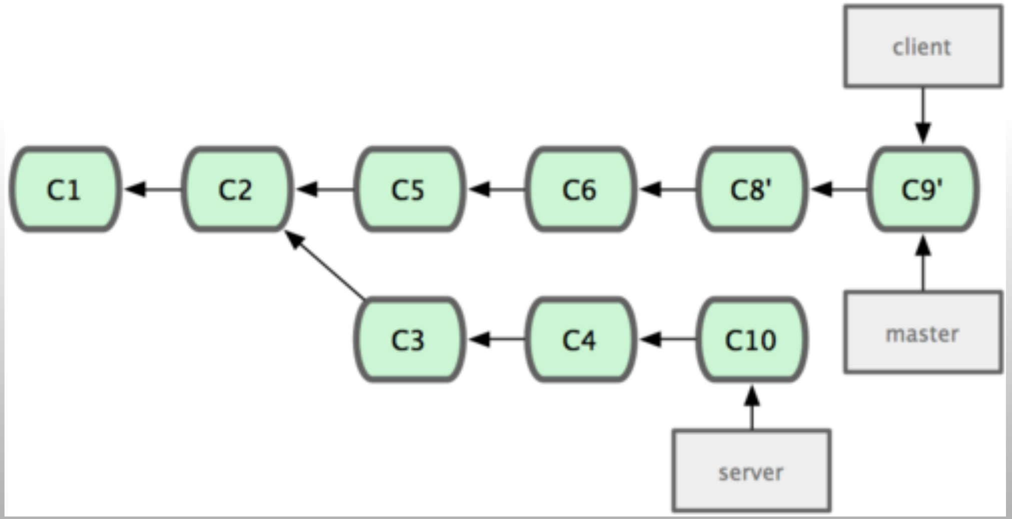
git checkout master
git merge experiment

Git Branching - Rebasing

- `More interesting rebase`
  `git rebase --onto master server client`



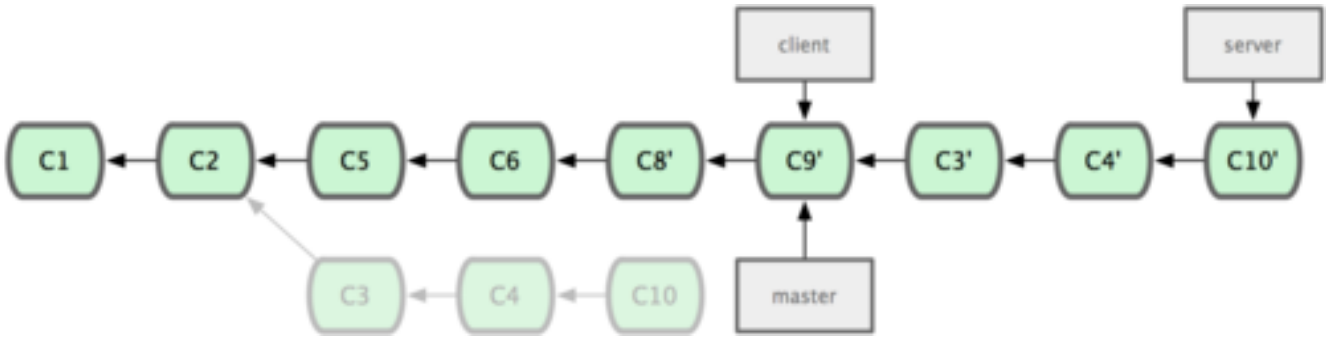




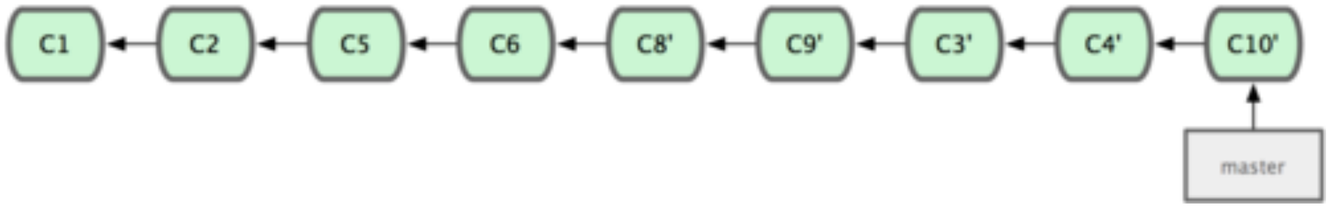
- **`How to do fast-forward ?`**

Git Branching - Rebasing

- `git rebase <base_branch> <topic_branch>`
  `git rebase master server`

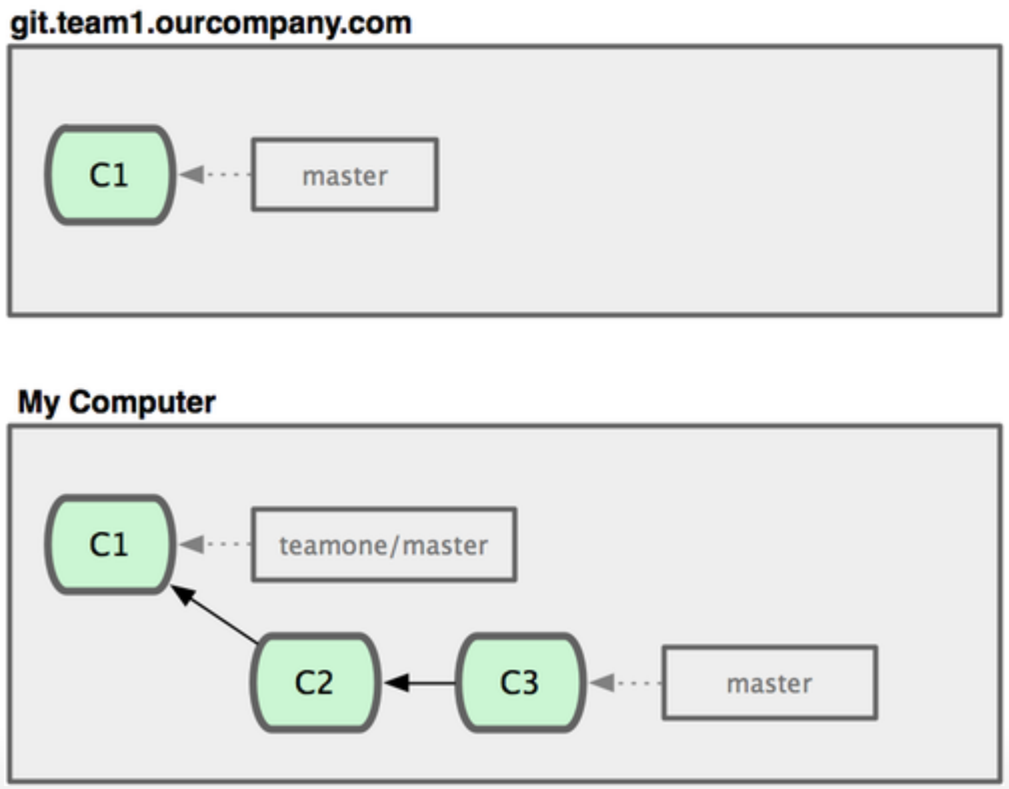


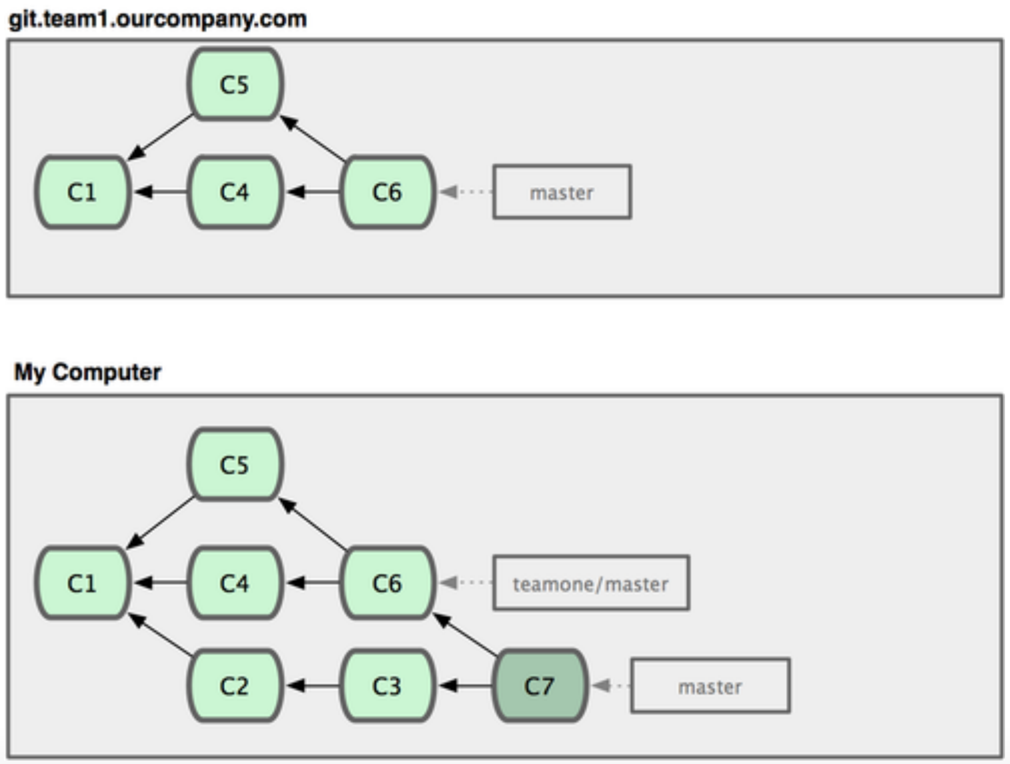
- `git checkout master`
  `git merge server`



Git Branching - Rebasing

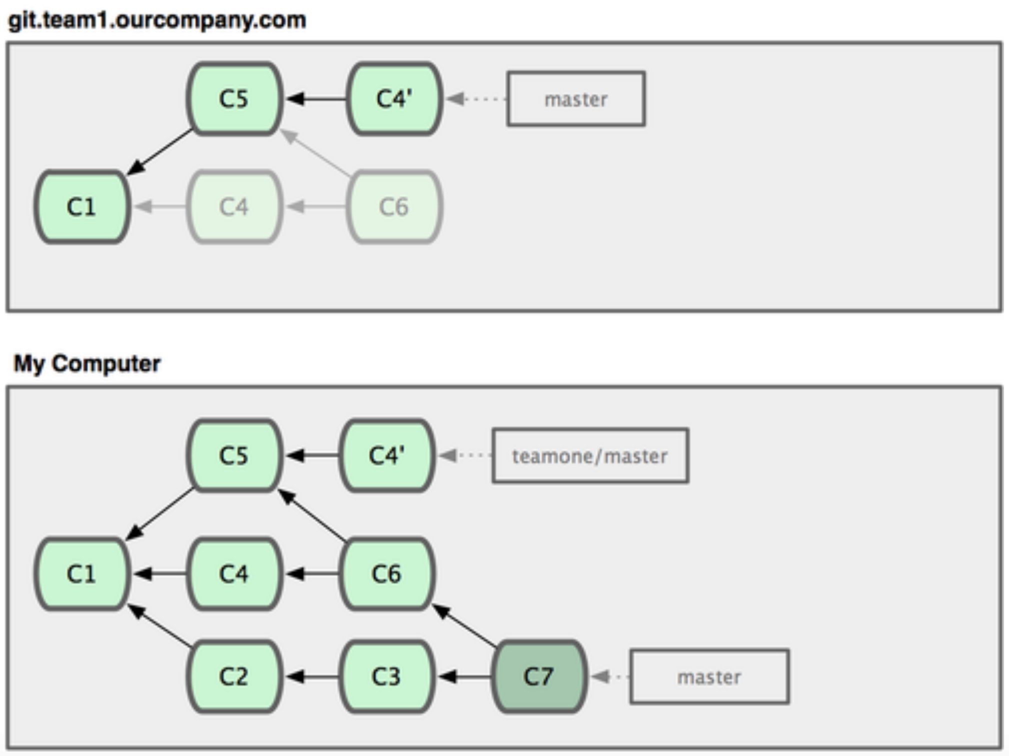
- `The perils of rebasing 1`

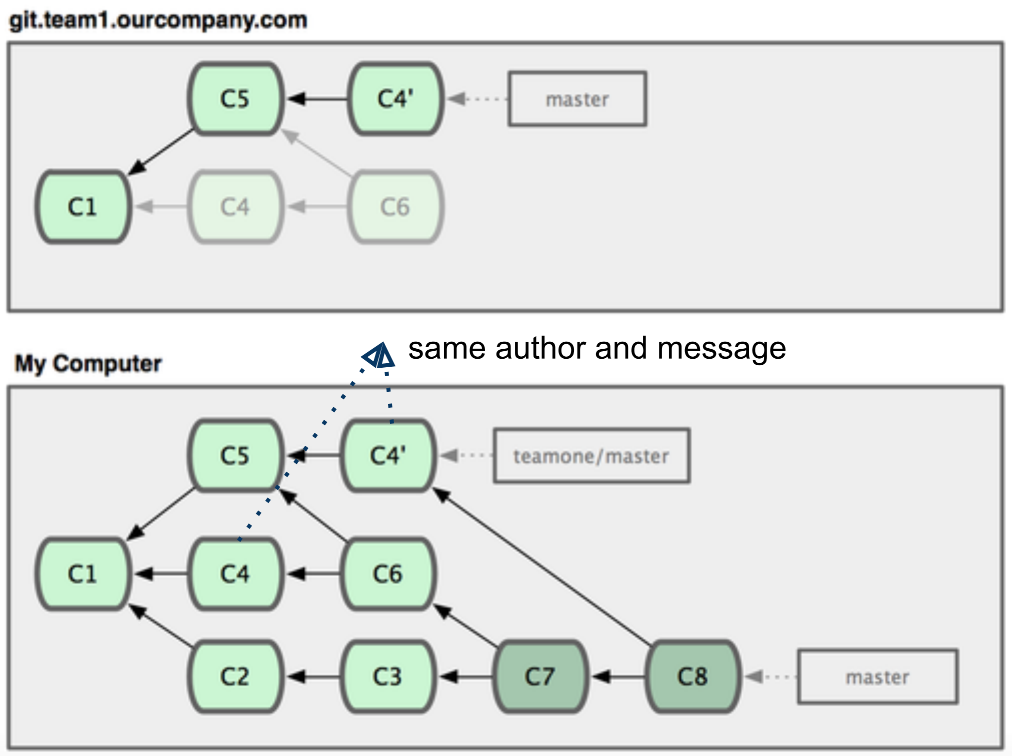




Git Branching - Rebasing

- `The perils of rebasing 2`





Do not rebase commits that you have pushed to a public repository.

Git on the Server - Protocol

- Local protocol
  - git clone /opt/git/project.git : copy or hardlink
    git clone file:///opt/git/project.git : seperate process
  - git pull /home/john/project
  - A local repository is fast only if you have fast access to the data.
  - Pro: You can run a command like git pull /home/john/project
  - Con: A repository on NFS is often slower than the repository over SSH.

- **SSH protocol**
  - git clone ssh://user@server/project.git = git clone user@server:project.git
  - You still need SSH for your own write commands.
  - Pros
    - **Authenticated write access to your repository over a network**
    - **Easy to set up**
    - **Access over SSH is secure** — all data transfer is encrypted and authenticated
    - **SSH is efficient,** making the data as compact as possible before transferring it
  - Cons
    - Can’t serve anonymous access of your repository

- Git protocol
  - Special daemon that comes packaged with Git
  - The fastest transfer protocol available
  - The lack of authentication
  - Use git:// for read-only access

- **HTTP/S protocol**
  - Normally, It is used for read-only access. Easy to set up.
  - Git push over HTTP is rarely used and needs any WebDAV server.
  - Not very resource intensive on your server
  - Relatively inefficient for the client

Git on the Server - How to Make Git Server

- Clone bare repository
  - `git clone --bare my_project my_project.git`
    `== cp -Rf my_project/.git my_project.git`

- Putting the bare repository on a Server
  - `scp -r my_project.git user@git.example.com:/opt/git`

- git account
  - ~/.ssh/authorized_keys

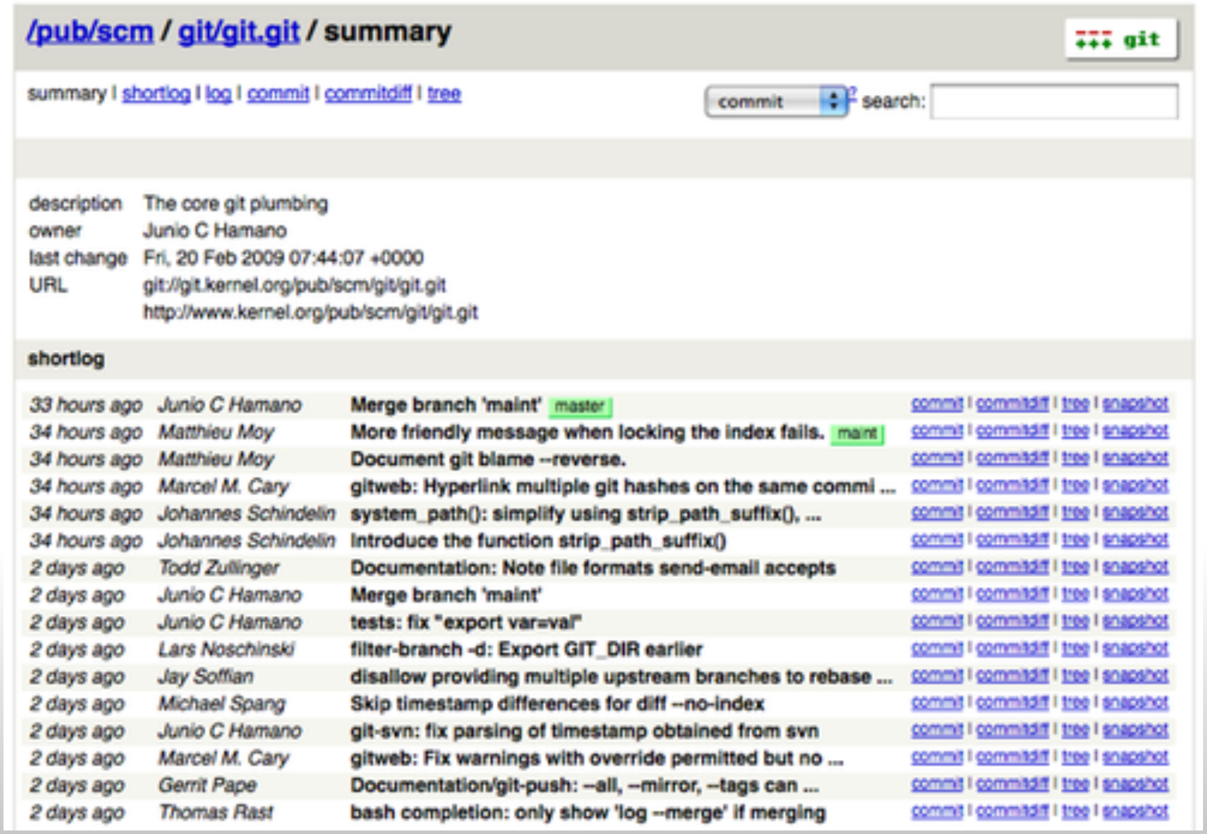
- SSH public key
  - ls -alt ~/.ssh
  - **ssh-keygen**
  - send email with id_rsa.pub

- Setting up the server
  - make git user and .ssh directory
    ```
    $ sudo adduser git && su git && mkdir .ssh
    ```
  - add SSH public keys to authorized_keys
    ```
    $ cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
    ```
  - make bare repository
    ```
    $ cd /opt/git && mkdir project.git && cd project.git && git --bare init
    ```
  - someone push his repository to the server
    ```
    # on Johns computer
    $ cd myproject && git init && git add . && git commit -m 'initial commit'
    $ git remote add origin git@gitserver:/opt/git/project.git
    $ git push origin master
    ```
  - another can clone the repository
    ```
    $ git clone git@gitserver:/opt/git/project.git
    ```
  - Setting up git-shell
    ```
    $ sudo vim /etc/passwd -> git:x:1000:1000::/home/git:/usr/bin/git-shell
    ```

Git Server - Anonymous Read Acess for the Repo

- Anonymous access through HTTP
  1. Run a static web server
  2. The server's document root where your Git repositories
     - Apache Virtual Host Setting
     - Set the Unix user group of the `/opt/git` directories to `www-data`
  3. Enable `post-update` hook

- GitWeb : `git instaweb`



/pub/scm / git/git.git / summary

summary | shortlog | log | commit | commitdiff | tree

commit search:

description	The core git plumbing
owner	Junio C Hamano
last change	Fri, 20 Feb 2009 07:44:07 +0000
URL	git://git.kernel.org/pub/scm/git/git.git
	http://www.kernel.org/pub/scm/git/git.git

shortlog

33 hours ago	Junio C Hamano	Merge branch 'maint' master	commit | commitdiff | tree | snapshot
34 hours ago	Matthieu Moy	More friendly message when locking the index fails. maint	commit | commitdiff | tree | snapshot
34 hours ago	Matthieu Moy	Document git blame --reverse.	commit | commitdiff | tree | snapshot
34 hours ago	Marcel M. Cary	gitweb: Hyperlink multiple git hashes on the same commi ...	commit | commitdiff | tree | snapshot
34 hours ago	Johannes Schindelin	system_path(): simplify using strip_path_suffix(), ...	commit | commitdiff | tree | snapshot
34 hours ago	Johannes Schindelin	Introduce the function strip_path_suffix()	commit | commitdiff | tree | snapshot
2 days ago	Todd Zullinger	Documentation: Note file formats send-email accepts	commit | commitdiff | tree | snapshot
2 days ago	Junio C Hamano	Merge branch 'maint'	commit | commitdiff | tree | snapshot
2 days ago	Junio C Hamano	tests: fix "export var=val"	commit | commitdiff | tree | snapshot
2 days ago	Lars Noschinski	filter-branch -d: Export GIT_DIR earlier	commit | commitdiff | tree | snapshot
2 days ago	Jay Soffian	disallow providing multiple upstream branches to rebase ...	commit | commitdiff | tree | snapshot
2 days ago	Michael Spang	Skip timestamp differences for diff --no-index	commit | commitdiff | tree | snapshot
2 days ago	Junio C Hamano	git-svn: fix parsing of timestamp obtained from svn	commit | commitdiff | tree | snapshot
2 days ago	Marcel M. Cary	gitweb: Fix warnings with override permitted but no ...	commit | commitdiff | tree | snapshot
2 days ago	Gerrit Pape	Documentation/git-push: --all, --mirror, --tags can ...	commit | commitdiff | tree | snapshot
2 days ago	Thomas Rast	bash completion: only show 'log --merge' if merging	commit | commitdiff | tree | snapshot

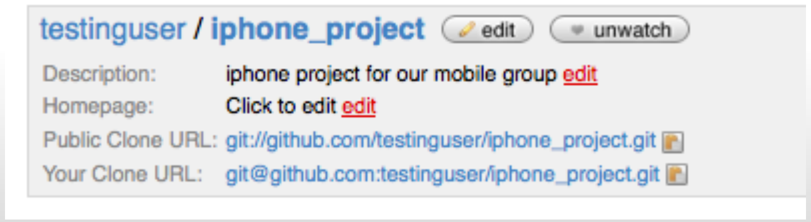
Git Server - Access Control (Gitosis, Gitolite), Hosted Git

- **Gitosis** help you manage the `authorized_keys` file as well as implement some simple access controls.

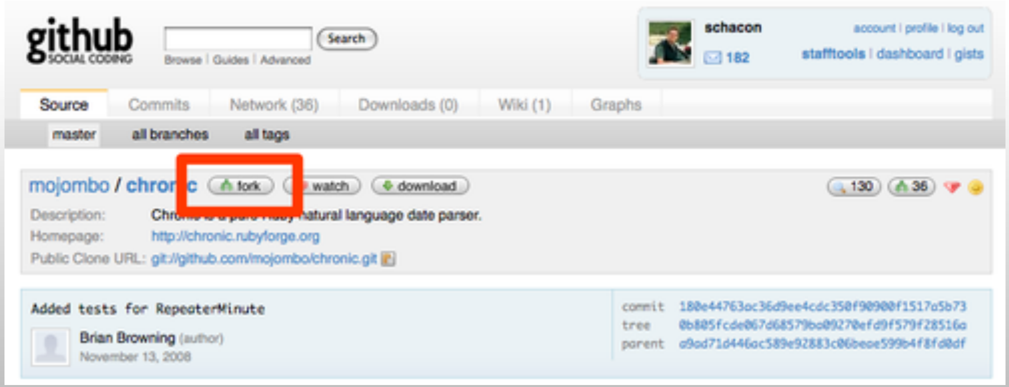
- **Gitolite** is an authorization layer on top of Git.
  - Authentication is identifying who the user is.
  - Authorization is deciding if he is allowed to do what he is attempting to.
  - Specify permissions not just by repository, but also **by branch or tag names** within each repository.

- **Hosted Git**
  - GitHub is by far the largest open source Git hosting site.
  - charges for accounts that maintain private repositories
  - Sign up, enroll the SSH key.
  - Create Repository : name and description
  - Push up your master branch

    ```
    $ git init && git add . && git commit -m 'initial commit'
    $ git remote add origin git@github.com:testinguser/iphone_project.git
    $ git push origin master
    ```
  - The Your Clone URL is a read/write SSH-based URL.
  - Admin page -> "Add another collaborator"
  - Fork a project in order to contribute on it.


testinguser / iphone_project

Description: iphone project for our mobile group edit
Homepage: Click to edit edit
Public Clone URL: git://github.com/testinguser/iphone_project.git
Your Clone URL: git@github.com:testinguser/iphone_project.git


mojombo / chronic

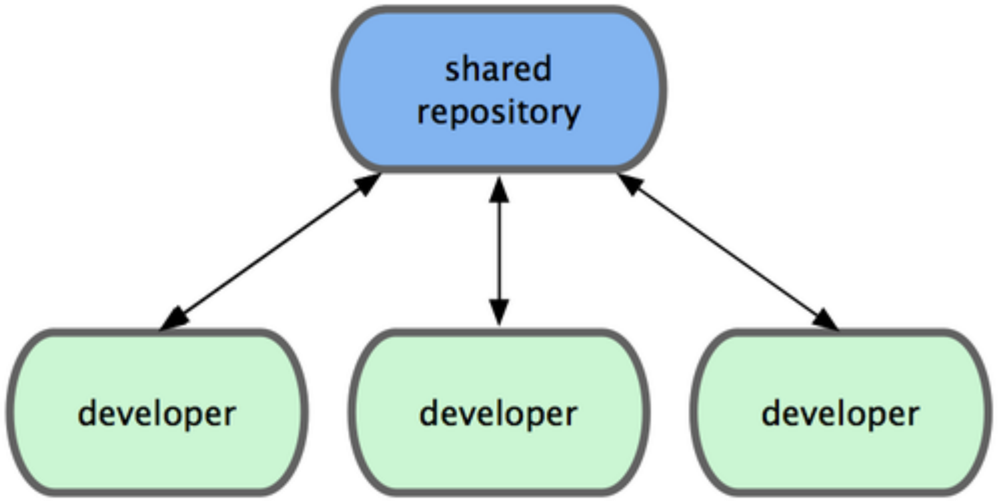
Description: Chronic is a pure Ruby natural language date parser.
Homepage: http://chronic.rubyforge.org
Public Clone URL: git://github.com/mojombo/chronic.git

Added tests for RepeaterMinute
Brian Browning (author)
November 13, 2008

commit 180e44763ac36d9ee4cdc350f90900f1517a5b73
tree 0b805fcde067d68579ba09270efd9f579f28516a
parent a9ad71d446ac589e92883c06beae599b4f8fd0df

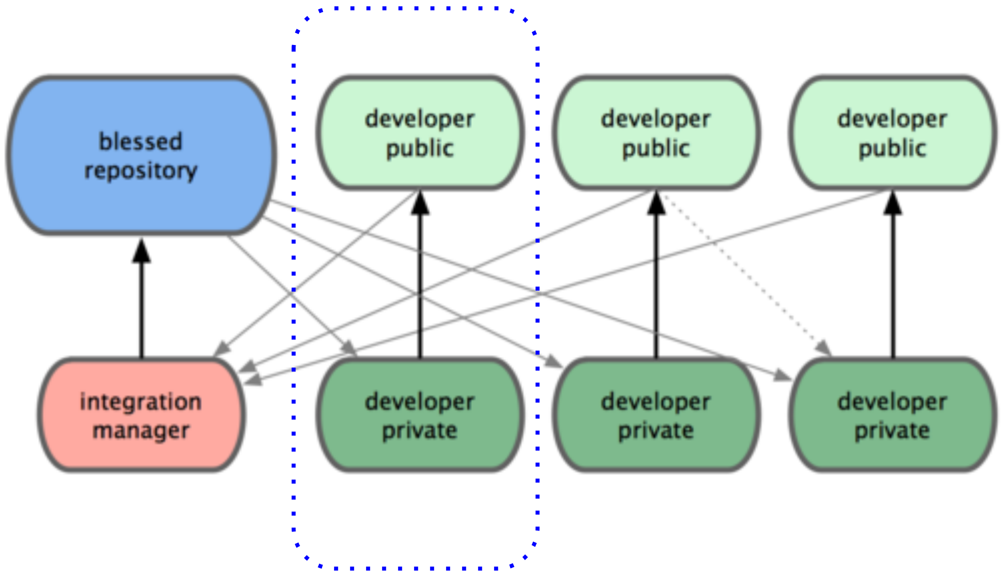
Distributed Git - Centralized Workflow

- `Centralized workflow`
  Shared repo can only do fast-forward merge.



Distributed Git - Integration-Manager Workflow

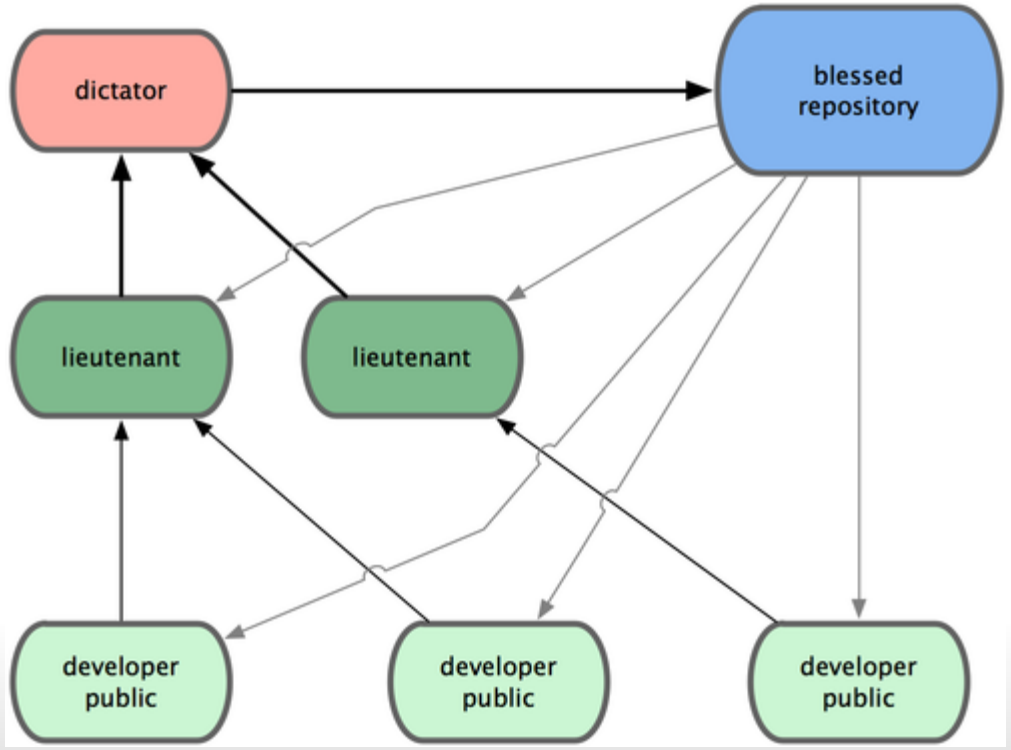
- `Integration-Manager workflow`
  Each party can work at their own pace.



Distributed Git - Dictator and Lieutenants Workflow

- `Dictator and Lieutenants workflow`

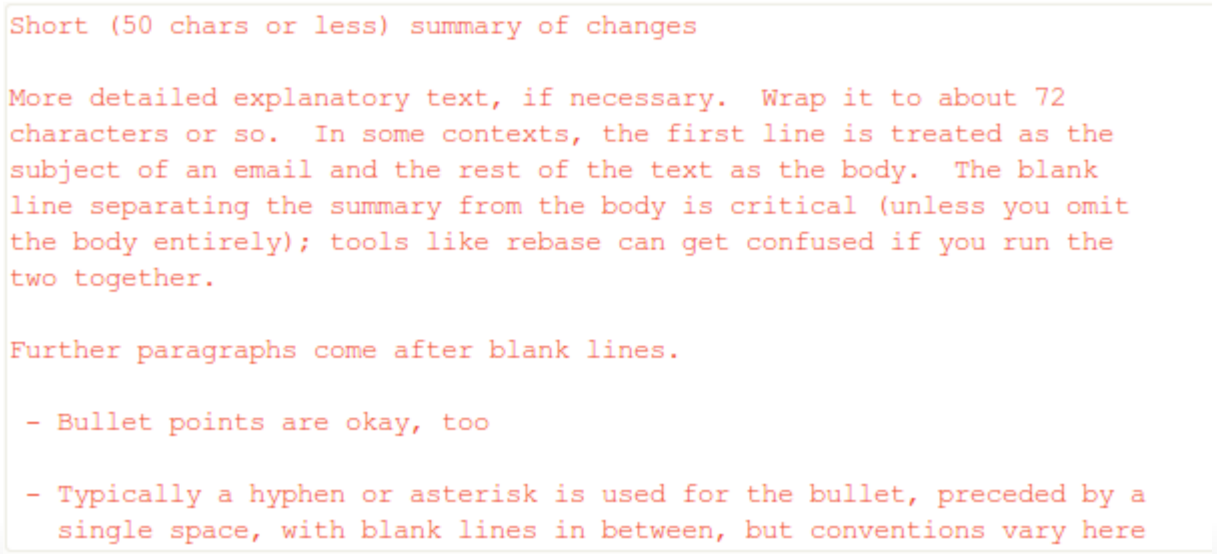
1. Regular developers work on their topic branch and rebase their work on top of master. The master branch is that of the dictator.
2. Lieutenants merge the developers’ topic branches into their master branch.
3. The dictator merges the lieutenants’ master branches into the dictator’s master branch.
4. The dictator pushes their master to the reference repository so the other developers can rebase on it.



Benevolent dictator workflow

Contributing to a project - Commit Guidelines

- Commit Guidelines
  - whitespaces errors check - git diff --check
  - *Try to make each commit a logically separate changeset.*
  - *Try to make your changes digestible.*
  - **one commit per issue**
  - Use the imperative present tense - Instead of "I added tests for " use "Add tests for"



```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded by a
   single space, with blank lines in between, but conventions vary here
```

Contributing to a project - **Private Small Team**

- John clones the repository, makes a change, and commits locally.

```
# John's Machine
$ git clone john@githost:simplegit.git
Initialized empty Git repository in /home/john/simplegit/.git/
...
$ cd simplegit/
$ vim lib/simplegit.rb
$ git commit -am 'removed invalid default value'
[master 738ee87] removed invalid default value
 1 files changed, 1 insertions(+), 1 deletions(-)
```

- Jessica clones the repository and commits a change.

```
# Jessica's Machine
$ git clone jessica@githost:simplegit.git
Initialized empty Git repository in /home/jessica/simplegit/.git/
...
$ cd simplegit/
$ vim TODO
$ git commit -am 'add reset task'
[master fbff5bc] add reset task
 1 files changed, 1 insertions(+), 0 deletions(-)
```

- Jessica pushes her work up to the server:

```
# Jessica's Machine
$ git push origin master
...
To jessica@githost:simplegit.git
   1edee6b..fbff5bc  master -> master
```

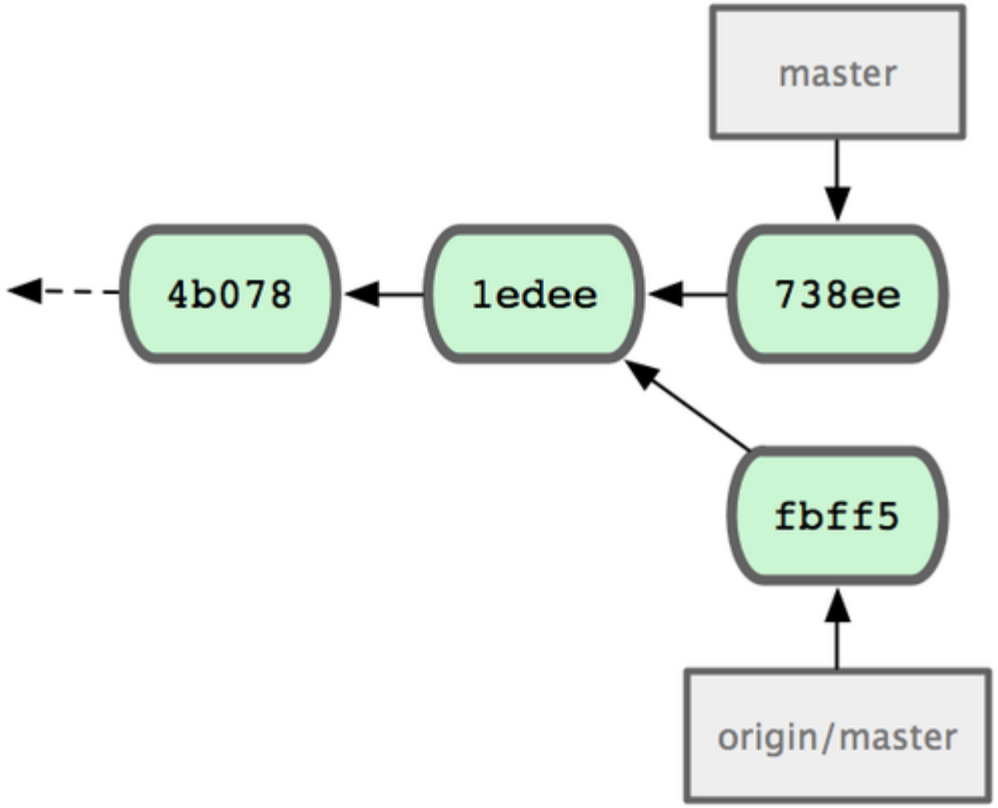
- John tries to push his change up, too:

```
# John's Machine
$ git push origin master
To john@githost:simplegit.git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to 'john@githost:simplegit.git'
```

Contributing to a project - **Private Small Team**

- John has to fetch Jessica's changes and merge them in.

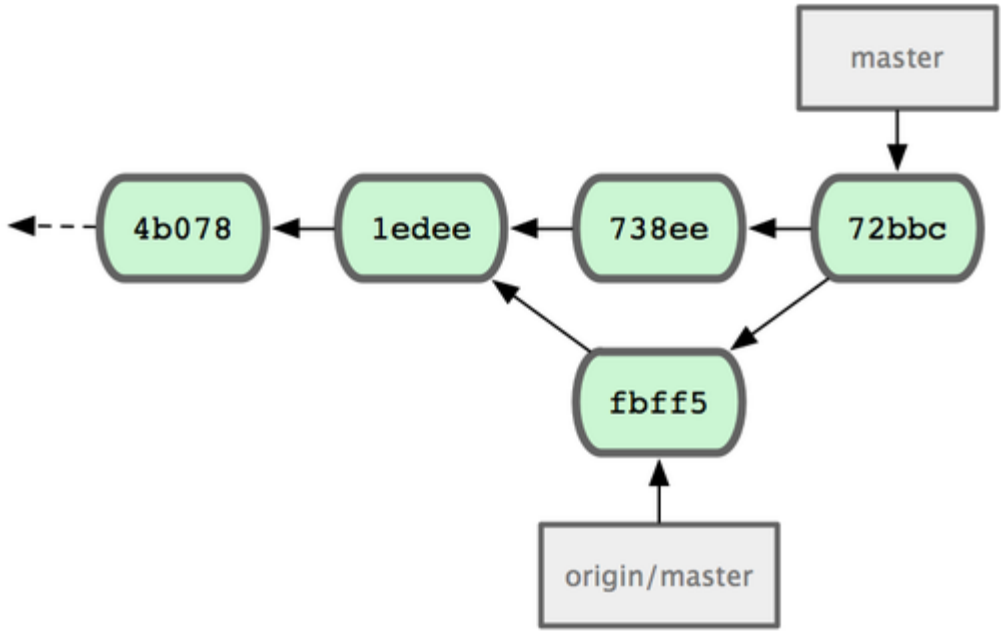
```
$ git fetch origin
...
From john@githost:simplegit
 + 049d078...fbff5bc master     -> origin/master
```



Contributing to a project - **Private Small Team**

- John has to merge them into his own work before he is allowed to push:

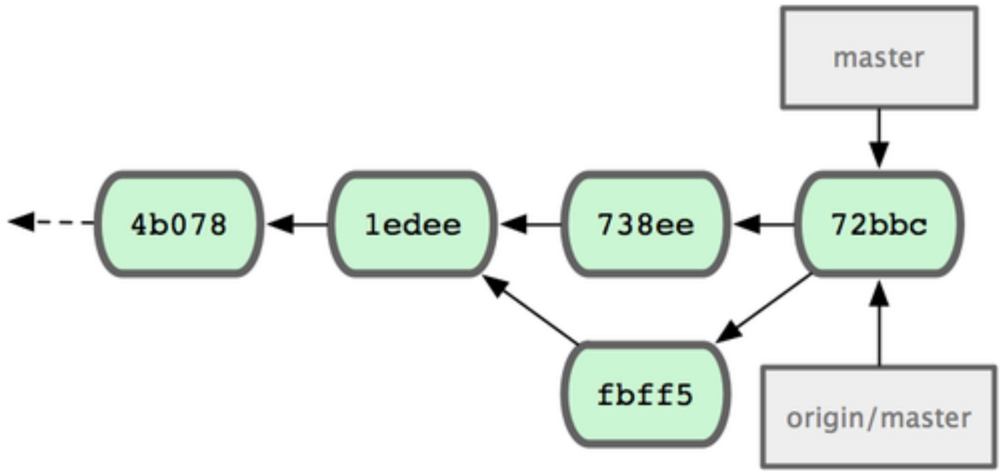
```
$ git merge origin/master
Merge made by recursive.
 TODO |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```



Contributing to a project - Private Small Team

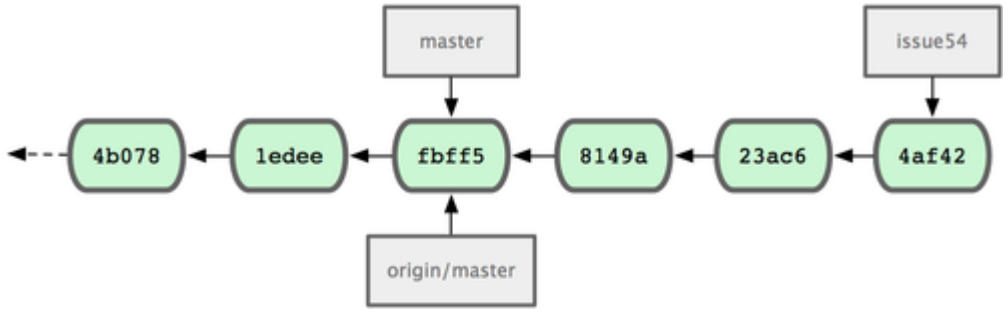
- John can push his new merged work up to the server:

```
$ git push origin master
...
To john@githost:simplegit.git
   fbff5bc..72bbc59  master -> master
```



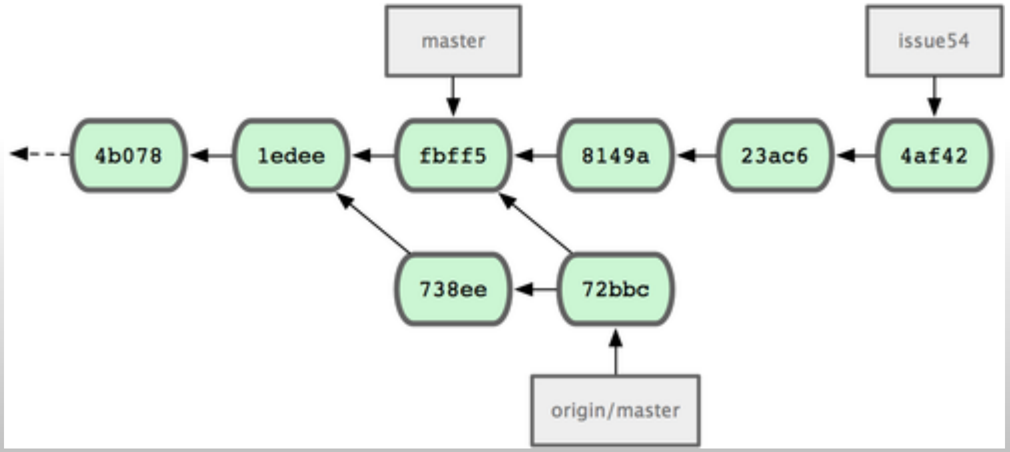
Contributing to a project - **Private Small Team**

- In the meantime, Jessica's created a topic branch called issue54 and done three commits.



- Jessica wants to sync up with John, so she fetches:

```
# Jessica's Machine
$ git fetch origin
...
From jessica@githost:simplegit
   fbff5bc..72bbc59  master     -> origin/master
```



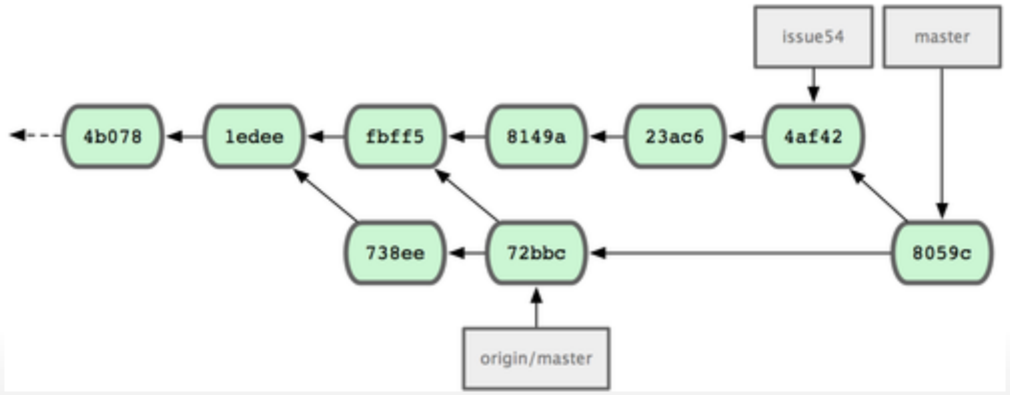
Contributing to a project - **Private Small Team**

- Jessica thinks her topic branch is ready, but she wants to know what she has to merge her work into so that she can push. She runs `git log` to find out:

```
$ git log --no-merges origin/master ^issue54
commit 738ee872852dfaa9d6634e0dea7a324040193016
Author: John Smith <jsmith@example.com>
Date:   Fri May 29 16:01:27 2009 -0700

    removed invalid default value
```

- Jessica can merge her topic work into her master branch, merge John's work (`origin/master`) into her `master` branch, and then push back to the server again.

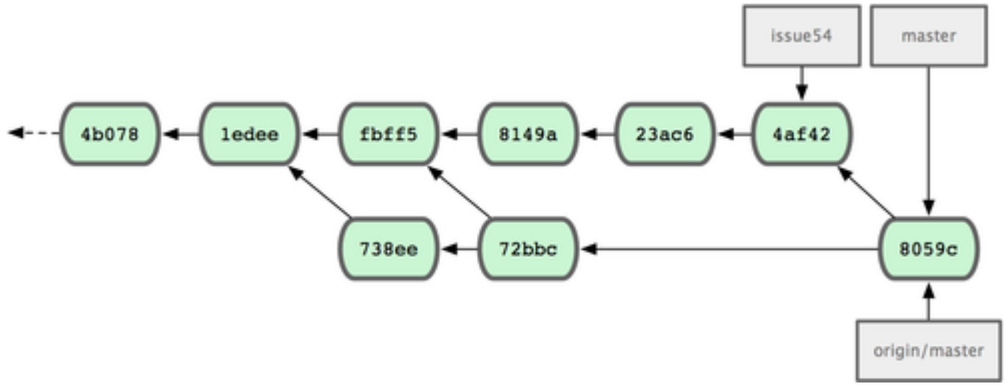
```
$ git checkout master
Switched to branch "master"
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.

$ git merge issue54
Updating fbff5bc..4af4298
Fast forward
 README           |    1 +
 lib/simplegit.rb |    6 +++++-
 2 files changed, 6 insertions(+), 1 deletions(-)

$ git merge origin/master
Auto-merging lib/simplegit.rb
Merge made by recursive.
 lib/simplegit.rb |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

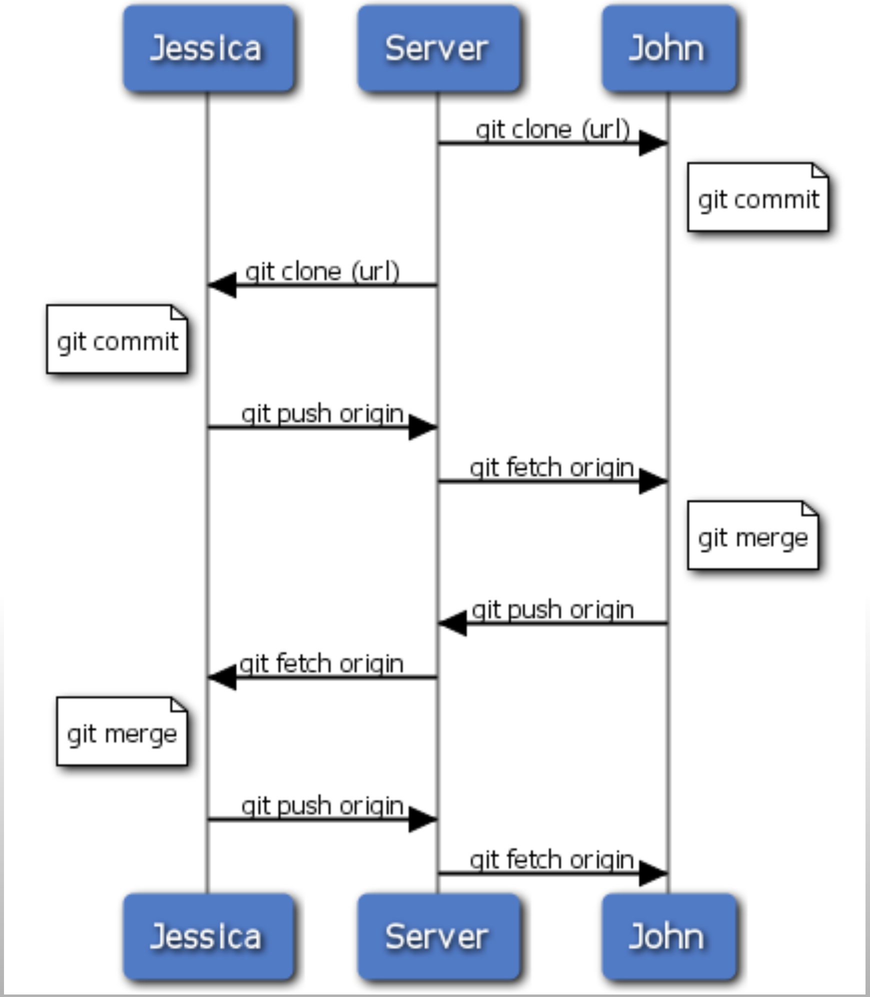


Contributing to a project - **Private Small Team**

- Now origin/master is reachable from Jessica's master branch, so she should be able to successfully push (assuming John hasn't pushed again in the meantime):

```
$ git push origin master
...
To jessica@githost:simplegit.git
   72bbc59..8059c15  master -> master
```





Contributing to a project - **Private Managed Team**

- John and Jessica are working together on one feature, while Jessica and Josie are working on a second.
- Jessica creates a new branch for featureA and does some work on it there:

```
# Jessica's Machine
$ git checkout -b featureA
Switched to a new branch "featureA"
$ vim lib/simplegit.rb
$ git commit -am 'add limit to log function'
[featureA 3300904] add limit to log function
 1 files changed, 1 insertions(+), 1 deletions(-)
```

- Jessica doesn’t have push access to the master branch — only the integrators do — so she has to push to another branch in order to collaborate with John:

```
$ git push origin featureA
...
To jessica@githost:simplegit.git
 * [new branch]      featureA -> featureA
```

- While she waits for feedback from John, Jessica decides to start working on featureB with Josie.

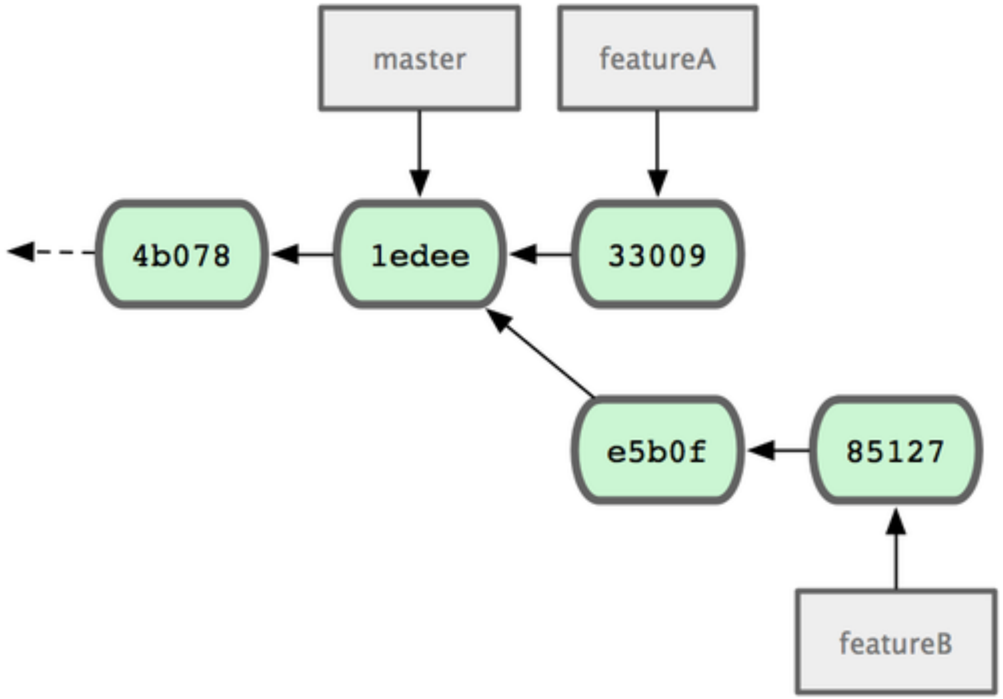
```
# Jessica's Machine
$ git fetch origin
$ git checkout -b featureB origin/master
Switched to a new branch "featureB"
```

- Jessica makes a couple of commits on the featureB branch:

```
$ vim lib/simplegit.rb
$ git commit -am 'made the ls-tree function recursive'
[featureB e5b0fdc] made the ls-tree function recursive
 1 files changed, 1 insertions(+), 1 deletions(-)
$ vim lib/simplegit.rb
$ git commit -am 'add ls-files'
[featureB 8512791] add ls-files
 1 files changed, 5 insertions(+), 0 deletions(-)
```

Contributing to a project - Private Managed Team

- Jessica's repository looks like:



Contributing to a project - **Private Managed Team**

- She's ready to push up her work, but gets an e-mail from Josie that a branch with some initial work on it was already pushed to the server as `featureBee`.
```
$ git fetch origin
...
From jessica@githost:simplegit
 * [new branch]      featureBee -> origin/featureBee
```
- Jessica can now merge this into the work she did with `git merge`:
```
$ git merge origin/featureBee
Auto-merging lib/simplegit.rb
Merge made by recursive.
 lib/simplegit.rb |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)
```
- She pushes the merged work in her `featureB` to the `featureBee` branch on the server.
```
$ git push origin featureB:featureBee
...
To jessica@githost:simplegit.git
   fba9af8..cd685d1  featureB -> featureBee
```
- John pushed some changes to the `featureA`. So she want to know what the changes are.
```
$ git fetch origin
...
From jessica@githost:simplegit
   3300904..aad881d  featureA   -> origin/featureA
```
- Jessica can see what has been changed with `git log`:
```
$ git log origin/featureA ^featureA
commit aad881d154acdaeb2b6b18ea0e827ed8a6d671e6
Author: John Smith <jsmith@example.com>
Date:   Fri May 29 19:57:33 2009 -0700

    changed log output to 30 from 25
```

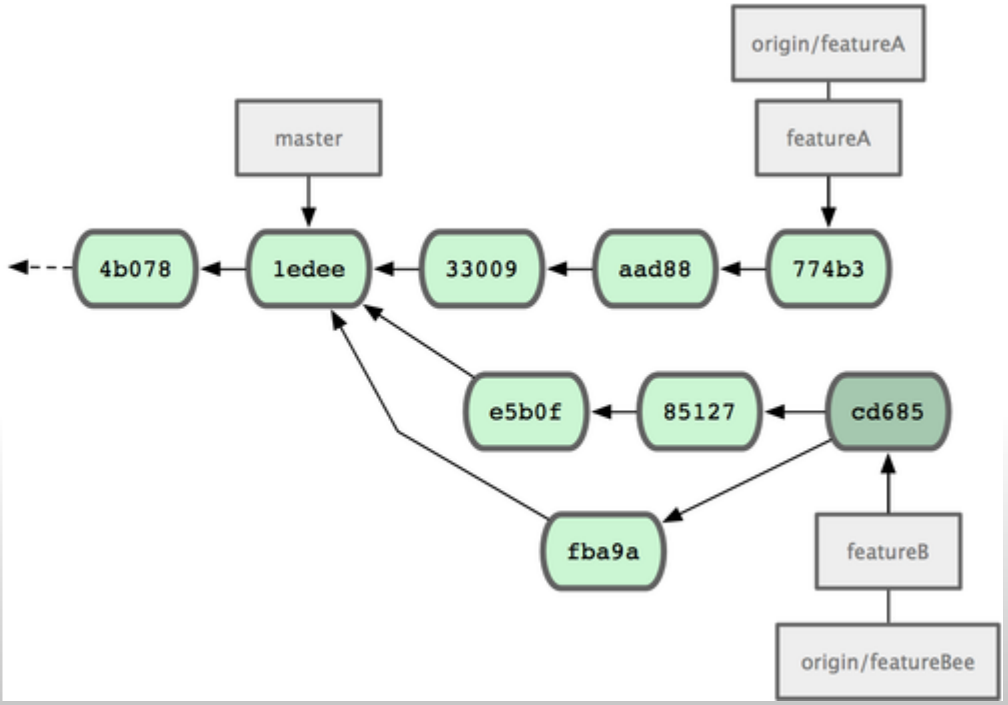
Contributing to a project - **Private Managed Team**

- Jessica merges John’s work into her own featureA.

```
$ git checkout featureA
Switched to branch "featureA"
$ git merge origin/featureA
Updating 3300904..aad881d
Fast forward
 lib/simplegit.rb |   10 +++++++++-
1 files changed, 9 insertions(+), 1 deletions(-)
```

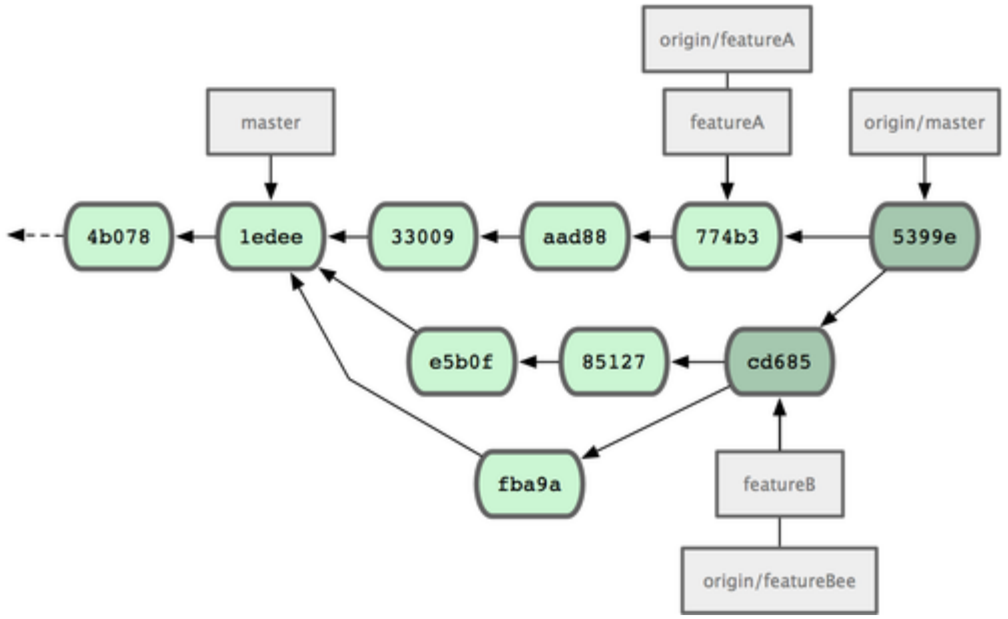
- Jessica wants to tweak something.

```
$ git commit -am 'small tweak'
[featureA 774b3ed] small tweak
 1 files changed, 1 insertions(+), 1 deletions(-)
$ git push origin featureA
...
To jessica@githost:simplegit.git
   3300904..774b3ed  featureA -> featureA
```



Contributing to a project - **Private Managed Team**

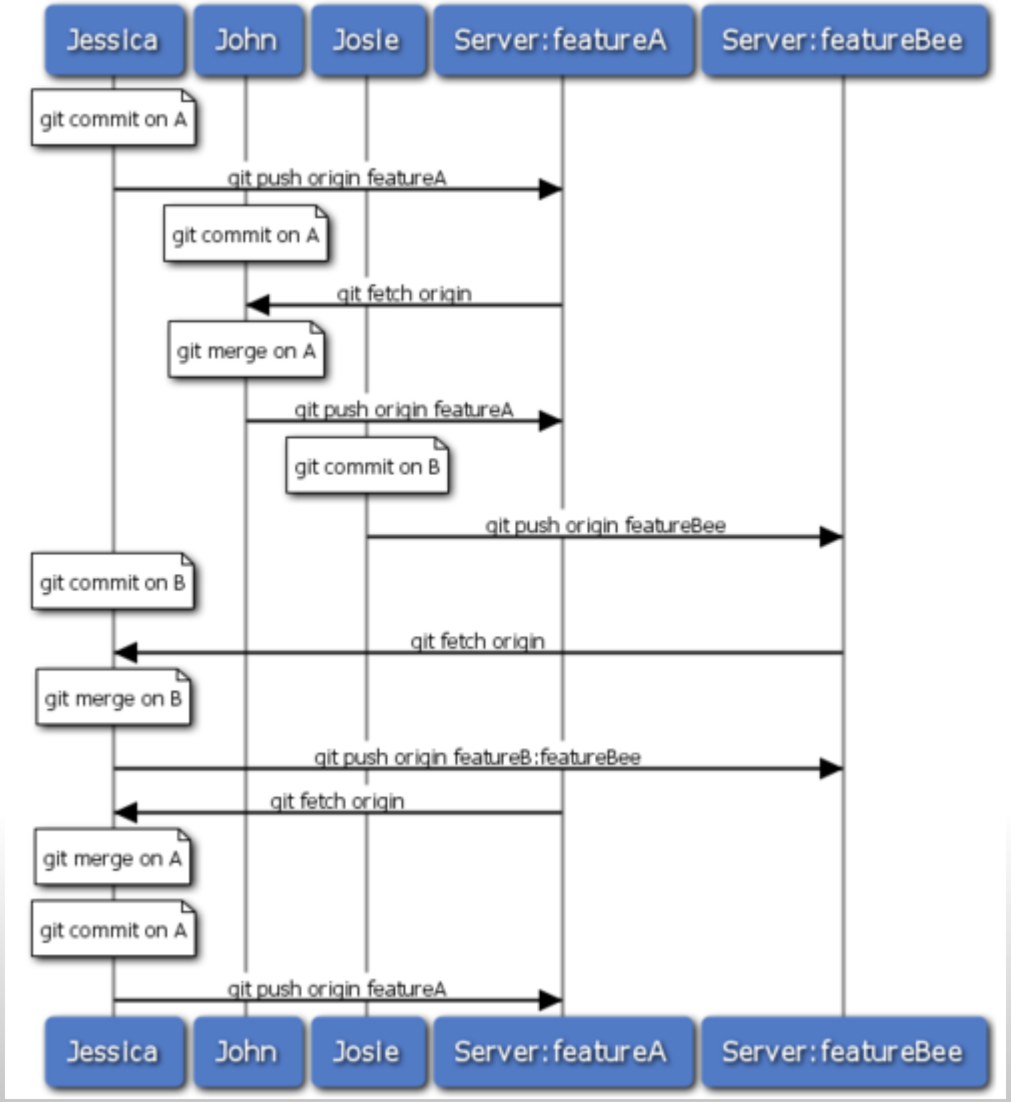
- Jessica, Josie, and John inform the integrators that the featureA and featureBee branches on the server are ready for integration into the mainline.



Jessica’s history

Contributing to a project - **Private Managed Team**

- Multiple teams can work in parallel (without necessarily having to involve or impede the entire team).



Contributing to a Pubilc Project - request-pull, format-patch

- Fork - public small project
  1. Clone the main repository.
  2. Create a topic branch you're planning to contribute.
  3. Do something in it.
  4. Create your own writable fork of the project.
  5. git push myfork featureA
  6. White 'pull request' message with 'git request-pull'

```
$ git request-pull origin/master myfork
The following changes since commit 1edee6b1d61823a2de3b09c160d7080b8d1b3a40:
  John Smith (1):
       added a new function

are available in the git repository at:

  git://githost/simplegit.git featureA

Jessica Smith (2):
      add limit to log function
      change log output to 30 from 25

 lib/simplegit.rb |   10 +++++++++-
 1 files changed, 9 insertions(+), 1 deletions(-)
```

- Send Emails - public large project
  - Create the content of mail with 'git format-patch -M'

```
$ git format-patch -M origin/master
0001-add-limit-to-log-function.patch
0002-changed-log-output-to-30-from-25.patch
```

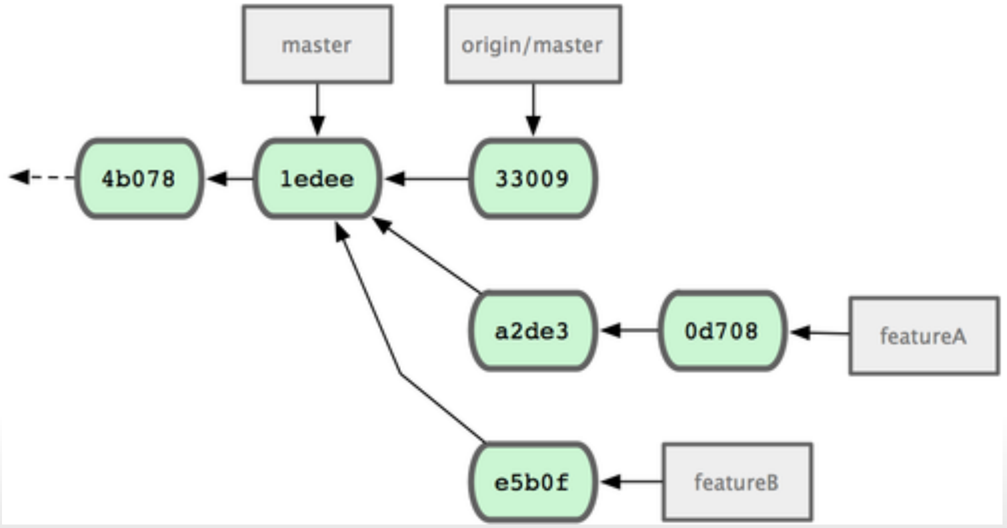
  - Send it with 'git send-email'.
    Read Docuementation/SubmittingPatches for more details.
    (How to set the configurations of email)

Contributing to a Public Project - Topic Branches

- **Having work themes isolated into topic branches**
  That makes it easier for you to rebase your work if the tip of the main repository has moved in the meantime.

- Don't continue working on the topic branch you just pushed up.

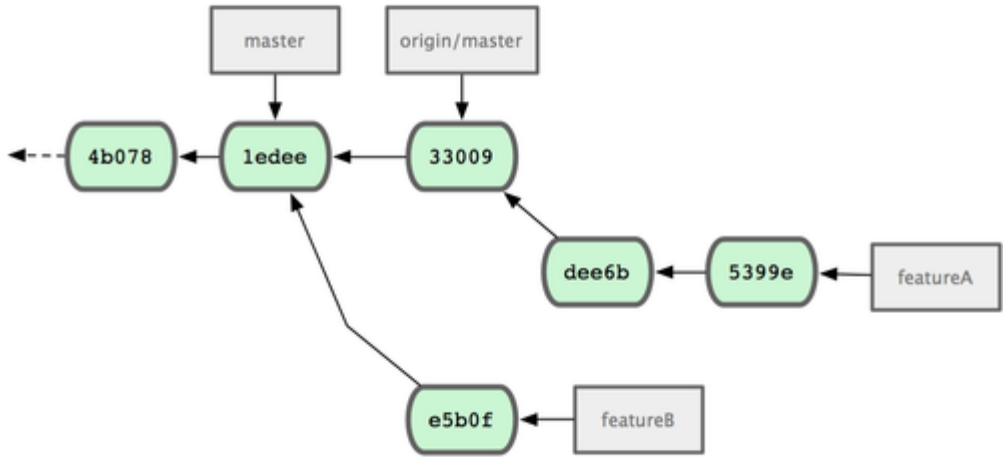
```
$ git checkout -b featureB origin/master
$ (work)
$ git commit
$ git push myfork featureB
$ (email maintainer)
$ git fetch origin
```



Contributing to a **Public Project - Topic Branches**

- When the maintainer no longer cleanly merges,
  Rebase that branch on top of `origin/master`, resolve the conflicts for the maintainer, and then resubmit your changes:

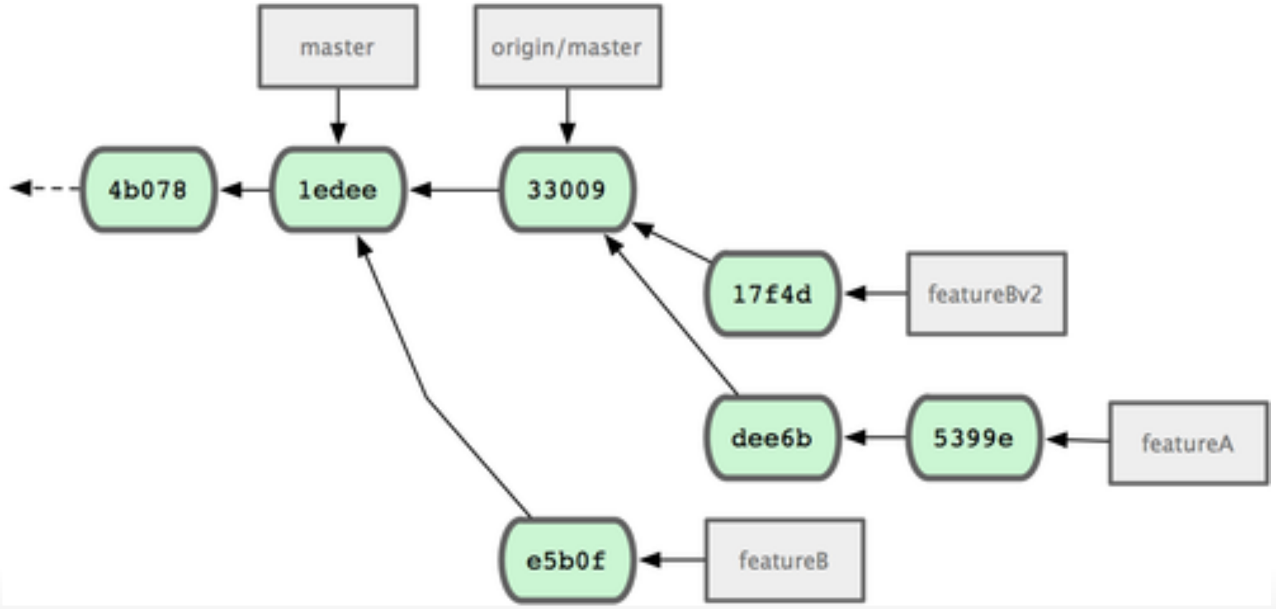
```
$ git checkout featureA
$ git rebase origin/master
$ git push -f myfork featureA
```



Contributing to a Public Project - Topic Branches

- The maintainer has looked at work in featureB and likes the concept but would like you to change an implementation detail.

```
$ git checkout -b featureBv2 origin/master
$ git merge --no-commit --squash featureB
$ (change implementation)
$ git commit
$ git push myfork featureBv2
```



Maintaining a Project - Combining Works - git apply, git am commands

- Working in Topic Branches

```
$ git branch sc/ruby_client master
```

- Applying Patches with apply

```
$ git apply /tmp/patch-ruby-client.patch - "apply all or abort all"
$ git apply --check 0001-seeing-if-this-helps-the-gem.patch
```

- Applying a Patch with am

0001-limit-log-function.patch

```
From 330090432754092d704da8e76ca5c05c198e71a8 Mon Sep 17 00:00:00 2001
From: Jessica Smith <jessica@example.com>
Date: Sun, 6 Apr 2008 10:17:23 -0700
Subject: [PATCH 1/2] add limit to log function

Limit log functionality to the first 20
```

```
$ git am 0001-limit-log-function.patch
Applying: add limit to log function

$ git log --pretty=fuller -1
commit 6c5e70b984a60b3cecd395edd5b48a7575bf58e0
Author:     Jessica Smith <jessica@example.com>
AuthorDate: Sun Apr 6 10:17:23 2008 -0700
Commit:     Scott Chacon <schacon@gmail.com>
CommitDate: Thu Apr 9 09:19:06 2009 -0700

   add limit to log function

   Limit log functionality to the first 20
```

Maintaining a Project - Viewing Differences between Branches

- `git log contrib --not master`

```
commit 5b6235bd297351589efc4d73316f0a68d484f118
Author: Scott Chacon <schacon@gmail.com>
Date:   Fri Oct 24 09:53:59 2008 -0700

    seeing if this helps the gem

commit 7482e0d16d04bea79d0dba8988cc78df655f16a0
Author: Scott Chacon <schacon@gmail.com>
Date:   Mon Oct 22 19:38:36 2008 -0700

    updated the gemspec to hopefully work better
```

- You should carefully use the command - 'git diff'.

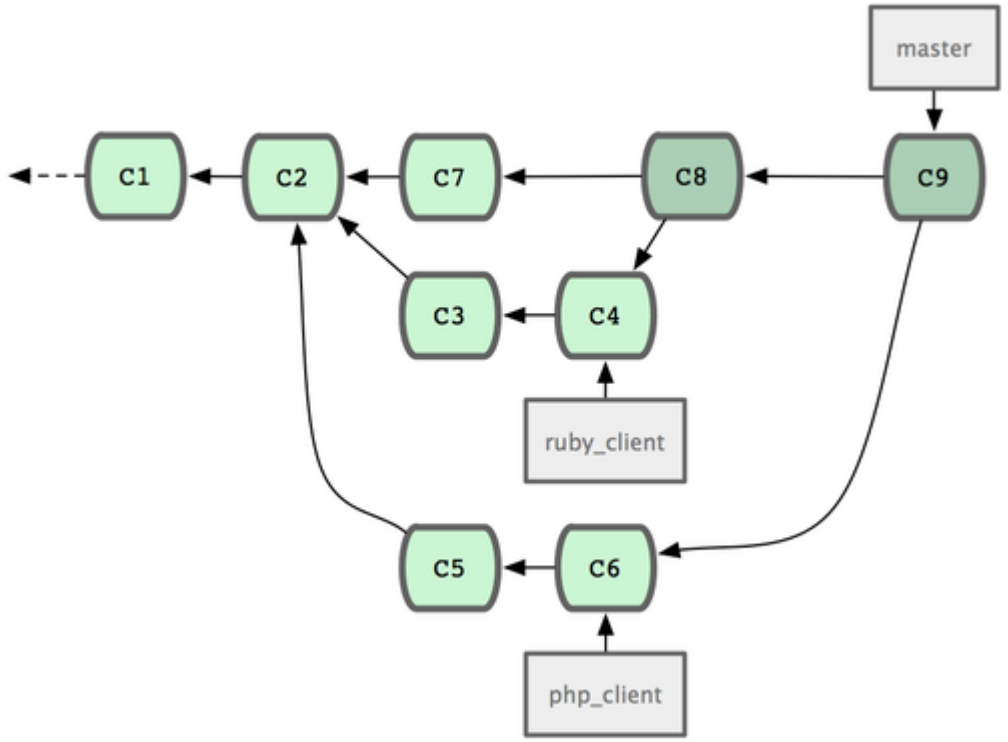
  $ git diff master - If `master` is a direct ancestor of your topic branch, this isn't a problem; but if the two histories have diverged, the diff will look like you're adding all the new stuff in your topic branch and removing everything unique to the `master` branch.

```
$ git merge-base contrib master
36c7dba2c95e6bbb78dfa822519ecfec6e1ca649
$ git diff 36c7db
```

```
$ git diff master...contrib
```

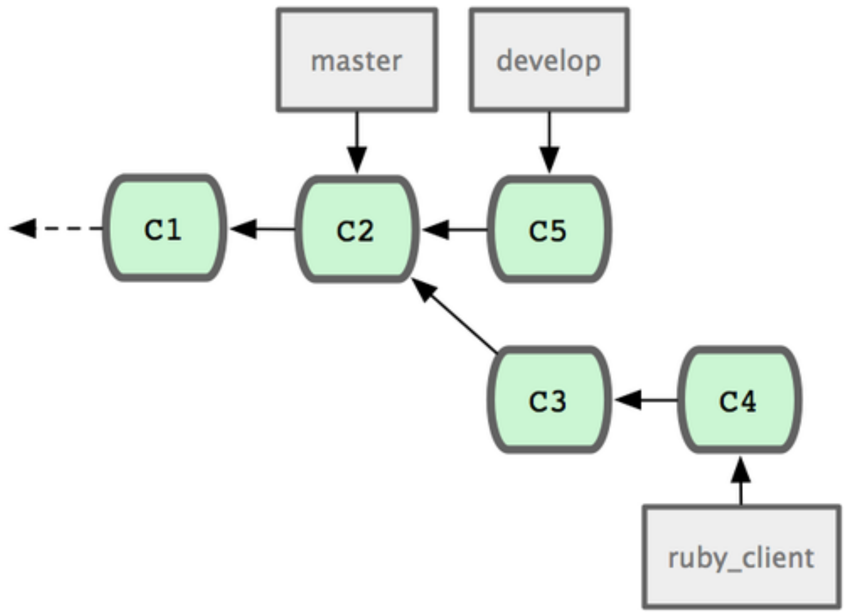
Maintaining a Project - Combining Works - Simple Workflow

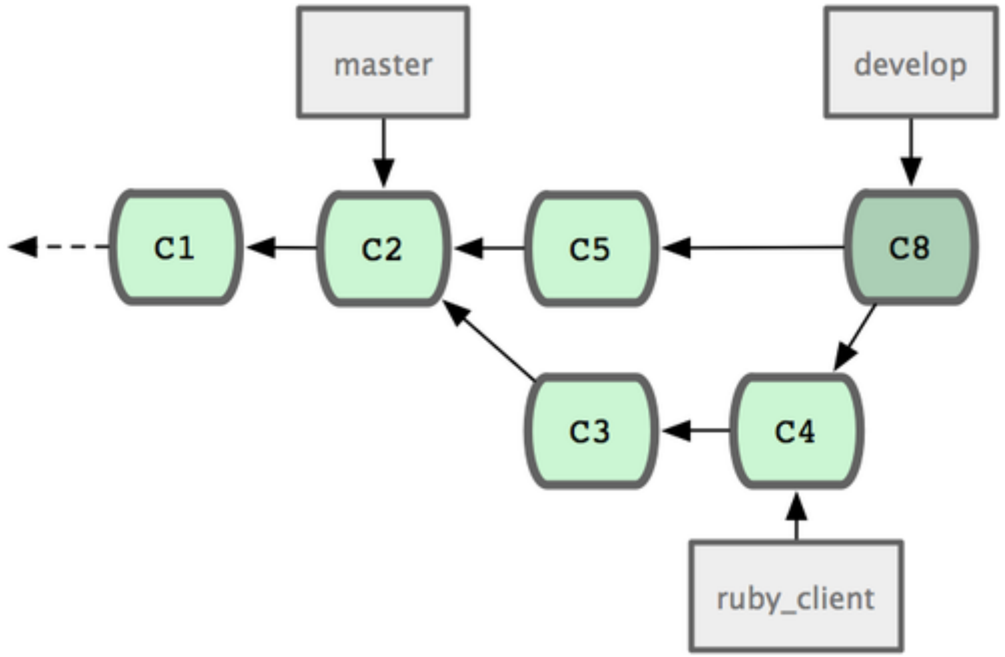
- `One simple workflow merges your work into your master branch.`

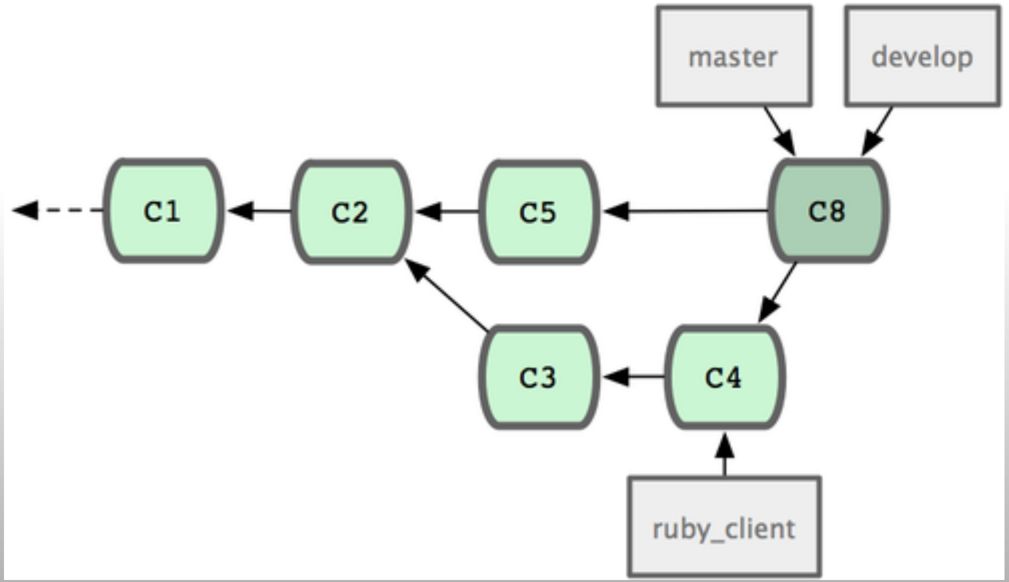


Maintaining a Project - Combining Works - Two-phase Merge Cycle

- `Two-phase merge cycle`



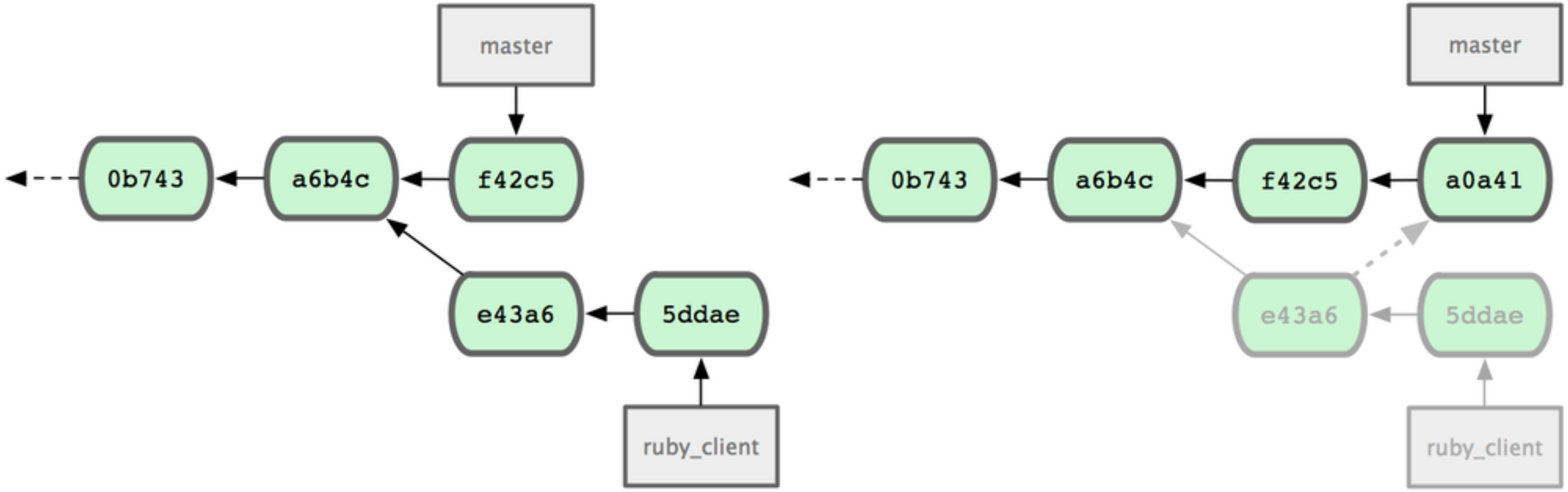




Maintaining a Project - Combining Works - Cherry Picking

- **Cherry Picking** Workflows

```
$ git cherry-pick e43a6fd3e94888d76779ad79fb568ed180e5fcdf
Finished one cherry-pick.
[master]: created a0a41a9: "More friendly message when locking the index fails."
 3 files changed, 17 insertions(+), 3 deletions(-)
```



Maintaining a Project - Tagging, Maintainer Public Key

- Tagging Your Releases
```
$ git tag -s v1.5 -m 'my signed 1.5 tag'
You need a passphrase to unlock the secret key for
user: "Scott Chacon <schacon@gmail.com>"
1024-bit DSA key, ID F721C45A, created 2009-02-09
```

- How to make the public key as a blob in the repository
```
$ gpg --list-keys
/Users/schacon/.gnupg/pubring.gpg
---------------------------------
pub   1024D/F721C45A 2009-02-09 [expires: 2010-02-09]
uid                  Scott Chacon <schacon@gmail.com>
sub   2048g/45D02282 2009-02-09 [expires: 2010-02-09]
```

  Exports the key info and writes a new blob with them into Git
```
$ gpg -a --export F721C45A | git hash-object -w --stdin
659ef797d181633c87ec71ac3f9ba29fe5775b92
```

```
$ git tag -a maintainer-pgp-pub 659ef797d181633c87ec71ac3f9ba29fe5775b92
```

  To share this key
```
$git push --tags
```

  The users can import the key by pulling the blob directly .
```
$ git show maintainer-pgp-pub | gpg --import
```

Maintaining a Project - Building Version Number and Source Tarball

- Generating a Build Number

  ```
  $ git describe master
  v1.6.2-rc1-20-g8c5b85c
  ```

  The `git describe` command favors annotated tags (which are created with the `-a` or `-s` flag).

- Preparing a Release

  ```
  $ git archive master --prefix='project/' | gzip > `git describe master`.tar.gz
  $ ls *.tar.gz
  v1.6.2-rc1-20-g8c5b85c.tar.gz

  $ git archive master --prefix='project/' --format=zip > `git describe master`.zip
  ```

- Shortlog

  ```
  $ git shortlog --no-merges master --not v1.0.1
  Chris Wanstrath (8):
        Add support for annotated tags to Grit::Tag
        Add packed-refs annotated tag support.
        Add Grit::Commit#to_patch
        Update version and History.txt
        Remove stray `puts`
        Make ls_tree ignore nils

  Tom Preston-Werner (4):
        fix dates in history
        dynamic version method
        Version bump to 1.0.2
        Regenerated gemspec for version 1.0.2
  ```

Git Tools - Revision Selection

- Git is smart enough to figure out what commit you meant to type as long as your partial SHA-1 is at least four characters long and unambiguous.
```
$ git show 1c002dd4b536e7479fe34593e72e6c6c1819e53b
$ git show 1c002dd4b536e7479f
$ git show 1c002d
```

- `git log --abbrev-commit --pretty=oneline`

- `git rev-parse topic1`
  `ca82a6dff817ec66f44342007202690a93763949`

- reflog - a log of where your HEAD and branch references have been for the last few months.
```
$ git reflog
734713b... HEAD@{0}: commit: fixed refs handling, added gc auto, updated
d921970... HEAD@{1}: merge phedders/rdocs: Merge made by recursive.
1c002dd... HEAD@{2}: commit: added some blame and merge stuff

$ git show HEAD@{5}
```

- To see where your `master` branch was yesterday
```
$ git show master@{yesterday}
```

- To see reflog information formatted like the `git log` output
```
$ git log -g master
commit 9ddfbdf5264aa8e44cedf8f0bf09887efc35a7dd
Reflog: master@{0} (seungzzang <seungzzang@gmail.com>)
Reflog message: commit (merge): Kruskal vertexSet test
Author: seungzzang <seungzzang@gmail.com>
Date:   Sat Jun 1 22:36:09 2013 +0900

    Kruskal vertexSet test
```

- **The reflog information is strictly local.**

Git Tools - Revision Selection

- Ancestry References

```
$ git log --pretty=format:'%h %s' --graph
* 734713b fixed refs handling, added gc auto, updated tests
*   d921970 Merge commit 'phedders/rdocs'
|\
| * 35cfb2b Some rdoc changes
* | 1c002dd added some blame and merge stuff
|/
* 1c36188 ignore *.gem
* 9b29157 add open3_detach to gemspec file list
```

```
$ git show HEAD^
commit d921970aadf03b3cf0e71becdaab3147ba71cdef
...
```

```
$ git show d921970^
commit 1c002dd4b536e7479fe34593e72e6c6c1819e53b
...
```

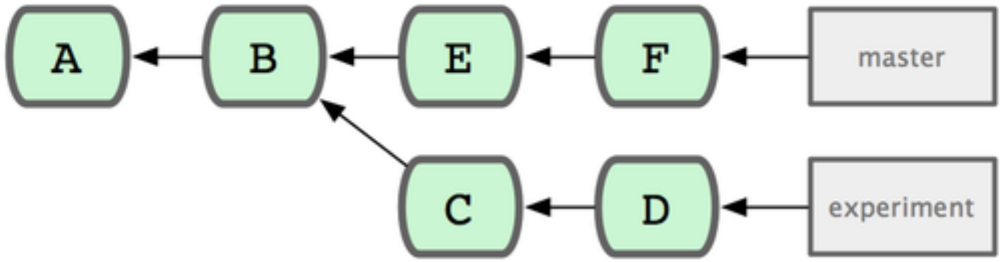
```
$ git show d921970^2
commit 35cfb2b795a55793d7cc56a6cc2060b4bb732548
...
```

```
$ git show HEAD~3 = $ git show HEAD^^^
commit 1c3618887afb5fbcbea25b7c013f4e2114448b8d
...
```

```
$ git show HEAD~3^2
```

Git Tools - Revision Selection

- Commit Range
  What work is on this branch that I haven't yet merged into my main branch?



```
$ git log master..experiment
D
C

$ git log experiment..master
F
E
```

```
$ git log master...
experiment
F
E
D
C
```

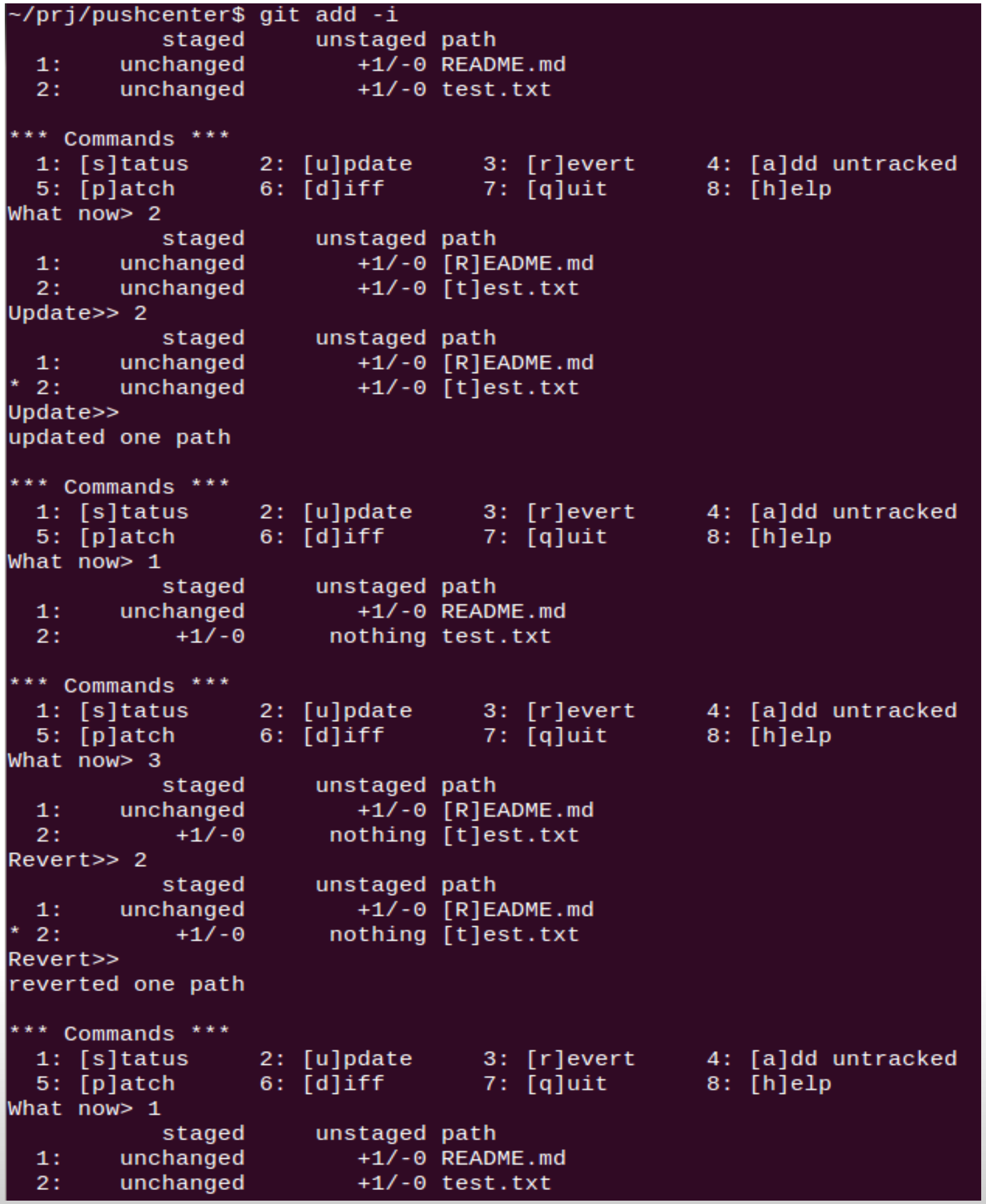
```
$ git log --left-right master...
experiment
< F
< E
> D
> C
```

- To see what you're about to push to a remote
  `git log origin/master..HEAD = git log origin/master..`

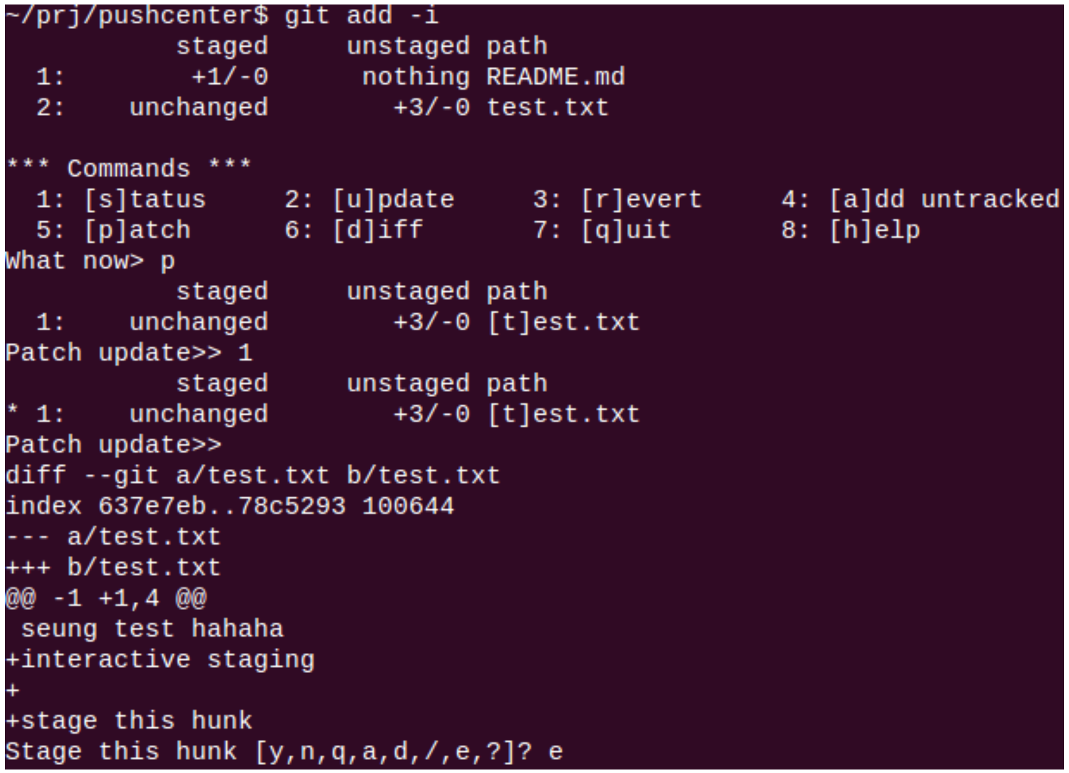
- Multiple points

```
git log refA..refB
git log ^refA refB
git log refB --not refA

git log refA refB ^refC
git log refA refB --not refC
```

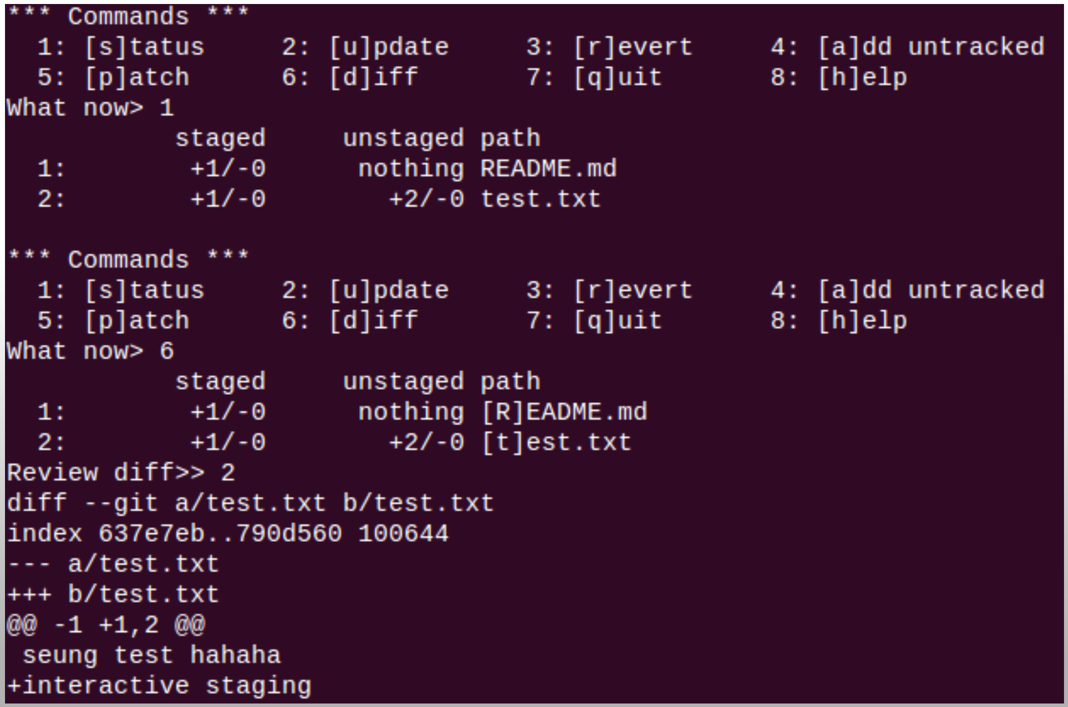
Git Tools - Interactive Staging



```
~/prj/pushcenter$ git add -i
           staged     unstaged path
  1:    unchanged        +1/-0 README.md
  2:    unchanged        +1/-0 test.txt

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> 2
           staged     unstaged path
  1:    unchanged        +1/-0 [R]EADME.md
  2:    unchanged        +1/-0 [t]est.txt
Update>> 2
           staged     unstaged path
  1:    unchanged        +1/-0 [R]EADME.md
* 2:    unchanged        +1/-0 [t]est.txt
Update>>
updated one path

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> 1
           staged     unstaged path
  1:    unchanged        +1/-0 README.md
  2:        +1/-0      nothing test.txt

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> 3
           staged     unstaged path
  1:    unchanged        +1/-0 [R]EADME.md
  2:        +1/-0      nothing [t]est.txt
Revert>> 2
           staged     unstaged path
  1:    unchanged        +1/-0 [R]EADME.md
* 2:        +1/-0      nothing [t]est.txt
Revert>>
reverted one path

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> 1
           staged     unstaged path
  1:    unchanged        +1/-0 README.md
  2:    unchanged        +1/-0 test.txt
```



```
~/prj/pushcenter$ git add -i
           staged     unstaged path
  1:        +1/-0      nothing README.md
  2:    unchanged        +3/-0 test.txt

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> p
           staged     unstaged path
  1:    unchanged        +3/-0 [t]est.txt
Patch update>> 1
           staged     unstaged path
* 1:    unchanged        +3/-0 [t]est.txt
Patch update>>
diff --git a/test.txt b/test.txt
index 637e7eb..78c5293 100644
--- a/test.txt
+++ b/test.txt
@@ -1 +1,4 @@
 seung test hahaha
+interactive staging
+
+stage this hunk
Stage this hunk [y,n,q,a,d,/,e,?]? e
```



```
*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> 1
           staged     unstaged path
  1:        +1/-0      nothing README.md
  2:        +1/-0        +2/-0 test.txt

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> 6
           staged     unstaged path
  1:        +1/-0      nothing [R]EADME.md
  2:        +1/-0        +2/-0 [t]est.txt
Review diff>> 2
diff --git a/test.txt b/test.txt
index 637e7eb..790d560 100644
--- a/test.txt
+++ b/test.txt
@@ -1 +1,2 @@
 seung test hahaha
+interactive staging
```

Git Tools - Stashing

- Stashing takes your modified tracked files and staged changes and saves it on a stack of unfinished changes.

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#      modified:   index.html
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#      modified:   lib/simplegit.rb

$ git stash
...
$ git status
# On branch master
nothing to commit (working directory clean)

$ git stash list
stash@{0}: WIP on master: 049d078 added the index file
stash@{1}: WIP on master: c264051... Revert "added file_size"
```

```
$ git stash apply stash@{0}
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be
committed)
#
#      modified:   index.html
#      modified:   lib/simplegit.rb
#

$ git stash drop stash@{0}
```

```
$ git stash apply --index
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#      modified:   index.html
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be
committed)
#
#      modified:   lib/simplegit.rb
```

Git Tools - Stashing

- Un-applying a Stash
```
$ git stash show -p stash@{0} | git apply -R
$ git stash show -p | git apply -R

$ git config --global alias.stash-unapply '!git stash show -p | git apply -R'
$ git stash
...
$ git stash-unapply
```

- Creating a Branch from a Stash
  git stash branch creates a new branch for you,
  checks out the commit you were on when you stashed your work,
  reapplies your work there, and then drops the stash if it applies successfully:

```
$ git stash branch testchanges
Switched to a new branch "testchanges"
# On branch testchanges
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#      modified:   index.html
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#      modified:   lib/simplegit.rb
#
Dropped refs/stash@{0} (f0dfc4d5dc332d1cee34a634182e168c4efc3359)
```

Git Tools - Rewriting History

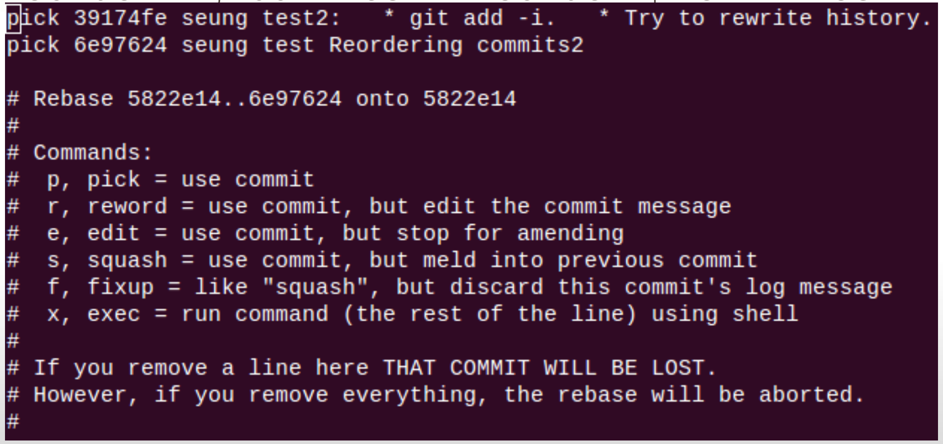
- One of the great things about Git is that it allows you to make decisions at the last possible moment.

- Changing the Last Commit
  - git add or git rm ...
    git commit --amend takes your current staging area and makes it the snapshot for the new commit.

  - **Amending changes the SHA-1 of the commit.**
    Don't amend your last commit if you've already pushed it.

- Changing Multiple Commits
  $ git rebase -i HEAD~3
  Every commit included in the range HEAD~3..HEAD will be rewritten.
  Don't include any commit you've already pushed to a central server.

- Reordering Commits - reorder pick lines



```
pick 39174fe seung test2:   * git add -i.   * Try to rewrite history.
pick 6e97624 seung test Reordering commits2

# Rebase 5822e14..6e97624 onto 5822e14
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Git Tools - Rewriting History

- `Splitting a Commit`

  pick **f7f3f6d** changed my name a bit
  edit 310154e updated README formatting and added blame
  pick **a5f4a0d** added cat-file

  ```
  $ git reset HEAD^
  $ git add README
  $ git commit -m 'updated README formatting'
  $ git add lib/simplegit.rb
  $ git commit -m 'added blame'
  $ git rebase --continue
  ```

  ```
  $ git log -4 --pretty=format:"%h %s"
  1c002dd added cat-file
  9b29157 added blame
  35cfb2b updated README formatting
  f3cc40e changed my name a bit
  ```

- filter-branch

  ```
  git filter-branch --tree-filter "rm -f *~" HEAD
  ```

- Changing E-Mail Address Globally

  ```
  $ git filter-branch --commit-filter '
          if [ "$GIT_AUTHOR_EMAIL" = "schacon@localhost" ];
          then
                  GIT_AUTHOR_NAME="Scott Chacon";
                  GIT_AUTHOR_EMAIL="schacon@example.com";
                  git commit-tree "$@";
          else
                  git commit-tree "$@";
          fi' HEAD
  ```

Git Tools - Debugging with Git - blame

- Blame tells when each line of the method was last edited and by whom.

```
$ git blame -L 12,22 simplegit.rb
^4832fe2 (Scott Chacon  2008-03-15 10:31:28 -0700 12)  def show(tree = 'master')
^4832fe2 (Scott Chacon  2008-03-15 10:31:28 -0700 13)   command("git show #{tree}")
^4832fe2 (Scott Chacon  2008-03-15 10:31:28 -0700 14)  end
^4832fe2 (Scott Chacon  2008-03-15 10:31:28 -0700 15)
9f6560e4 (Scott Chacon  2008-03-17 21:52:20 -0700 16)  def log(tree = 'master')
79eaf55d (Scott Chacon  2008-04-06 10:15:08 -0700 17)   command("git log #{tree}")
9f6560e4 (Scott Chacon  2008-03-17 21:52:20 -0700 18)  end
9f6560e4 (Scott Chacon  2008-03-17 21:52:20 -0700 19)
42cf2861 (Magnus Chacon 2008-04-13 10:45:01 -0700 20)  def blame(path)
42cf2861 (Magnus Chacon 2008-04-13 10:45:01 -0700 21)   command("git blame #{path}")
42cf2861 (Magnus Chacon 2008-04-13 10:45:01 -0700 22)  end
```

  ^4832fe2 commit lines designate that those lines were in this file's original commit. That commit is when this file was first added to this project, and those lines have been unchanged since.

- -C option makes blame to figure out where snippets of code within it originally came from if they were copied from elsewhere.

```
$ git blame -C -L 141,153 GITPackUpload.m
f344f58d GITServerHandler.m (Scott 2009-01-04 141)
f344f58d GITServerHandler.m (Scott 2009-01-04 142) - (void) gatherObjectShasFromC
f344f58d GITServerHandler.m (Scott 2009-01-04 143) {
70befddd GITServerHandler.m (Scott 2009-03-22 144)         //NSLog(@"GATHER COMMI
ad11ac80 GITPackUpload.m    (Scott 2009-03-24 145)
ad11ac80 GITPackUpload.m    (Scott 2009-03-24 146)         NSString *parentSha;
ad11ac80 GITPackUpload.m    (Scott 2009-03-24 147)         GITCommit *commit = [g
ad11ac80 GITPackUpload.m    (Scott 2009-03-24 148)
ad11ac80 GITPackUpload.m    (Scott 2009-03-24 149)         //NSLog(@"GATHER COMMI
ad11ac80 GITPackUpload.m    (Scott 2009-03-24 150)
56ef2caf GITServerHandler.m (Scott 2009-01-05 151)         if(commit) {
56ef2caf GITServerHandler.m (Scott 2009-01-05 152)                 [refDict setOb
56ef2caf GITServerHandler.m (Scott 2009-01-05 153)
```

  This is really useful - You get the original commit where you copied the code over, because that is the first time you touched those lines in this file.

Git Tools - Debugging with Git - bisect

- Binary Search

```
$ git bisect start
$ git bisect bad
$ git bisect good v1.0
Bisecting: 6 revisions left to test after this
[ecb6e1bc347ccecc5f9350d878ce677feb13d3b2] error handling on repo

$ git bisect good
Bisecting: 3 revisions left to test after this
[b047b02ea83310a70fd603dc8cd7a6cd13d15c04] secure this thing

$ git bisect bad
Bisecting: 1 revisions left to test after this
[f71ce38690acf49c1f3c9bea38e09d82a5ce6014] drop exceptions table

$ git bisect good
b047b02ea83310a70fd603dc8cd7a6cd13d15c04 is first bad commit
commit b047b02ea83310a70fd603dc8cd7a6cd13d15c04
Author: PJ Hyett <pjhyett@example.com>
Date:   Tue Jan 27 14:48:32 2009 -0800

    secure this thing

:040000 040000 40ee3e7821b895e52c1695092db9bdc4c61d1730
f24d3c6ebcfc639b1a3814550e62d60b8e68a8e4 M  config

$ git bisect reset : finish binary search and move to HEAD.

$ git bisect start HEAD v1.0
$ git bisect run test-error.sh
```

Git Tools - Submodules

- Starting with Submodules

```
$ git submodule add git://github.com/chneukirchen/rack.git rack
Initialized empty Git repository in /opt/subtest/rack/.git/
...
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#      new file:   .gitmodules
#      new file:   rack

$ cat .gitmodules
[submodule "rack"]
      path = rack
      url = git://github.com/chneukirchen/rack.git

$ git diff --cached rack
diff --git a/rack b/rack
new file mode 160000
index 0000000..08d709f
--- /dev/null
+++ b/rack
@@ -0,0 +1 @@
+Subproject commit 08d709f78b8c5b0fbeb7821e37fa53e69afcf433

$ git commit -m 'first commit with submodule rack'
[master 0550271] first commit with submodule rack
 2 files changed, 4 insertions(+), 0 deletions(-)
 create mode 100644 .gitmodules
 create mode 160000 rack
```

Git Tools - Submodules

- Cloning a Project with Submodules

```
$ git clone git://github.com/schacon/myproject.git
Initialized empty Git repository in /opt/myproject/.git/
...
$ cd myproject
$ ls -l
total 8
-rw-r--r--  1 schacon  admin   3 Apr  9 09:11 README
drwxr-xr-x  2 schacon  admin  68 Apr  9 09:11 rack
$ ls rack/
$
$ git submodule init
Submodule 'rack' (git://github.com/chneukirchen/rack.git) registered for path 'rack'
$ git submodule update

$ git merge origin/master
Updating 0550271..85a3eee
Fast forward
 rack |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

$git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   rack

$ git diff
diff --git a/rack b/rack
index 6c5e70b..08d709f 160000
--- a/rack
+++ b/rack
@@ -1 +1 @@
-Subproject commit 6c5e70b984a60b3cecd395edd5b48a7575bf58e0
+Subproject commit 08d709f78b8c5b0fbeb7821e37fa53e69afcf433
```

```
$ git submodule update
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
...
```

Git Tools - Submodule Issues

- **One common problem** happens when a developer makes a change locally in a submodule but doesn't push it to a public server.

  He commit a pointer to that non-public state and push up the superproject. When other developers try to run `git submodule update`,

  ```
  $ git submodule update
  fatal: reference isn't a tree: 6c5e70b984a60b3cecd395edd5b48a7575bf58e0
  Unable to checkout '6c5e70b984a60b3cecd395edd5ba7575bf58e0' in submodule path 'rack'
  ```

- You do an initial `submodule update`, commit in that submodule directory, and then run `git submodule update` again from the superproject. Git will overwrite your changes without telling you. -> **Avoid this!**

- Switching from subdirectories to submodules

  ```
  $ rm -Rf rack/
  $ git submodule add git@github.com:schacon/rack.git rack
  'rack' already exists in the index

  $ git rm -r rack
  $ git submodule add git@github.com:schacon/rack.git rack
  Initialized empty Git repository in /opt/testsub/rack/.git/
  remote: Counting objects: 3184, done.
  remote: Compressing objects: 100% (1465/1465), done.
  ...
  ```

- If you try to switch back to a branch where those files are still in the actual tree rather than a submodule - you get this error.

  ```
  $ git checkout master
  error: Untracked working tree file 'rack/AUTHORS' would be overwritten by merge.

  $ mv rack /tmp/
  $ git checkout master
  Switched to branch "master"
  ```

  when you switch back, You can either run `git submodule update` to reclone, or you can move your `/tmp/rack` directory back.

Git Tools - Subtree

- **Subtree** deals with the subproject issue.

  ```
  $ git remote add rack_remote git@github.com:schacon/rack.git
  $ git fetch rack_remote
  …
  $ git checkout -b rack_branch rack_remote/master
  Branch rack_branch set up to track remote branch refs/remotes/rack_remote/master.
  Switched to a new branch "rack_branch"
  ```

  To pull the rack project into the rack subdirectory of your master branch,

  ```
  $ git checkout master
  $ git read-tree --prefix=rack/ -u rack_branch
  ```

  The read-tree reads the root tree of one branch into your current staging area and working directory.

- Merge changes in rack_remote into rack_branch.

  ```
  $ git checkout rack_branch
  $ git pull
  ```

- Merge changes in rack_remote into master.

  ```
  $ git checkout master
  $ git merge --squash -s subtree --no-commit rack_branch
  ```

- To get a diff between rack subdirectories in master and rack_branch

  ```
  $ git diff-tree -p rack_branch
  ```

  To get a diff between rack subdirectories in master and rack_remote/master

  ```
  $ git diff-tree -p rack_remote/master
  ```

Git and Subversion

- Subversion can have only a single linear history.
  It's generally best to **keep your history as linear as possible by rebasing your work although you can easily do local branching and merging.**

- Working with Git-based remote versions of your projects concurrently with a Subversion server isn't a good idea.

  git svn dcommit rewrites your local Git commit to include a unique identifier.
  It means that **all the SHA-1 checksums for your commits change.**

  If you want to push to both a Git server and a Subversion server,
  you have to push (dcommit) to the Subversion server first, because that action changes your commit data.

- **If you push to a SVN server before you do 'git svn rebase', your project state can be messy.**

- **Recommanded Workflows**
  1. `git commit`
  2. `git svn rebase`
  3. `( git rebase --continue )`
  4. `git svn dcommit`

  or

  1. `git stash`
  2. `git svn rebase`
  3. `git stash apply`
  4. `git commit`
  5. `git svn dcommit`
  6. `git stash drop`

Git and Subversion

- **How to get svn history in order to use it in Git repo**
  `git svn clone svn://svnRepoServer/... -T trunk -r<revision>:HEAD local-dir`

- **Subversion has a linear history.**
  So, If you're pushing to a Subversion server via git svn, **It's better for you to rebase your work onto a single branch each time** instead of merging branches together.

- Creating a New SVN Branch
  `git svn branch [branchname]` = svn copy trunk branches/opera

- To commit into the new SVN Branch,
  `git branch opera remotes/opera`

- To merge new SVN Branch into master,
  `git merge opera`
  This merge likely will be much easier than it would be in Subversion.
  (Because Git will automatically detect the appropriate merge base for you)

- If you push the merge back to a SVN server, The merge information will be disapper
  Because of SVN characteristics.
  The dcommit makes your git merge result look like you ran git merge --squash.