#### **Understanding Git**

Nelson Elhage Anders Kaseorg

Student Information Processing Board

October 21, 2008

#### Outline

- The Git model
- Using Git
- Collaboration with Git
- 4 Rewriting history
- 6 And beyond!

#### The Git model

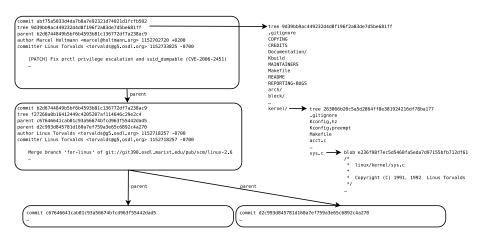
- A Git repository contains four kinds of objects.
- An object is either a *blob* (file), a *tree* (directory), a *commit* (revision), or a *tag*.
- Every object is uniquely identified by a 40 hex digit number, which is the SHA-1 hash of its contents.
  - Don't worry—identifiers can be abbreviated by truncation, or referenced with human-readable names.
- Some objects refer to other objects using their identifiers.

# **Objects**

- A blob object is a file's contents.
- A tree object is a directory—a list of zero or more directory entries, each of which has
  - a name
  - a UNIX mode
  - a tree id or blob id
- A commit object contains
  - a tree id
  - zero or more parents, which are commit ids
  - an author (name, email, date)
  - a committer (name, email, date)
  - a log message
- A tag object contains
  - a tag name
  - a tagger (name, email, date)
  - a reference to another object (usually a commit)
  - an optional log message



#### A commit



#### More commits

[PATCH] /fs/proc/: 'larger than buffer size' m Adam B. Jerome <abj@nov 2006-07-12 12:03:07 [PATCH] lockdep: annotate the sysfs i mute Arjan van de Ven <a href="mailto:arjan@2006-07-12">arjan@2006-07-12</a> 12:03:06 [PATCH] fix fdset leakage Kirill Korotaev <dev@open 2006-07-12 12:03:05 prctl [PATCH] Fix prctl privilege escal Marcel Holtmann <marcel 2006-07-12 07:12:00 Merge branch 'for-linus' of git://git390.osdl.n Linus Torvalds <torvalds@ 2006-07-12 11:30:57 [S390] Fix sparse warnings. Heiko Carstens < heiko.car | 2006-07-12 10:41:55 [S390] path grouping and path verifications Cornelia Huck < cornelia.ht 2006-07-12 10:40:19 [S390] xpram module parameter parsing. Heiko Carstens < heiko.car | 2006-07-12 10:40:14 [S390] cpu relax() is supposed to have barri Heiko Carstens < heiko.car 2006-07-12 10:39:58 [S390] fix futex atomic cmpxchg inatomic Martin Schwidefsky <schw | 2006-07-12 10:39:55 [S390] subchannel register/unregister mute: Cornelia Huck < cornelia.ht 2006-07-12 10:39:50 [S390] raw local save flags/raw local irq re Heiko Carstens <heiko.car 2006-07-12 10:39:47 [S390] builtin trap() and gcc version. Heiko Carstens <heiko.car | 2006-07-12 10:39:42 Add PIIX4 APCI guirk for the 440MX chipset toc Linus Torvalds < torvalds@ 2006-07-12 11:29:46 Merge branch 'splice' of git://brick.kernel.dk/da Linus Torvalds <torvalds@ 2006-07-12 11:14:48 [PATCH] splice: fix problems with sys tee() Jens Axboe <axboe@suse. 2006-07-10 05:00:01 x86 MacMini: make built-in speaker sound actul Linus Torvalds <torvalds@ 2006-07-11 01:21:43 Merge branch 'release' of git://git.kernel.org/pil Linus Torvalds <torvalds@ 2006-07-10 18:14:38 Pull dock into release branch Len Brown <len.brown@in 2006-07-10 14:20:17 ACPI: ACPI DOCK: Initialize the atomic not Kristen Accardi < kristen.c. 2006-07-10 14:19:15 Pull acpi os allocate into test branch Len Brown <len.brown@in 2006-07-10 02:39:47 ACPI: acpi os allocate() fixes Len Brown <len.brown@in 2006-07-10 01:35:51 Pull acpica-20060707 into test branch Len Brown <len.brown@in 2006-07-10 02:39:41

# A Git repository

- A Git repository is a collection of refs—branches and tags. (Branches are also known as heads.)
- A ref is a named mutable pointer to an object (usually a commit).
  - HEAD → refs/heads/master
  - refs/heads/master → commit fec6ed...
  - refs/heads/ftrace → commit ce5c1e...
  - refs/tags/v2.6.8  $\rightarrow$  commit e8ce2f...
  - refs/tags/v2.6.27  $\rightarrow$  tag 4b5127...
- The repository automatically stores the directed acyclic graph of objects rooted at these refs.

#### **Branches**

- Git was designed to enable lightweight branching and merging.
- Each repository can have any number of branches.
- Branches are just refs—pointers into the DAG of commits—and these pointers themselves are not versioned.
  - So you don't need to be afraid of making throwaway branches for experiments.

### Consequences of the Git model

- Git tracks the history of your whole project, not the history of individual files.
  - Best practice is to keep projects that are logically separate in separate Git repositories.
- Git does not track renames as metadata in the repository.
  - Instead, renames are automatically detected based on content when this information is needed.
- A commit ID cryptographically certifies the integrity of the entire history of the repository up to that commit.
  - Git has powerful tools for rewriting history—but this requires communication with everyone that has pulled any affected commits from your repository.



#### Outline

- 1 The Git model
- Using Git
- Collaboration with Git
- 4 Rewriting history
- 6 And beyond!

### Getting a Git repository

git init

Create an empty Git repository in the current directory. By default it will have one branch named master.

git clone url Clone the Git repository from url. This may be over HTTP, SSH, or the Git protocol, or it may be a path to another local repository.

Both of these operations will create a working copy.

### Working copy

- Every working copy has its own Git repository in the .git subdirectory (with arbitrarily many branches and tags).
  - The most important ref is HEAD, which refers to the current branch.
- The .git subdirectory also stores the index: a staging area for changes on top of HEAD that will become part of the next commit.
- Finally, the files outside of .git are the working tree.

#### Git workflow

- Changes made to the working tree can be added to the index.
- The index can be committed to the current branch (where it will then become the new HEAD).



# Constructing commits

git add file	Add or update $file$ from the working tree into the index.
git reset file	Unstage changes to $file$ in the index, without touching the working tree.
git checkout file	Undo modifications to $file$ in the working tree by reading it back from the index.
git rm file	Delete $file$ from the index and the working tree.
git mv oldfile ne	wfile Shortcut for mv oldfile newfile plus the appropriate additions and removals in the index.
git status	Display the files changed in the index and in the working tree.
git commit	Make a commit out of the current index.
git commit -a	Shortcut for adding all modified files to the index and committing.

### Referring to objects

```
fc8da7a06bb66b707e7f5406657d5a3b7ee42c66 You can always refer to an object by its full SHA-1 ID, but this gets unwieldy very quickly.
```

fc8da7 You can use a truncated SHA-1 as long as it is unambiguous.

*refname* You can refer to a branch or tag by name.

commit ^ Append a ^ to get the (first) parent of a commit.

commit ^2 The second parent of a commit, etc.

commit ~4 Short for commit ~^^~—the great-great-grandparent of a commit.

commit:filename The given file or directory inside commit's tree.

...and more (see git help rev-parse for a full description of the syntax).

# Displaying changes

git	log		List the commits on the current branch.
git	show	object	Show an object (e.g. the log information and patch for a commit, or the contents of a file).
git	diff		Show the differences between the index and the working tree.
git	diff	cached	Show the differences between HEAD and the index.
git	diff	commit	Show the differences between <i>commit</i> and the working tree.

# Manipulating branches and tags

```
List the branches in your repository, with the
git branch
                        current branch highlighted.
git checkout branch Switch to the branch named branch. This
                        updates HEAD, the index, and the working tree.
git checkout -b branch [commit] Create a new branch named
                        branch starting at commit (defaulting to
                        current HEAD), and switch to it.
git branch -d branch Delete the branch branch.
git branch -m oldbranch newbranch Rename oldbranch to
                        newbranch
```

git tag tag [commit] Attach a new tag named tag to commit (defaulting to current HEAD).

git tag -d tag Delete the tag named tag.

# Configuration hints

You should tell Git who you are:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your@email.edu"
```

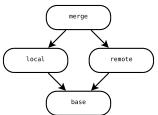
And, if you're feeling colorful,

```
$ git config --global color.ui auto
(This configuration is stored in ~/.gitconfig.)
```

#### Merging

git merge *commit* Merge *commit* into HEAD. The index must not contain any staged changes.

 In the general case, this will result in a merge commit—a commit with more than one parent.



- If commit is an ancestor of HEAD, then the merge is a no-op.
- If *commit* is a descendent of HEAD, then the merge degenerates into a *fast-forward*.

#### Resolving merge conflicts

- git merge works roughly by creating a diff against the common ancestor commit, and applying it against the current HEAD. (The general case is much more complicated.)
- Sometimes this patch will not apply to the current HEAD. This situation is called a merge conflict.
  - Git will respond by inserting *conflict markers* into the conflicted files, and asking you resolve the conflict.
  - Don't panic!
  - To resolve the conflict, edit the conflicted files appropriately and then git add them.
  - Alternatively, you can run git mergetool to resolve the conflicts interactively in a graphical diff program.



\$ seq 5 > numbers



```
$ seq 5 > numbers
$ git init
Initialized empty Git repository in /tmp/foo/.git/
```

```
$ seq 5 > numbers
$ git init
Initialized empty Git repository in /tmp/foo/.git/
$ git add numbers
```

```
$ seq 5 > numbers
$ git init
Initialized empty Git repository in /tmp/foo/.git/
$ git add numbers
```

\$ git commit -m '1, 2, 3, 4, 5!'

create mode 100644 numbers

Created initial commit 4172330: 1, 2, 3, 4, 5!
1 files changed, 5 insertions(+), 0 deletions(-)

```
$ seq 5 > numbers
$ git init
Initialized empty Git repository in /tmp/foo/.git
$ git add numbers
$ git commit -m '1, 2, 3, 4, 5!'
Created initial commit 4172330: 1, 2, 3, 4, 5!
1 files changed, 5 insertions(+), 0 deletions(-)
create mode 100644 numbers
$ git checkout -b andersk
Switched to a new branch "andersk"
```

```
$ seq 5 > numbers
$ git init
Initialized empty Git repository in /tmp/feed .git/
$ git add numbers
$ git commit -m '1, 2, 3, 4, 5!'
                                                  initial commit
Created initial commit 4172330: 1, 2, 3, 4, 5!
 1 files changed, 5 insertions(+), 0 deletions(-)
 create mode 100644 numbers
$ git checkout -b andersk
Switched to a new branch "andersk"
$ git branch
* andersk
 master
```

```
$ seq 5 > numbers
$ git init
Initialized empty Git repository in /tmp/foo/.git/
$ git add numbers
$ git commit -m '1, 2, 3, 4, 5!'
Created initial commit 4172330: 1, 2, 3 andersk initial commit
                                                                master
 1 files changed, 5 insertions(+), 0 deletions(-)
 create mode 100644 numbers
$ git checkout -b andersk
Switched to a new branch "andersk"
$ git branch
* andersk
 master
$ (echo 0; cat numbers) | sponge numbers
```

```
1 files changed, 5 insertions(+), 0 deletions(-)
 create mode 100644 numbers
$ git checkout -b andersk
Switched to a new branch "andersk"
$ git branch
* andersk
  master
$ (echo 0; cat numbers) | sponge numbers
$ git diff
                                                                 master
diff --git a/numbers b/numbers
                                                   initial commit
index 8a1218a..e8371f0 100644
--- a/numbers
+++ b/numbers
00 - 1,3 + 1,4 00
+0
```

```
create mode 100644 numbers
$ git checkout -b andersk
Switched to a new branch "andersk"
$ git branch
* andersk
  master
$ (echo 0; cat numbers) | sponge numbers
$ git diff
diff --git a/numbers b/numbers
index 8a1218a..e8371f0 100644
                                                   initial commit
--- a/numbers
+++ b/numbers
00 - 1,3 + 1,4 00
+0
 1
2
$ git add numbers
```

master

```
$ git branch
* andersk
  master
$ (echo 0; cat numbers) | sponge numbers
$ git diff
                                    HEAD
diff --git a/numbers b/numbers
index 8a1218a..e8371f0 100644
                                                Add 0
--- a/numbers
+++ b/numbers
                                                                  master
00 - 1,3 + 1,4 00
                                                    initial commit
+0
2
$ git add numbers
$ git commit -m 'Numbers start at 0.'
Created commit 7aeb494: Numbers start at 0.
 1 files changed, 1 insertions(+), 0 deletions(-)
```

```
master
$ (echo 0; cat numbers) | sponge numbers
$ git diff
diff --git a/numbers b/numbers
index 8a1218a..e8371f0 100644
--- a/numbers
+++ b/numbers
                                               Add 0
@@ -1.3 +1.4 @@
+0
 1
                                                   initial commit
2
$ git add numbers
$ git commit -m 'Numbers start at 0.'
Created commit 7aeb494: Numbers start at 0.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch "master"
```

```
$ (echo 0; cat numbers) | sponge numbers
$ git diff
diff --git a/numbers b/numbers
index 8a1218a..e8371f0 100644
--- a/numbers
+++ b/numbers
00 - 1,3 + 1,4 00
                                               Add 0
+0
                                                                  HEAD
2
                                                   initial commit
$ git add numbers
$ git commit -m 'Numbers start at 0.'
Created commit 7aeb494: Numbers start at 0.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch "master"
$ echo 6 >> numbers
```

```
$ git diff
diff --git a/numbers b/numbers
index 8a1218a..e8371f0 100644
--- a/numbers
+++ b/numbers
00 - 1,3 + 1,4 00
+0
                                               Add 0
 1
2
                                                   initial commit
$ git add numbers
$ git commit -m 'Numbers start at 0.'
Created commit 7aeb494: Numbers start at 0.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch "master"
$ echo 6 >> numbers
$ git add numbers
```

```
--- a/numbers
+++ b/numbers
00 - 1,3 + 1,4 00
+0
2
                                                             Add 6
                                               Add 0
$ git add numbers
$ git commit -m 'Numbers start at 0.'
Created commit 7aeb494: Numbers start at 0.
                                                   initial commit
 1 files changed, 1 insertions(+), 0 deletions(-
$ git checkout master
Switched to branch "master"
$ echo 6 >> numbers
$ git add numbers
$ git commit -m '6 is a number too.'
Created commit 383c158: 6 is a number too.
 1 files changed, 1 insertions(+), 0 deletions(-)
```

```
2
$ git add numbers
$ git commit -m 'Numbers start at 0.'
                                                     merge
Created commit 7aeb494: Numbers start at 0.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
                                              Add 0
                                                            Add 6
Switched to branch "master"
$ echo 6 >> numbers
$ git add numbers
                                                   initial commit
$ git commit -m '6 is a number too.'
Created commit 383c158: 6 is a number too.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git merge andersk
Auto-merged numbers
Merge made by recursive.
 numbers | 1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

```
$ echo 6 >> numbers
$ git add numbers
                                                                  HEAD
$ git commit -m '6 is a number too.'
Created commit 383c158: 6 is a number too.
 1 files changed, 1 insertions(+), 0 deletions(-
$ git merge andersk
Auto-merged numbers
                                               Add 0
                                                             Add 6
Merge made by recursive.
 numbers |
           1 +
 1 files changed, 1 insertions(+), 0 deletions(-finitial commit
$ cat numbers
3
5
6
```

```
$ git commit -m '6 is a number too.'
Created commit 383c158: 6 is a number too.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git merge andersk
Auto-merged numbers
                                    HEAD
Merge made by recursive.
 numbers |
           1 +
                                               Add 0
                                                             Add 6
 1 files changed, 1 insertions(+), 0 delet
$ cat numbers
                                                   initial commit
3
5
$ git checkout andersk
Switched to branch "andersk"
```

```
Created commit 383c158: 6 is a number too.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git merge andersk
Auto-merged numbers
Merge made by recursive.
                                      HEAD
 numbers |
            1 +
 1 files changed, 1 insertions(+) andersk eleg
                                                  Add 0
                                                                Add 6
$ cat numbers
0
                                                      initial commit
3
5
6
$ git checkout andersk
Switched to branch "andersk"
$ echo 5\frac{1}{5} >>  numbers
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
$ git merge andersk
Auto-merged numbers
Merge made by recursive.
 numbers |
           1 +
 1 files changed, 1 insertions(+), we deletions(-)
 cat numbers
                                                Add 0
                                                               Add 6
                                                     initial commit
5
6
$ git checkout andersk
Switched to branch "andersk"
$ echo 5\frac{1}{2} >> numbers
$ git add numbers
```

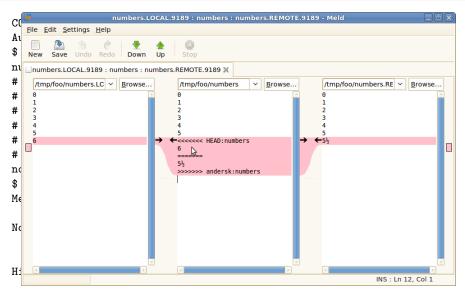
```
Merge made by recursive.
                                  HEAD
 numbers |
 1 files changed, 1 insertions ,,
                                                      (-)
                                              Add 55
  cat numbers
                                                            merge
0
                                                    Add 0
                                                                    Add 6
5
                                                         initial commit
6
$ git checkout andersk
Switched to branch "andersk"
$ echo 5\frac{1}{2} >> numbers
$ git add numbers
$ git commit -m '5\frac{1}{2} is a better number.'
Created commit 5360c2d: 5\frac{1}{2} is a better number.
 1 files changed, 1 insertions(+), 0 deletions(-)
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
$ cat numbers
                                                                        HEAD
                                             Add 5⅓
3
                                                                   Add 6
                                                   Add 0
5
6
$ git checkout andersk
                                                        initial commit
Switched to branch "andersk"
$ echo 5\frac{1}{2} >> numbers
$ git add numbers
$ git commit -m '5\frac{1}{2} is a better number.'
Created commit 5360c2d: 5\frac{1}{2} is a better number.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch "master"
```

```
3
                                                                      HEAD
                                            Add 53
5
6
$ git checkout andersk
Switched to branch "andersk"
                                                                 Add 6
                                                  Add 0
$ echo 5\frac{1}{5} >>  numbers
$ git add numbers
$ git commit -m '5\frac{1}{2} is a better number.'
                                                       initial commit
Created commit 5360c2d: 5\frac{1}{2} is a better number.
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch "master"
$ git merge andersk
Auto-merged numbers
CONFLICT (content): Merge conflict in numbers
Automatic merge failed; fix conflicts and then commit the result.
```

```
$ git commit -m '5\frac{1}{5} is a better number.'
Created commit 5360c2d: 5\frac{1}{2} is a better number.
 1 files changed, 1 insertions ( ),
                                                                   HEAD
                                                  (-)
                                          Add 55
$ git checkout master
Switched to branch "master"
$ git merge andersk
Auto-merged numbers
                                                Add 0
                                                              Add 6
CONFLICT (content): Merge conflict in numb
Automatic merge failed; fix conflicts and then commit the result.
$ git status
                                                    initial commit
numbers: needs merge
# On branch master
# Changed but not updated:
    (use "git add <file>..." to update what will be committed)
#
#
 unmerged: numbers
no changes added to commit (use "git add" and/or "git commit -a")
```

```
CONFLICT (content): Merge conflict in numbers
Automatic merge failed; fix conflicts and then commit the result.
                                                                  HEAD
$ git status
                                         Add 55
numbers: needs merge
# On branch master
# Changed but not updated:
    (use "git add <file>..." to update wh/
#
                                                       COI
                                               Add 0
                                                             Add 6
#
 unmerged: numbers
#
                                                   initial commit
no changes added to commit (use "git add" and/or
$ git mergetool
Merging the files: numbers
Normal merge conflict for 'numbers':
  local: modified
  remote: modified
Hit return to start merge resolution tool (meld):
```



```
#
no changes added to commit (use "git add" and/or "git commit -a")
                              andersk
                                                                    HEAD
$ git mergetool
                                           Add 55
Merging the files: numbers
Normal merge conflict for 'numbers':
  local: modified
                                                 Add 0
                                                               Add 6
  remote: modified
Hit return to start merge resolution tool (meld)
$ cat numbers
                                                     initial commit
0
3
5
5출
6
```

```
HEAD
3
                                           Add 5⅓
5
5출
6
                                                               Add 6
                                                 Add 0
 git status
  On branch master
  Changes to be committed:
                                                     initial commit
    (use "git reset HEAD <file>..." to unstage)
#
 modified: numbers
  Untracked files:
    (use "git add <file>..." to include in what will be committed)
 numbers.orig
```

```
3
5
                                          Add 55
5출
                                                       merge
6
  git status
  On branch master
                                                               Add 6
                                                Add 0
  Changes to be committed:
#
    (use "git reset HEAD <file>..." to unstage)
#
                                                    initial commit
 modified: numbers
  Untracked files:
    (use "git add <file>..." to include in what will be committed)
# numbers.orig
$ git commit
Created commit fc8da7a: Merge branch 'andersk'
```

HEAD

# Getting out of trouble

The graphical repository browser is gitk HEAD --all immensely useful for visualizing what's going on in your repository. git reflog Show the *reflog* entries for HEAD. Show the *reflog* entries for *ref*. git reflog show ref Resets the ref pointed to by HEAD, as well git reset --hard commit as the index and working tree, to commit.

- The reflog tracks all local changes to refs. Whenever a ref is updated to point at a new commit, it gets an entry in the reflog.
- If you find yourself somewhere you don't expect, you can examine the log or the reflog, and then use reset to get back to a known point.
- This works even in a conflicted merge or rebase, if you just want to bail out and try something different.

fc8da7a... HEAD@0: commit (merge): Merge branch 'andersk'
994be80... HEAD@1: checkout: moving from andersk to master

# A peek at the reflog

\$ git reflog

```
5360c2d... HEAD@2: commit: 5\frac{1}{2} is a better number.
7aeb494... HEAD@3: checkout: moving from master to andersk
994be80... HEAD@4: merge andersk: Merge made by recursive.
383c158... HEAD@5: commit: 6 is a number too.
4172330... HEAD@6: checkout: moving from andersk to master
7aeb494... HEAD@7: commit: Numbers start at 0.
4172330... HEAD@8: checkout: moving from master to andersk
$ git reflog show master
fc8da7a... master@0: commit (merge): Merge branch 'andersk'
994be80... master@1: merge andersk: Merge made by recursive.
383c158... master@2: commit: 6 is a number too.
4172330... master@3: commit (initial): 1, 2, 3, 4, 5!
$ git reflog show andersk
5360c2d... and ersk@0: commit: 5\frac{1}{2} is a better number.
7aeb494... andersk@1: commit: Numbers start at 0.
4172330... andersk@2: branch: Created from HEAD
Nelson Elhage, Anders Kaseorg (SIPB)
                                 Understanding Git
                                                            October 21, 2008
                                                                          23 / 41
```

# Cherry-picking and reverting

git cherry-pick commit Constructs a new commit on HEAD that performs the same changes as commit. Constructs a new commit on HEAD that performs the reverse of the changes in commit.

- These commands construct a new commit that does not preserve any parent information pointing back to the old one. Use with care.
- Instead of cherry-picking from your development branch into your stable branch, for example, it is usually better to make the commit on stable and merge the entire stable branch into development.

#### Outline

- The Git model
- Using Git
- 3 Collaboration with Git
- 4 Rewriting history
- 6 And beyond!

#### Collaboration with Git

- Git allows bidirectional communication between any pair of repositories.
- Git speaks many protocols.
  - SSH
  - HTTP/HTTPS
  - DAV
  - Git protocol
  - rsync
  - direct filesystem access
- This flexibility lets you implement a wide range of centralized or distributed development models.

## The simple case

 A freshly cloned repository has one remote called origin, which is the default source for pulls and destination for pushes.

git branch -r List the available remote branches.
git branch -a List the available local and remote branches.

• Development is done on local branches. To work on a remote branch, you first create a local *tracking branch*, and then push any changes back to the remote branch as a separate operation.

## Tracking branches

git checkout -b branch origin/branch Create and switch to a new tracking branch named branch, set up to track the remote branch origin/branch. Update the current tracking branch from git pull origin/branch. Short for git fetch; git merge origin/branch. Push the current tracking branch back to git push origin/branch. This will only fast-forward the remote branch by default, so you may need to git pull first. git push origin : branch Delete the remote branch branch. git remote prune origin Clean up any refs to branches that have been deleted remotely.

#### Remotes

- A Git repository can be configured with references to any number of remotes.
- By default, a newly cloned repository has one remote named origin pointing to the source of the clone.

```
$ git clone /mit/andersk/Public/git/nss_nonlocal.git
Initialized empty Git repository in /tmp/nss_nonlocal/.git/
$ cat nss_nonlocal/.git/config
...
[remote "origin"]
url = /mit/andersk/Public/git/nss_nonlocal.git
fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote = origin
merge = refs/heads/master
```

## Hosting a public Git repository

- A repository that's used for cloning, pulling, and pushing should usually be a bare repository (git clone --bare). A bare repository has no working tree, and lives in a directory named project.git instead of project/.git.
- The quickest solution at MIT is to drop your repository into AFS.
- To serve a repository on the web, you need to run git update-server-info, and enable the hooks/post-update hook.
- To serve a repository via the Git protocol, you need to create the git-daemon-export-ok file inside it.
  - scripts.mit.edu provides a Git hosting service. Drop your repository into /mit/locker/Scripts/git/project.git and access it at git://locker.scripts.mit.edu/project.git.

#### Outline

- 1 The Git mode
- Using Git
- Collaboration with Git
- Rewriting history
- 6 And beyond!

## Rewriting history

- Git includes powerful tools for rewriting history.
- Of course, since modifying a commit changes its SHA-1 identifier, by "rewriting history" we actually mean "transforming a sequence of commits into a different sequence of commits".
- You need to be careful about rewriting commits that others may have already pulled.
  - By default, Git will prevent you from pushing changes that are not fast-forwards, unless you ask very hard.
- Rewriting is extremely useful for cleaning up a private branch before making it publicly available.

# Why rewriting is useful

- A good history will include one commit for each self-contained logical change to the tree.
- Avoid cluttering the history with typos and trivial bugs that are fixed in the following commits.
  - This makes things more pleasant for anyone who wants to read or review your changes.
  - It also makes it easier to pinpoint bugs with git bisect.
- You don't need to worry about making your commits perfect as you write them, since you can rearrange them later.

## Resetting branches

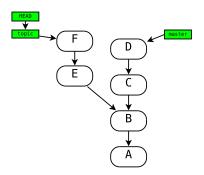
```
git reset --hard commit Resets the current HEAD, as well as the
                            index and working tree, to commit.
                            Resets the current HEAD and index to
git reset commit
                            commit, without touching the working tree.
git reset --soft commit
                            Resets the current HEAD to commit, without
                            touching the index or the working tree.
                            Adds the modifications in the index to the
git commit --amend
                            current commit at HEAD "in place".
                            Approximately equivalent to git reset
                            HEAD'; git commit.
```

# Rebasing

- rebase finds all commits that are in HEAD but not in commit, and re-applies them starting with commit. The current branch is reset to the result.
- This has a similar effect to a merge, but maintains a linear history, at the cost of losing some information.
- rebase changes the object identifiers of the re-applied commits.
- rebase is often preferred to keep history clean.

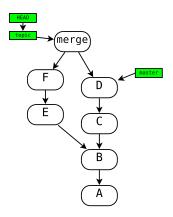
## Rebase vs. merge

• We have development on both a topic branch and master.



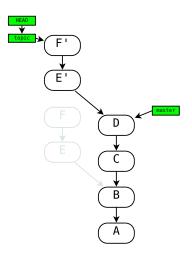
# Rebase vs. merge

• merge results in a forked history:



## Rebase vs. merge

• rebase rewrites commits and maintains a linear history:



# Interactive rebasing

git rebase -i commit Rebase HEAD onto commit, letting you interactively edit the resulting history.

(Typically commit will already be an ancestor of HEAD, to edit history "in place".)

- Git will start your editor on a list of the commits to be applied on top of commit.
- You can cut and paste to arbitrarily reorder the commits.
- You can delete a line to remove that commit completely.
- You can insert the squash directive to fuse a commit into the previous commit.
- You can insert the edit directive to have Git pause after applying a commit, so you can amend it in place or insert new commits, before further commits are applied.

# Advanced rewriting

git filter-branch Rewrite history by mapping each commit through an arbitrary script. Causes the local repository to pretend that certain .git/info/grafts commits have different parents than their real ones. (git filter-branch can then rewrite the fake parents into real ones.) Dump history in a human-readable format, with git fast-export SHA-1 IDs replaced by symbolic marks, so that it can be edited by hand. git fast-import Read back commits produced by git fast-export.

#### Outline

- 1 The Git model
- Using Git
- Collaboration with Git
- 4 Rewriting history
- 6 And beyond!

#### Other awesome Git commands

git bisect	Easily pinpoint a regression in your history using a repeated bisection search.
git blame	Annotate each line of a file with information about its last modification.
git cvsimport, gi	t svn Use Git to work with repositories in other formats. (I think Git makes a much better CVS or SVN client than the native ones!)
git format-patch	, git send-email, git am Send and receive Git patches by email.
git grep	Search for a regex in a Git tree.
git stash	Quickly stash away and reapply temporary changes while you do other work.
git submodule	Manage a group of related Git repositories.

# Exploring Git yourself

- There are many commands we haven't talked about, and the ones we have take additional options that can help you work more efficiently.
- Anything you think you should be able to do within the Git model can probably be done.
- Git is designed to be conveniently scriptable.
- Git has extensive documentation—start with man git.
  - To get documentation on any git command, run git help command or (equivalently) man git-command.