Git

A free and open source distributed version control system

Chris Sakalis
chrissakalis@gmail.com
AUTH ACM Student Chapter

Introduction

BEFORE VCS

Before VCS people would:

- Send source code
  - Became impossible as projects grew larger
  - With the Internet, programmers are no longer in the same room
- Send patches
  - `patch` and `diff` command line tools
  - `diff` created by Douglas McIlroy
  - `patch` created by Larry Wall
  - The patches would usually be tarballed and emailed
  - Better than sharing the whole file, still impractical

VERSION CONTROL SYSTEMS

Three ”generations” of Version Control Systems

1. Simple, only one person could access the files at any given moment
   - RCS
2. Networked but still centralized
   - CVS
   - Subversion (SVN)
3. Networked and distributed
   - Git
   - Mercurial
   - Bazaar

WHY GIT?

- The Linux Kernel needed a new and free VCS
- Linus Torvalds didn't like anything else
- Git was designed in a weekend
- Design goals:
  - Do not be like CVS
  - Distributed
  - Safe against corruption / Secure
  - Fast

AS A RESULT, GIT:

- is distributed software configuration management (DSCM)
  - Each user has a complete copy of the repository
  - No need for a central infrastructure
- is free, licensed under the GPLv2 license
- is fast and supports very large projects
- is safe, the history and code are cryptographically authenticated
- is modular and uses existing protocols (ie HTTP, ssh, rsync)
- allows for low level control (many ways to break your repository)
- is used by about 30% of the developers

IMPLEMENTATION DETAILS

- The data are organized in a tree structure, similar to a filesystem
- Two main structures:
  1. Index or Cache
     - The currest state of the working directory
     - Contains all the changes to be committed
  2. Object database
     - Contains all the history and files
     - Includes blobs, trees, commits and tags
     - Each object has a SHA1 hash

FIRST TIME SETUP

- Set the user's name and email address
  - `git config --global user.name "Chris Sakalis"`
  - `git config --global user.email "chrissakalis@gmail.com"`
- All the user settings go in the file `.gitconfig` in the user's home directory

EXAMPLE .GITCONFIG

```
[user]
    name = Chris Sakalis
    email = chrissakalis@gmail.com
    signingkey = 0E476575
[color]
    ui = true
[push]
    default = simple
[alias]
    hist = log --pretty=format:\"%h %ad | %s%d [%an]\" --graph
    co = checkout
    br = branch
    st = status
[merge]
    tool = vimdiff
[mergetool]
    keepBackup = false
```

STARTING A GIT REPOSITORY

There are two ways to start using Git for a project:

- Create your own repository
  `git init`
- Clone an existing one
  `git clone <repo> [<path>]`

MAKING SOME CHANGES

The basic Git workflow is:

1. Edit the files
2. Check the status of your repository
   `git status` and `git diff`
3. Add the changes you want to commit in the index
   `git add <path>`
4. Commit the changes
   `git commit`
5. Optionally: Upload your changes
   `git push origin master`

GIT STATUS

`git status` displays the current status of the repository

- Changes in the index (to be committed)
- Changes not in the index (will not be committed)
- Untracked files (git does not care for those)
  - `.gitignore`

```
# On branch parallel
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   example/ref_impl/mymtree.cpp
#
# 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:   example/Makefile
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       example/res.txt
#       example/sum_dist.py
```

ADDING FILES IN THE INDEX

`git add` adds file in the index to be committed

- `git add <path>` : Add the path specified
- `git add -u` : Add all the files that are *already* tracked
- `git add -A` : Add *all* the files
- `git add --patch` : Interactive mode

ADDING FILES IN THE INDEX

```
# 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 d
#
#       modified:   A.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       B.txt
no changes added to commit (use "git add" and/or "git commit -a")
[chriss@optimus ~/example]% git add A.txt B.txt
[chriss@optimus ~/example]% git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   A.txt
#       new file:   B.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
```

COMMITTING THE CHANGES

`git commit` saves the changes in the index in a commit. The default editor will open and the user will be asked for a commit message.
`git commit --amend` changes the last commit message.

```
[chriss@optimus ~/example]% git commit
[master ffa487b] Some changes in A and the new file B
 2 files changed, 1 insertion(+)
 create mode 100644 B.txt
[chriss@optimus ~/example]% git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
nothing added to commit but untracked files present (use "git add" to track)
```

EXPLORING THE REPOSITORY

1. Commit history
   `git log`
2. Changesets
   `git diff` and `git show`
3. Commits and branches
   `git checkout`

CHANGELOG - HISTORY

`git log` will display all the commits in the current branch

```
commit ffa487bcf9c778ceeffd7e85b8f13351794983e4
Author: Chris Sakalis <chrissakalis@gmail.com>
Date:   Sun Mar 17 14:40:44 2013 +0200

    Some changes in A and the new file B

commit 2d333a31e5c69a63fe9bb655dc60ddaf42f16304
Author: Chris Sakalis <chrissakalis@gmail.com>
Date:   Sun Mar 17 14:26:24 2013 +0200

    Added A
```

- The first 7 letters of the SHA1 sum can identify the commit
- `git log --grep=<pattern>` will filter the commits by the given pattern

GETTING DIFFERENCES IN FILES

The `git diff` command will display differences between file versions as a patchset.

- `git diff [<path>]` : Changes relative to the index (not added for commit yet)
- `git diff --cached [<path>]` : Changes in the index (to be committed)
- `git diff <commit> [<commit>] [<path>]` : Changes between two commits. If omitted, the second commit default to HEAD

The `git show <commit>` command will display all the changes introduced in the given commit.

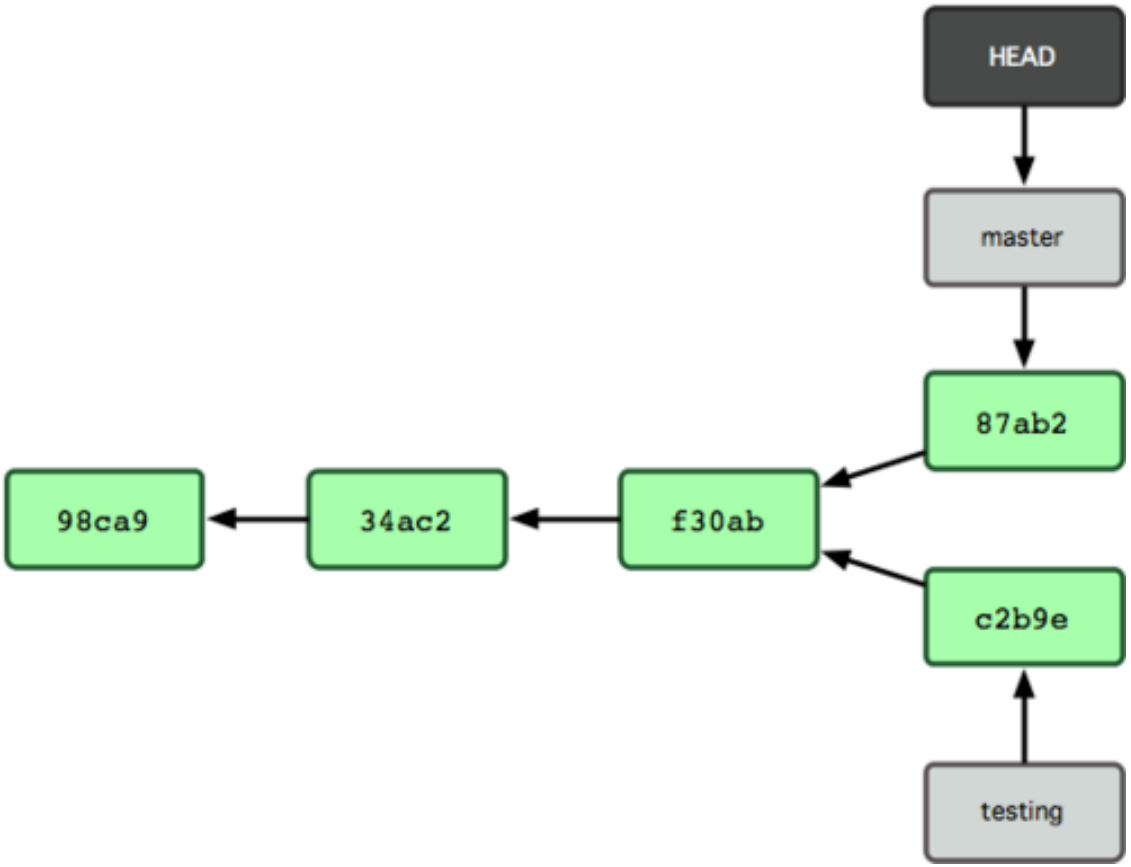
DIFF FORMAT

```
diff --git a/tpie/block_collection.inl b/tpie/block_collection.inl
index e3e1162..d52ca93 100644
--- a/tpie/block_collection.inl
+++ b/tpie/block_collection.inl
@@ -163,7 +163,7 @@ void block_collection<file_accessor_t>::delete_block(bid_t blockID) {
        // If it is the last block, just truncate the file;
        if (blockID == m_size - 1) {
                --m_size;
-               m_fileAccessor.truncate_i(m_size * m_blockSize);
+               m_fileAccessor.truncate_i(m_size * m_blockSize + header_size());
        } else {
                if (!m_stack) {
                        open_stack();
```

WORKING WITH BRANCHES

1. Managing branches
   `git branch`
2. Switching between branches
   `git checkout`
3. Merging branches
   `git merge` and `git rebase`

BRANCHES



Graphic taken from git-scm.com

GIT BRANCH

This command is used to create and delete branches.

- List all the local branches
  ```
  git branch
  ```
- Create a new branch
  ```
  git branch <branchname> [<start point>]
  git checkout -b <branchname> [<start pt>]
  ```
- Delete or rename a branch
  ```
  git branch (-d | -m)
  ```

GIT CHECKOUT

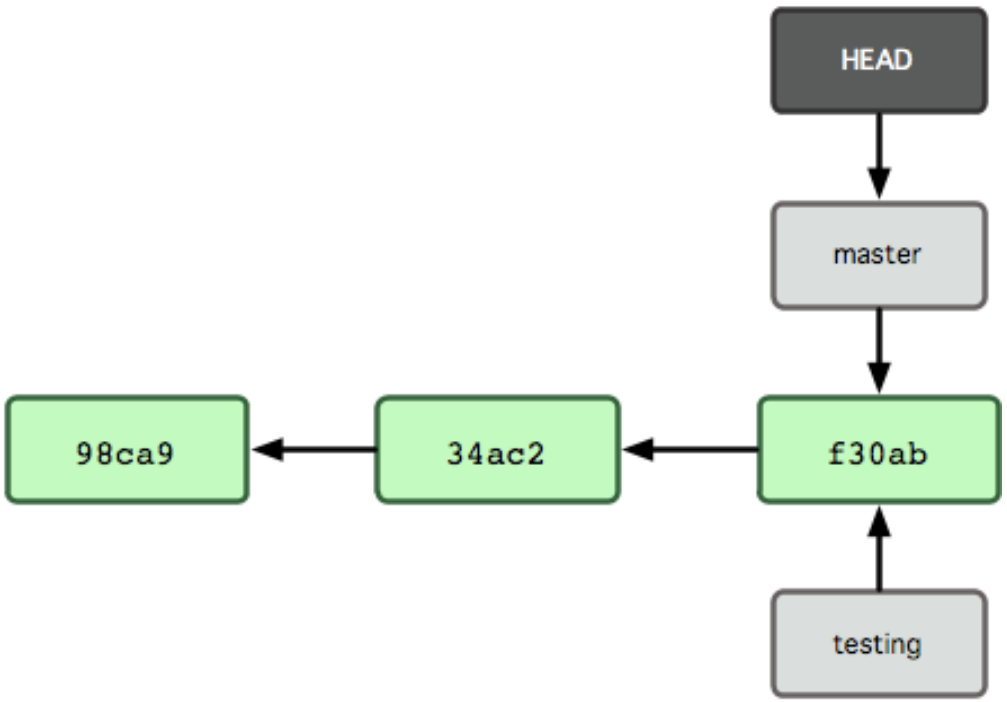
Change HEAD to the specified branch (or even commit)

- `git checkout <branchname>`
- `git checkout -b <branchname> [<start pt>]`

The `checkout` command is somewhat strange

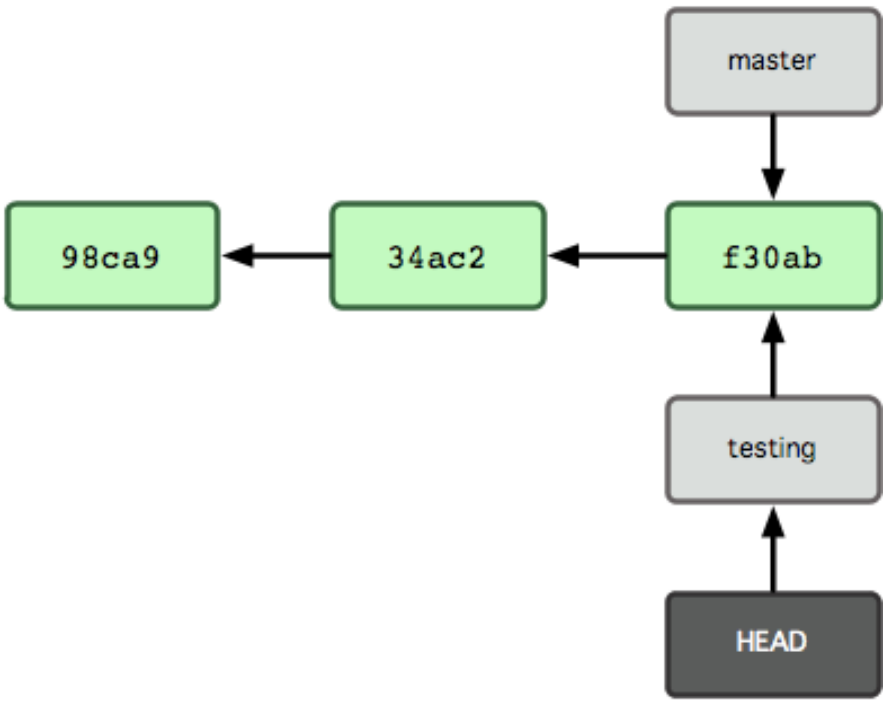
- `git checkout [<tree>] <path>`
- `git checkout -- <file>`

BRANCHES



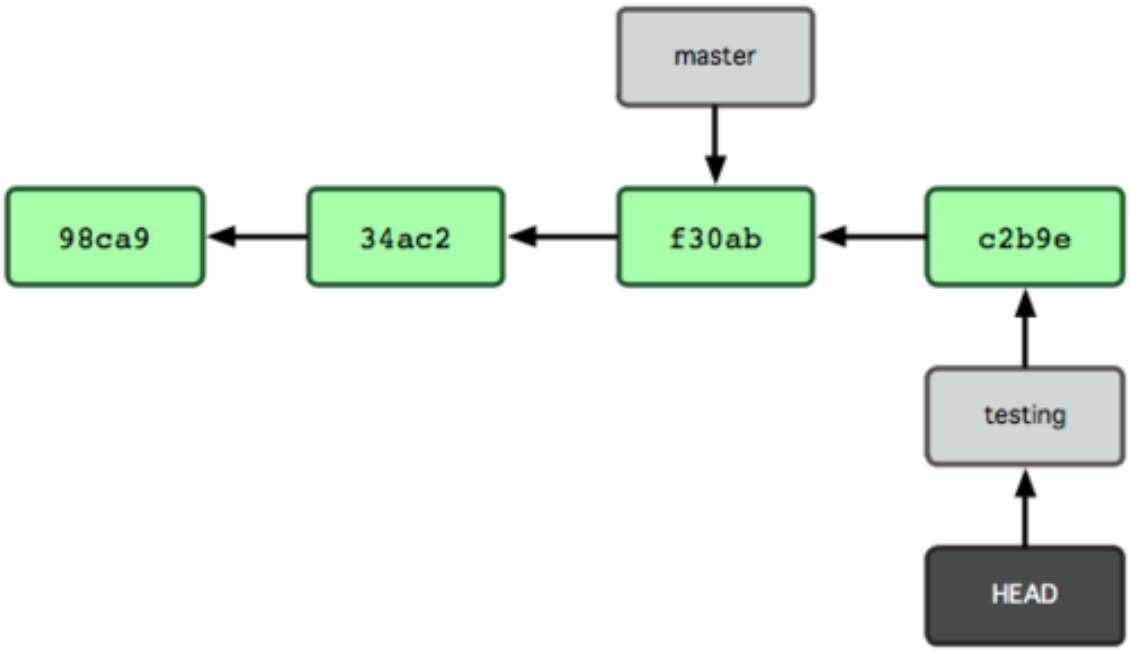
Graphic taken from git-scm.com

BRANCHES



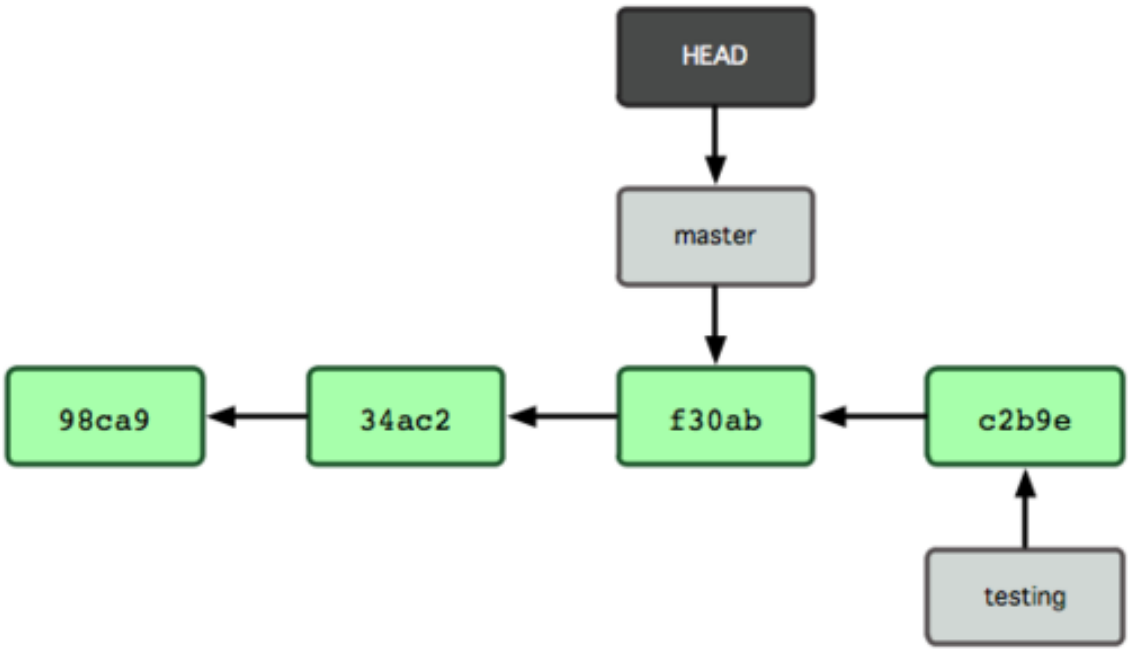
Graphic taken from git-scm.com

BRANCHES



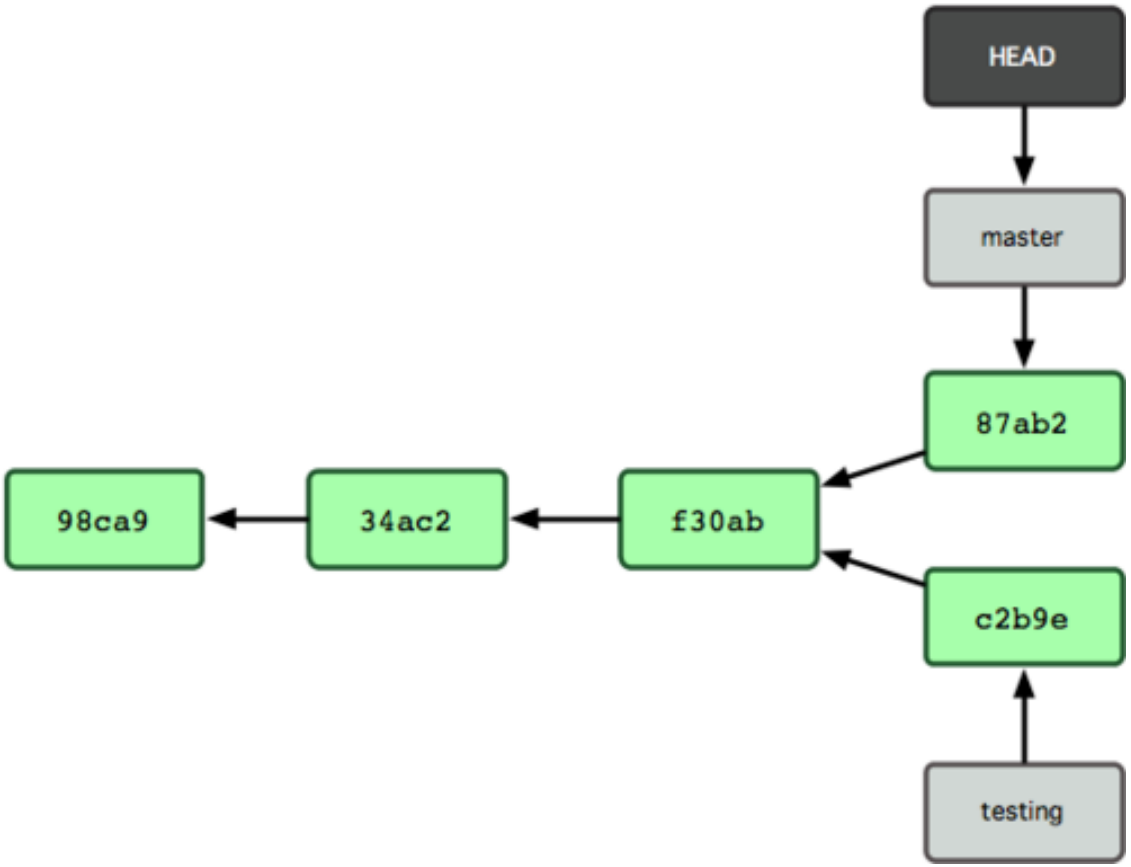
Graphic taken from git-scm.com

BRANCHES



Graphic taken from git-scm.com

BRANCHES

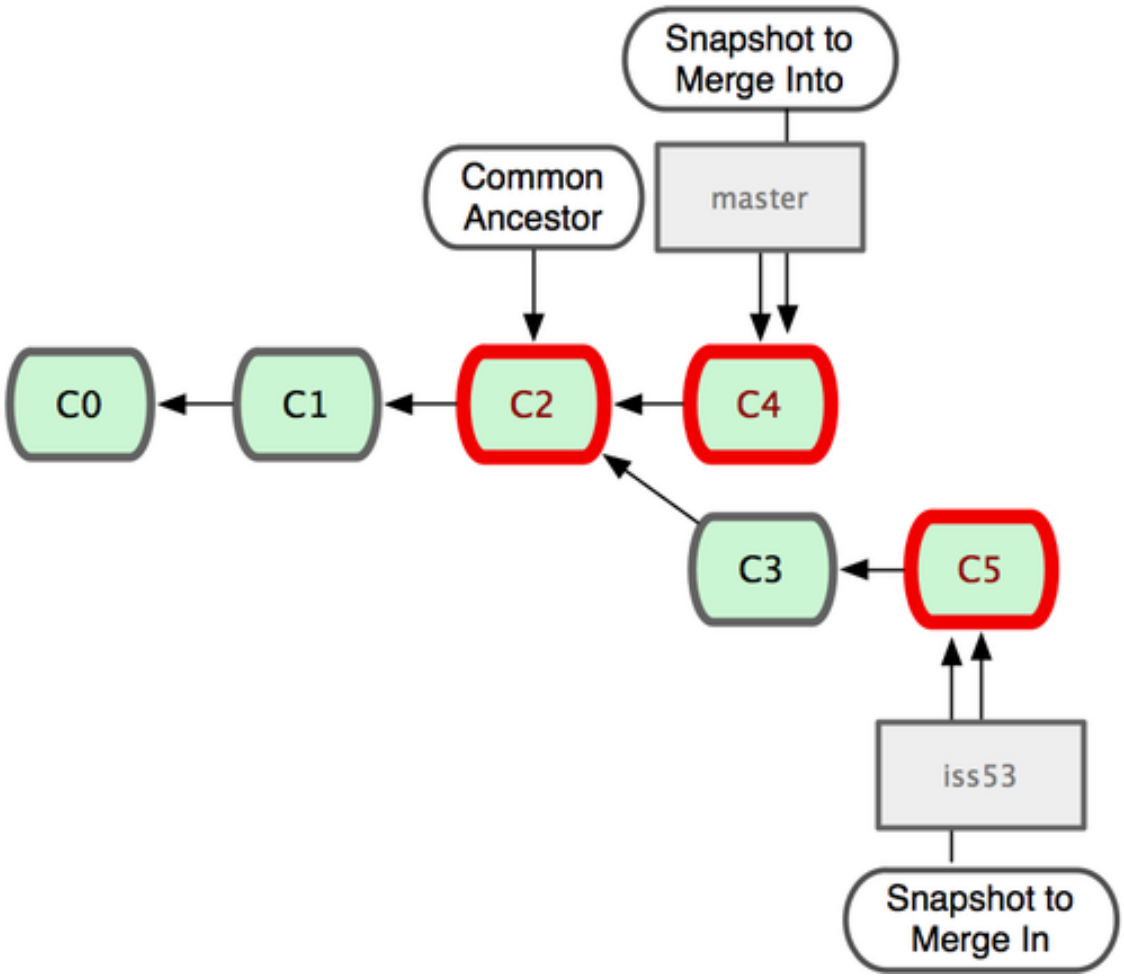


Graphic taken from git-scm.com

MERGING BRANCES

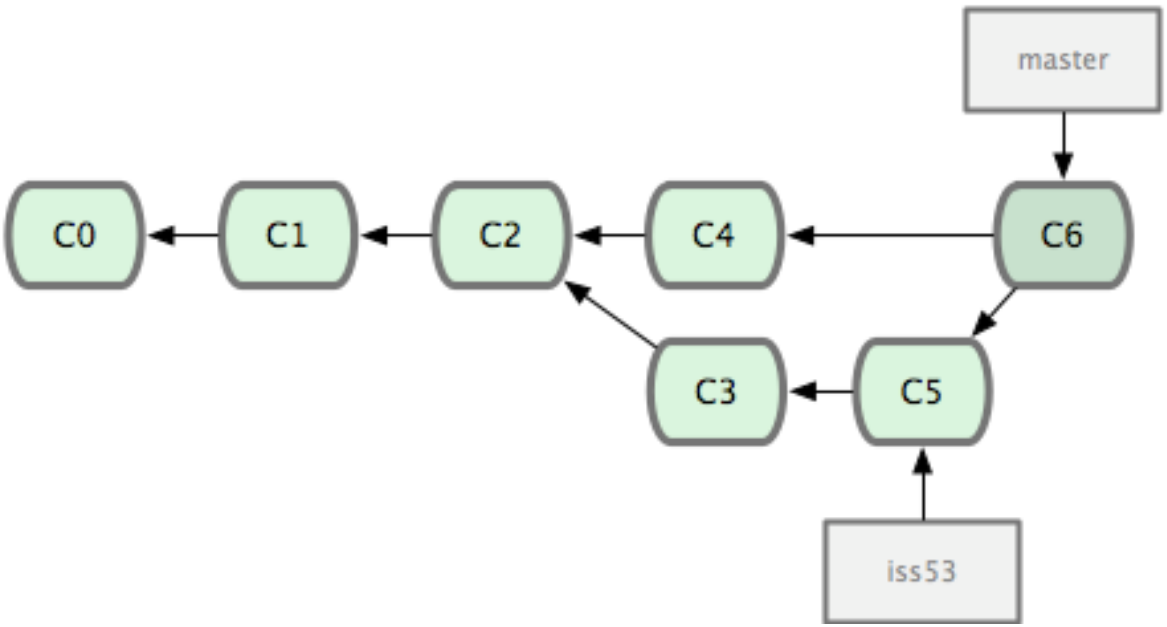
- Incorporate changes from the named commits into the current branch
  `git merge <commit>...`
- Undo a merge in progress (*always* commit before merging)
  `git merge --abort`
- Select only some commits to merge
  `git cherry-pick <commit>`
- `git rebase`

MERGING BRANCHES



Graphic taken from git-scm.com

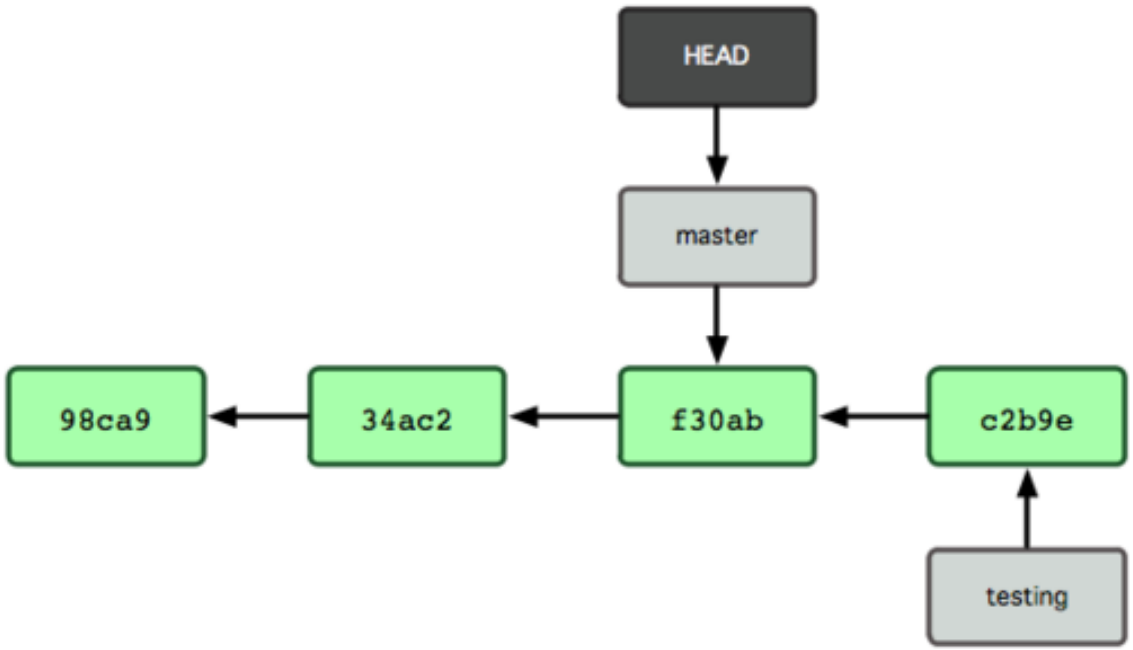
MERGING BRANCHES



Graphic taken from git-scm.com

FAST-FORWARD MERGE

When the current branch head is an ancestor of the named commit. No new commit needed, just update the HEAD.



Graphic taken from git-scm.com

MERGE CONFLICTS

When the branches have changed the same piece of code, conflicts arise.

```
 <<<<<<< local
changes made in your current branch
=======
changes made in the other branch
>>>>>>> remote
```

How to handle merge conflicts:

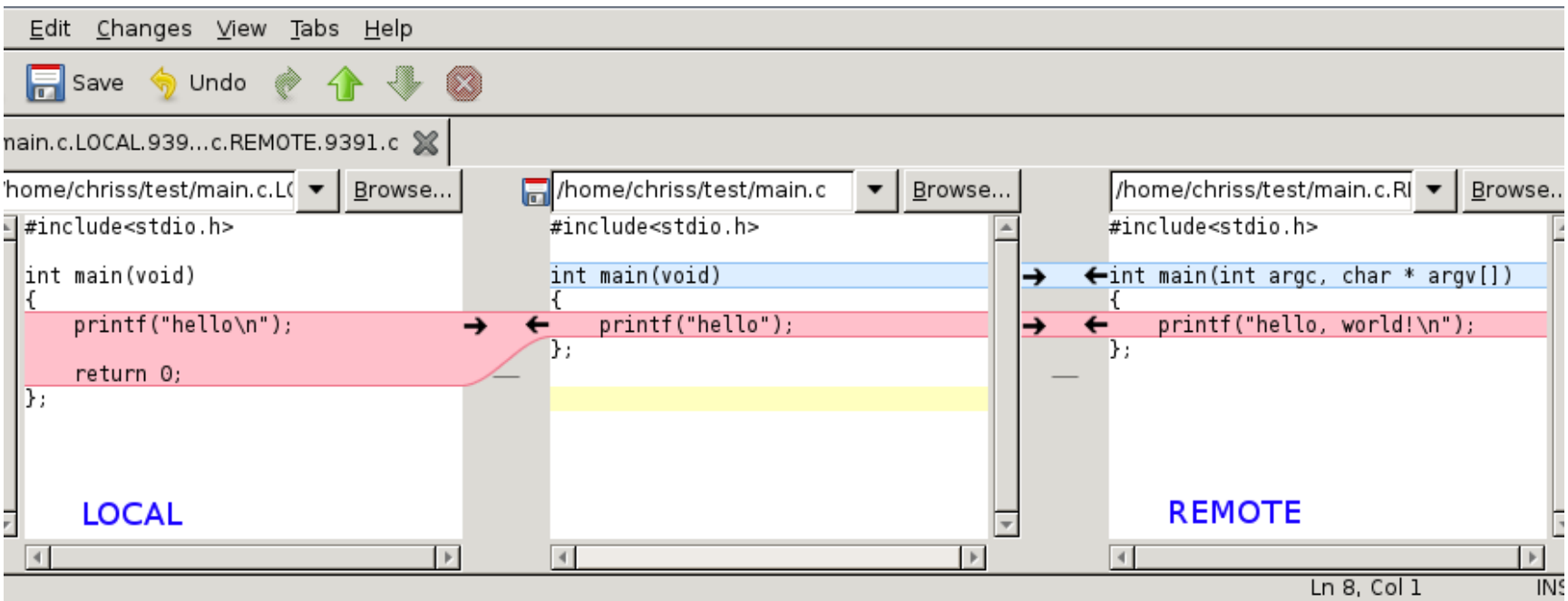
1. Resolve the conflicts
   - Doing so manually is usually very hard
   - Use a mergetool: `git mergetool`
2. Add the files to the index
3. Commit the resolved files

MERGETOOL - MELD

```
#include<stdio.h>

int main(int argc, char * argv[])
{
<<<<<<< HEAD
    printf("hello\n");

    return 0;
=======
    printf("hello, world!\n");
>>>>>>> testing
};
```



MERGE STRATEGIES

```
git merge -s <strategy> -X <strategy option>
```

- recursive: Default strategy when merging one branch
  - ours: Conflicts are autoresolved by favouring local version
  - theirs: Conflicts are autoresolved by favouring remote version
- octopus: Default with multiple branches
- ours: Ignore changes in the branch, the head remains the same

SHARING THE REPOSITORY

1. Pushing and pulling changes
   `git push` and `git fetch`
2. Remotes
   `git remote`

GIT FETCH AND PUSH

`git fetch` brings remote changes in a branch named `<repo>/<ref>`

- `git fetch <repo> [<ref>]`
- `git pull` = fetch and merge

`git push` sends your changes to another repository

- `git push [<repo>] [<ref>]`
- `git push [<repo>] :[<ref>]`

REMOTES

A git remote is a URL to a git repository in another location.
Various protocols are supported, like ssh and http.
The `git remote` command manages the remotes in the local repository. Changes made are not shared amongst different repositories.

GIT RESET

This command resets the current HEAD to the specified state

- `git reset <commit> <file>`: Opposite of `git add`
  - `git reset HEAD <file>`
- `git reset <commit>`
  - `--soft` : Change the HEAD, leave the index and files alone
  - `--mixed`: Change the HEAD and the index, leave the files alone
  - `--hard` : Change the HEAD, the index and the files as well

FINDING WHERE THE PROBLEM IS

`git bisect` runs a binary search through the commits.

1. Find a working commit
2. Find a broken commit
3. `git bisect start`
   `git bisect good <good commit>`
   `git bisect bad <bad commit>`
4. Check each commit presented and mark it as good or bad
   `git bisect (good | bad)`
5. After finding the faulty code, return to the normal git state
   `git bisect reset`

- `git bisect skip` skips the current commit
- `git bisect run` automates the whole process

TEMPORARY "COMMITS"

`git stash` hides any changes and reverts to a clean working directory

- `git stash` or `git stash save <message>`
- `git stash list`
- `git stash show`
- `git stash branch <branchname>`
- `git stash pop` and `git stash apply`
- `git stash drop` and `git stash clear`

MISCELLANEOUS

- `git tag <tag name> <commit>` adds a tag (label) to the commit
- `git reflog`
- `.git/hooks`
- `git filter-branch` changes the whole git history; useful to remove sensitive data accidentally committed
- Git GUIs
  - `git gui`
  - Netbeans, Intellij, Visual Studio → Native Git Plugins
  - Eclipse → EGit plugin
  - Vim → Fugitive
- git-scm.com
- `man git <command>` or Google