

Using Git on OS X

Munich Cocoaheads

2009-11-12

2009-12-10

©2009 Stephen Riehm

Coming up

Basic Concepts

Daily Git

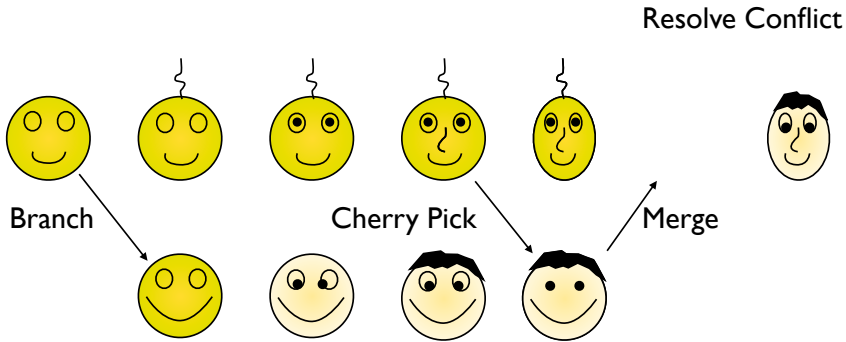
Git with XCode

Non-Obvious Git

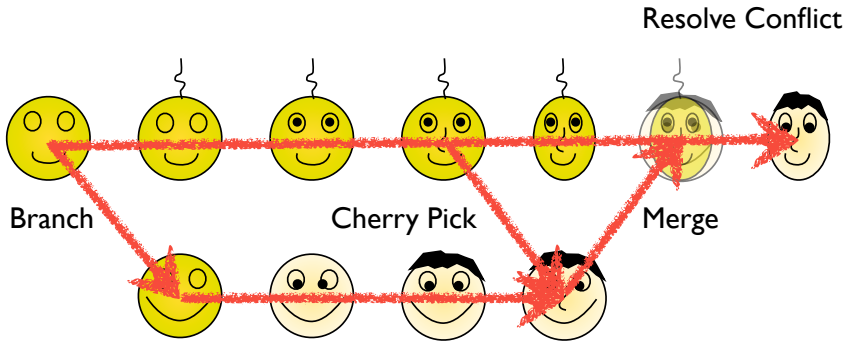
What is git?

Source Configuration Management

Source Configuration Management

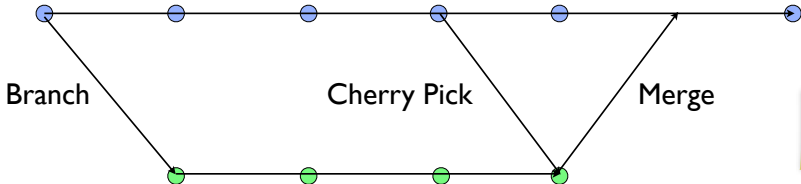


Source Configuration Management



Source Configuration Management

Resolve Conflict



unresolved conflicts
are not committed to
the repository!

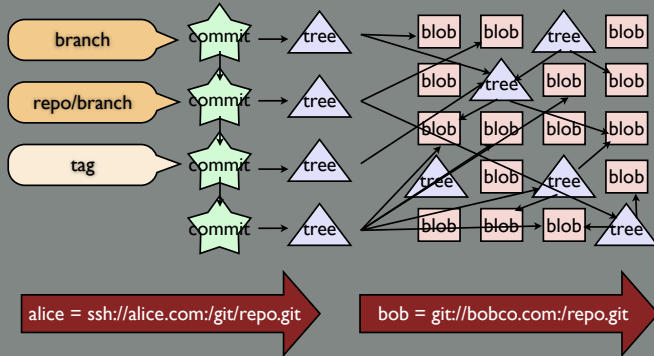
Back to git

project_dir/

.git/

What is git?

.git/



Git's Bits

Blobs

blob	size
<pre>// Blah.m // This class does @implementation Bla @synthesize a;</pre>	

Trees

tree		size	
fW-fW-fW-	blob	5b1d3...	ReadMe
fWxfWxfWx	tree	03e78...	Classes
fWxfWxfWx	tree	cdc8b...	Resources
fW-fW-fW-	blob	cba0a...	Info.plist
fWxfWxfWx	blob	911e7...	distrib.pl

Commits

commit	size
tree	c4ec5...
previous commit	a149e...
author	Scott
committer	Scott
date	2009-11-10 21:15
commit message	

Tags

tag	size
commit	ae668...
type	commit
tagger	Scott
tag message	

Blobs

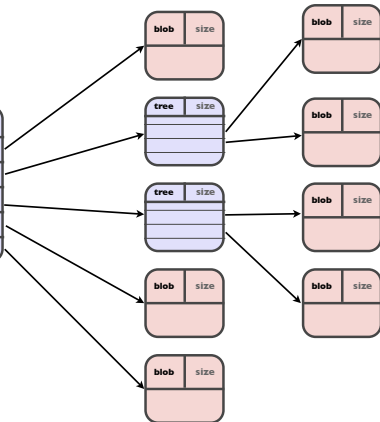
58206c1725109d441fe846300fd4e57063cc6d6b

blob	size
<pre>// Blah.m // This class does @implementation Bla @synthesize a;</pre>	

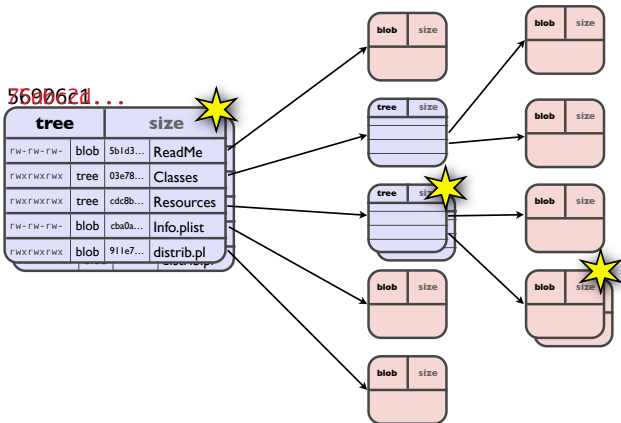
Trees

56906c1...

tree		size	
ΓW-ΓW-ΓW-	blob	5b1d3...	ReadMe
ΓWXΓWXΓWX	tree	03e78...	Classes
ΓWXΓWXΓWX	tree	cdc8b...	Resources
ΓW-ΓW-ΓW-	blob	cba0a...	Info.plist
ΓWXΓWXΓWX	blob	911e7...	distrib.pl



Changes

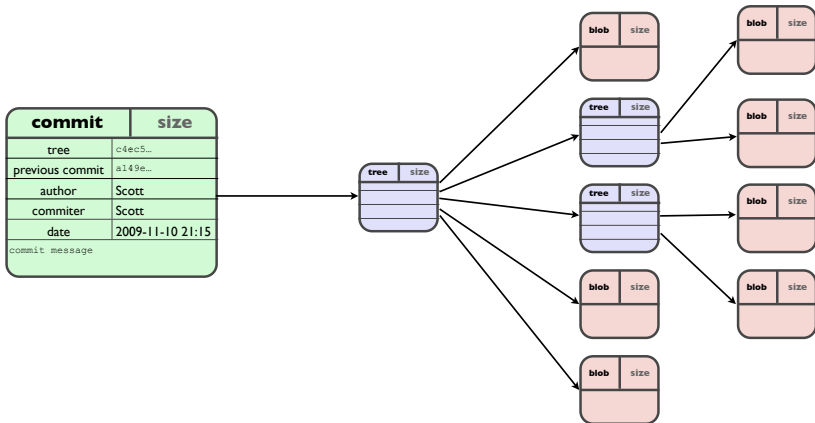


Commits

56906c1...

commit	size
tree	c4ec5...
previous commit	a149e...
author	Scott
committer	Scott
date	2009-11-10 21:15
commit message	

Commits

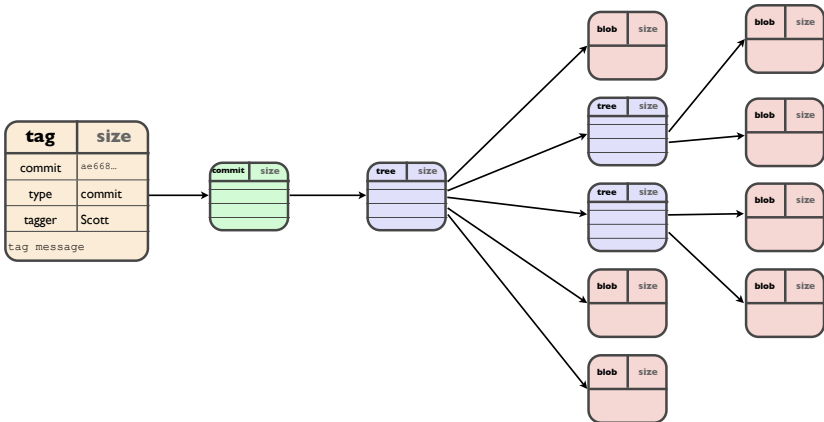


Tags

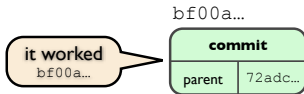
56906c1...

tag	size
commit	ae668...
type	commit
tagger	Scott
tag message	

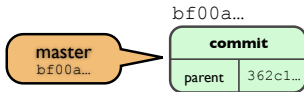
Tags



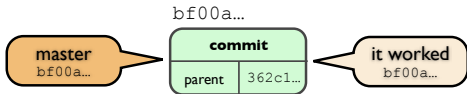
Light-weight tags



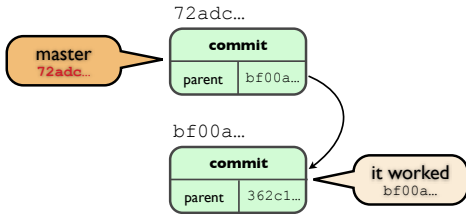
Branches



Branches 'v' Tags

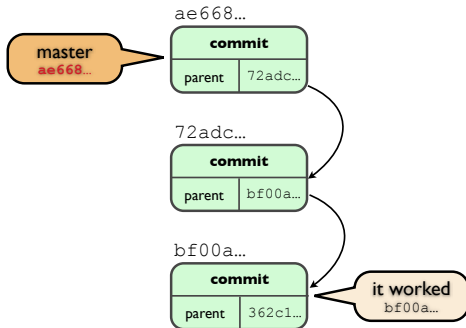


Branches 'v' Tags



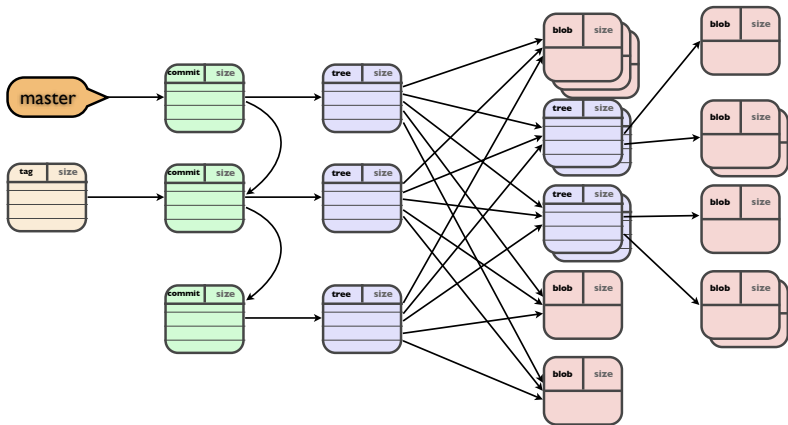
Branches 'v' Tags

HEAD and your current branch are automatically updated when you commit a change.

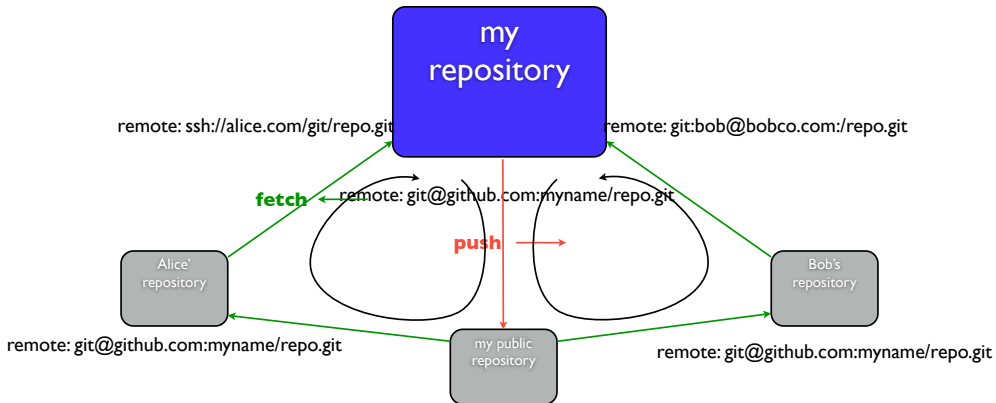


tags are immutable and reliably point to a known state of the entire repository

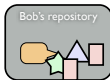
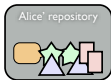
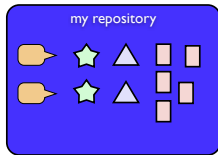
The whole lot



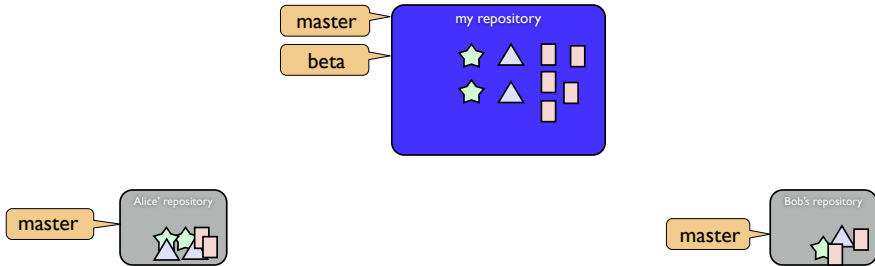
Sharing



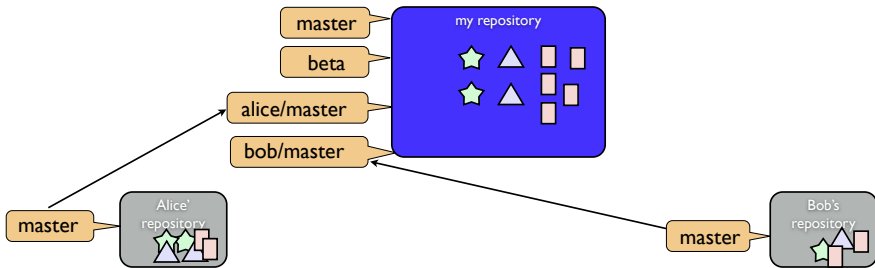
Namespaces



Namespaces

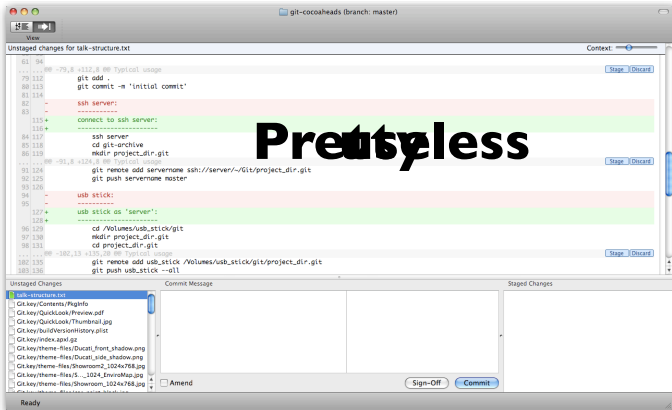


Namespaces

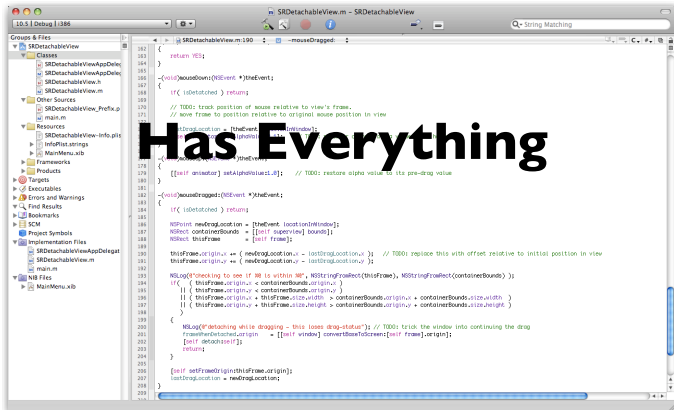


Git On OS X

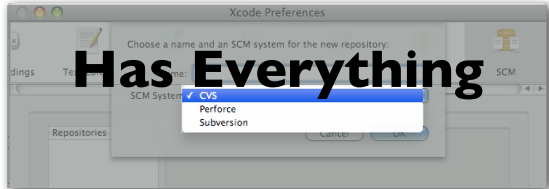
gitx



XCode



XCode



Has Everything

Git your hands dirty

```
~/ > echo "you're going to need the command line :-)"
```

First Steps

global Configuration

```
~/ > $EDITOR ~/.gitconfig
```

```
[core]
  pager = more
  excludesfile = /Users/me/.gitignore
[user]
  name = My Full Public Identity
  email = h4x0r@example.com
[format]
  pretty = format:%h %ci [%aN] %s
```

pager stops long lists from flying past your nose uncontrollably

excludesfile specifies files should never be checked into a git repository

format:

%h = hash id

%ci = commit date, iso 8601 format

%aN = author name

%s = summary

more info: `git log --help`

global Configuration

```
~/ > $EDITOR ~/.gitignore

# apple typical files
.DS_Store
.Spotlight-V100
.com.apple.timemachine.supported
.fseventsdbuild

# XCode user state files
*.modelv3
*.pbxuser
*.objc_sync

# other SCM systems
.svn

# editor temporary files
*~
*.swp

# files you generate while building
build/
version.txt
CHANGELOG
```

Things which git should probably ignore (for commands like `git add .` which just grab everything)

Where to look for Help

```
~/ > git help <cmd>
```

```
~/ > git <cmd> --help
```

make sure you install
git's man-pages.

Use the [man](#) branch of
the git repository

Where to look for Help

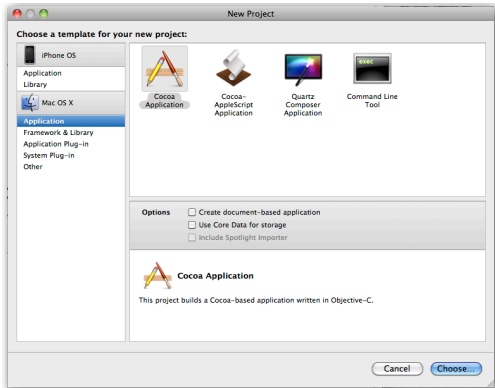
<http://git-scm.com>

<http://github.com>

<http://gitready.com>

<http://google.com>

A new Project



New Project in Git

```
~/> cd project
```

```
~/project/ > git init
```

```
Initialized empty Git repository in project_dir/.git/
```

```
~/project/ > git add .
```

```
~/project/ > git commit -m 'initial commit'
```

```
[master (root-commit) 64fb323] initial commit
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```

```
create mode 100644 hello.txt
```

In english:

- create your project in XCode
- create a new git repository
- add your files & directories to git
- commit your changes

Git Configuration for Xcode

```
~/project/ > $EDITOR .gitattributes
```

```
*.pbxproj -crlf -diff -merge  
*.nib      -crlf -diff -merge  
*.xib      -crlf -diff -merge  
*.graftle -crlf -diff -merge
```

```
~/project/ > git add .gitattributes
```

```
~/project/ > git commit -m 'add .gitattributes - prevent accidental merging of special XCode files'
```

```
[master (root-commit) 64fb323] initial commit  
1 files changed, 1 insertions(+), 0 deletions(-)  
create mode 100644 .gitattributes
```

Tell git to treat these files as if they were binaries.

XML files are notorious for being text, but "unmergeable".

`.gitattributes` is project-specific
must be checked into each project separately
will automatically be used by all project members

Joining An Existing Project

cloning a repository automatically sets up a remote repository called `origin`.

You can specify a different name for the remote repository with `-o name`

```
~/ > git clone -o cloned_repo URL/project.git
```

```
~/ > cd project
```

```
~/project/ > git checkout -b my_stuff cloned_repo/master
```


Clone a local repository

cloning a repository automatically sets up a remote repository called `origin`.

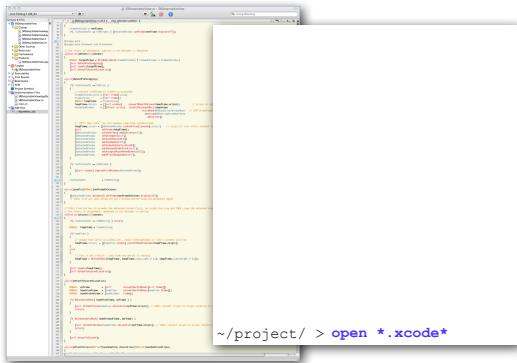
You can specify a different name for the remote repository with `-o name`

```
~/ > git clone ~/old_project_dir ~/new_project_dir
```

git with XCode

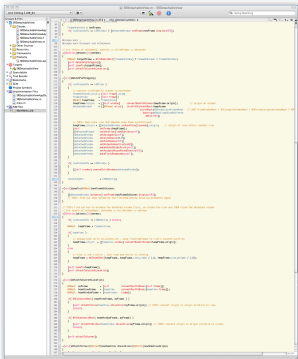
XCode & git

I warned you!
Grab your favourite terminal
window and start typing...



XCode & git

A typical sequence of commands



```
~/project/ > git status
```

```
~/project/ > git diff
```

```
~/project/ > git checkout -b fix
```

work work work...

```
~/project/ > git commit -am '...'
```

```
~/project/ > git checkout master
```

```
~/project/ > git merge fix
```

```
~/project/ > git push public
```

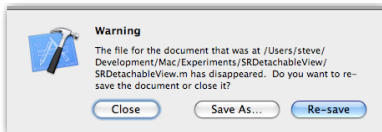
if this happens...

`git checkout`



"Read from Disk" will force Xcode not to update your repository

However...



If you get this message, you should:

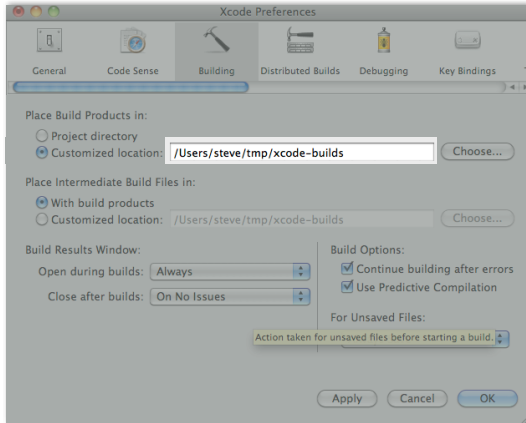
Save your work (possibly in a temporary directory)

Close XCode

Fix up your working directory

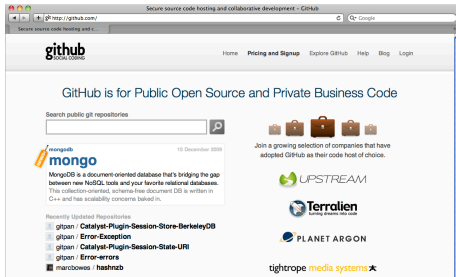
Open XCode again

XCode - Tips



github

github



github - ssh keys

Copy and paste your public key into the SSH Public Keys tab of your github account settings.

```
~/ .ssh/ > ssh-keygen -t rsa -f github
```

Generating public/private rsa key pair.

Enter passphrase (empty for no passphrase): *password or just hit return*

Enter same passphrase again: *password or just hit return again*

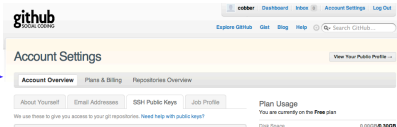
```
~/ .ssh/ > ls
```

github

github.pub

```
~/ .ssh/ > cat github.pub
```

```
ssh-rsa AAAAB3NzaClyc2EAAAABIwAAAQEArkyf...
```



Daily Work

add & commit

work work work...

```
~/project/ > git status
```

```
~/project/ > git add file file file directory... or git add -A or git add -u
```

```
~/project/ > git status
```

```
~/project/ > git commit -m 'what I just did'
```

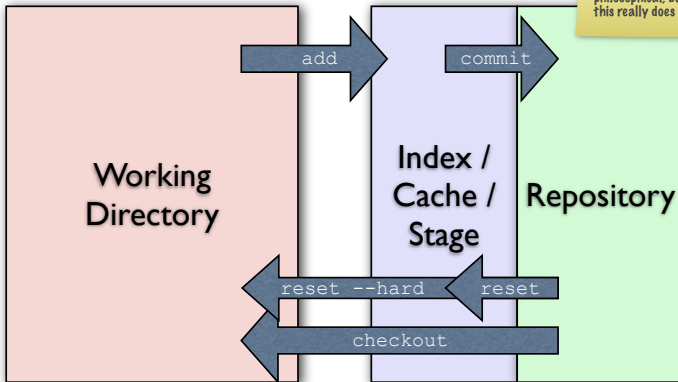
```
~/project/ > git commit -a -m 'what I just did'
```

`git add -A`
new files
changed files
removed files

`git add -u`
changed files
removed files

`git commit -a`
same as
`git add -A; git commit`

Why Add & Commit?



OK, so this is more the "how" than "why". Asking why very quickly gets philosophical, but splitting things up like this really does have its uses.

Branching

```
~/project/ > git checkout -b new_branch
```

```
~/project/ > git branch -a
```

```
~/project/ > git branch -d old_branch
```

```
~/project/ > git branch -D old_branch
```

TIP: you can create a new branch AFTER you have already made changes.

Just `checkout -b new_branch` before you `git add`

Delete a branch which has become part of another branch (nothing will be lost)

Delete a branch that cannot be re-constructed without knowing the commit ID (if you didn't write it down, it's gone!)

Differences?

```
~/project/ > git diff
```

```
~/project/ > git diff --cached
```

```
~/project/ > git diff HEAD
```

```
~/project/ > git diff other_branch
```

Merging

git always merges into the working directory

merged files are added automatically

conflicts are not added - you need to resolve them first

```
~/project/ > git merge other_branch
```

fix conflicts...

```
~/project/ > git add -A
```

```
~/project/ > git commit -m 'merge changes from other_branch'
```


Throwing things away

```
~/project/ > git reset commit
```

`git reset` updates the cache to reflect the named commit. No changes are made to your working tree. (usefull if you want to un-add something)

```
~/project/ > git reset --hard commit
```

`git reset --hard` updates the cache and the working tree to match the named branch (by default `HEAD`). This will kill any uncommitted changes!

Multiple Branches

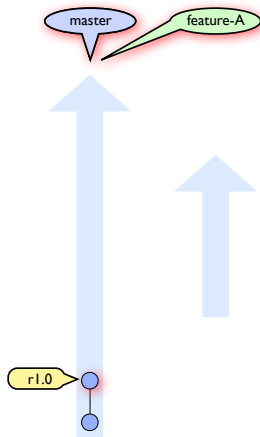
step by step...



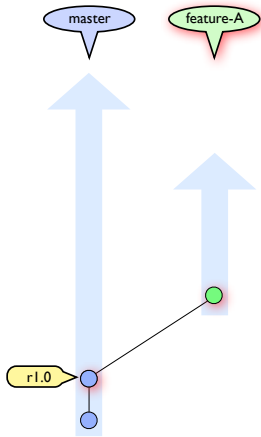
```
git checkout master
```



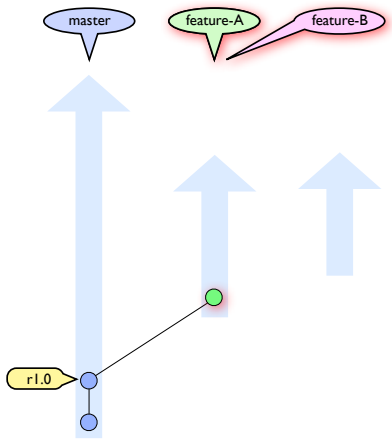
```
git checkout master  
git checkout -b feature-A
```



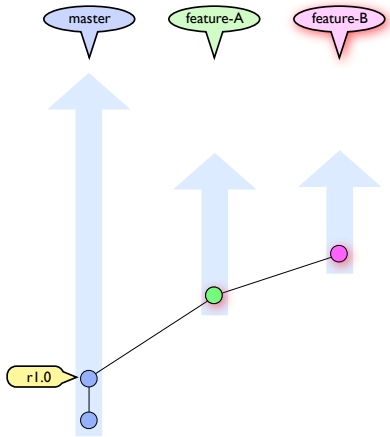
```
git checkout master  
git checkout -b feature-A  
git commit -a -m 'basic feature A structure'
```



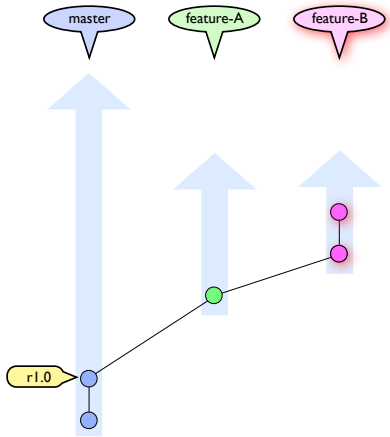
```
git checkout master  
git checkout -b feature-A  
git commit -a -m 'basic feature A structure'  
git checkout -b feature-B
```



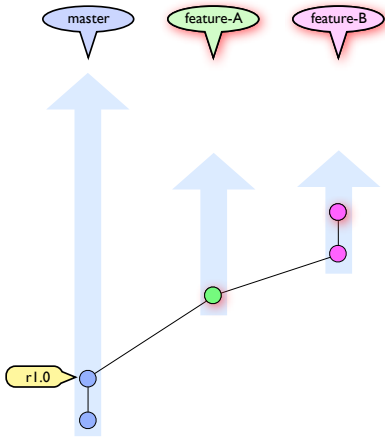
```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
```



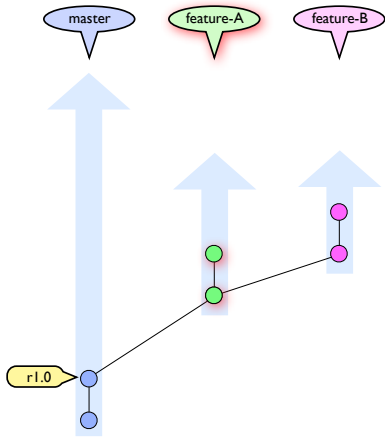

```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
```



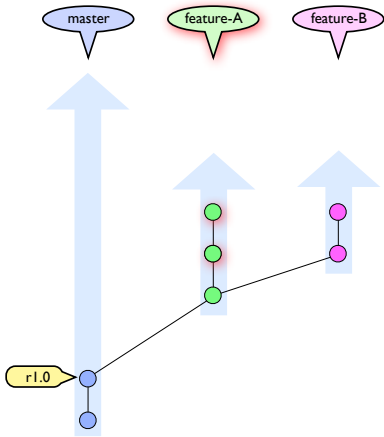
```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
```



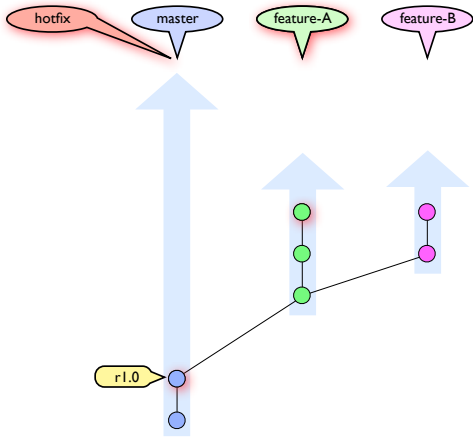
```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
```



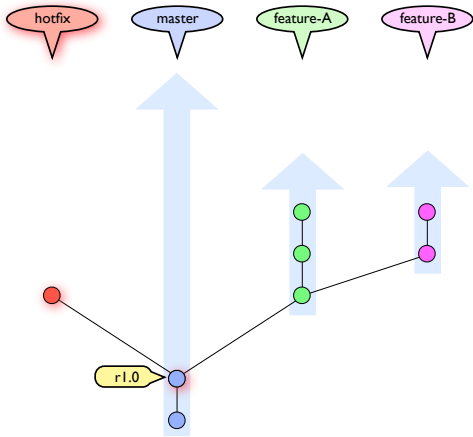
```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
```



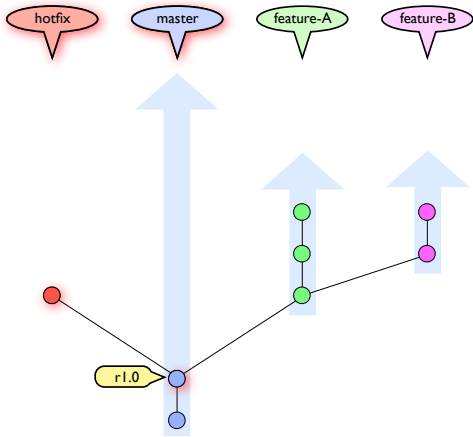
```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
```



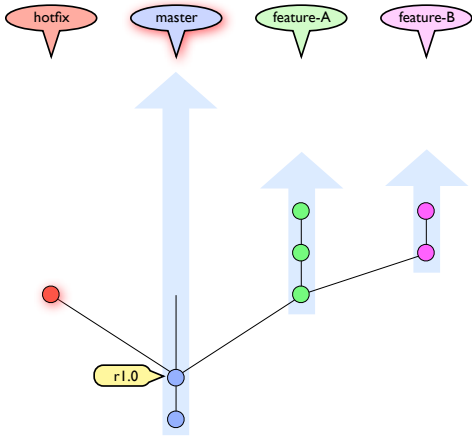
```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
```



```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
```

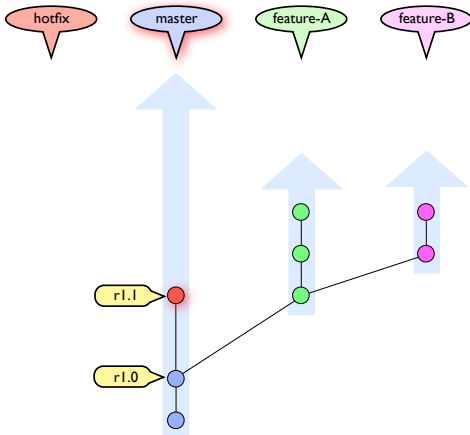


```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
```

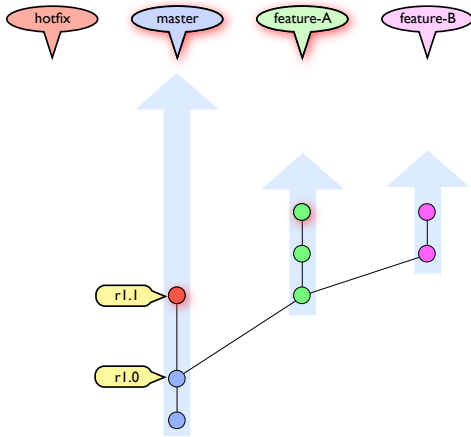


This is a "fast forward" merge.
No *merging* actually takes place.
Instead, the current branch is simply updated to the head of the branch being merged.

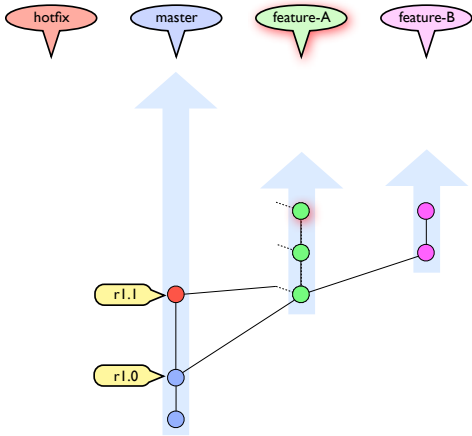

```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
```



```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
```



```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
```

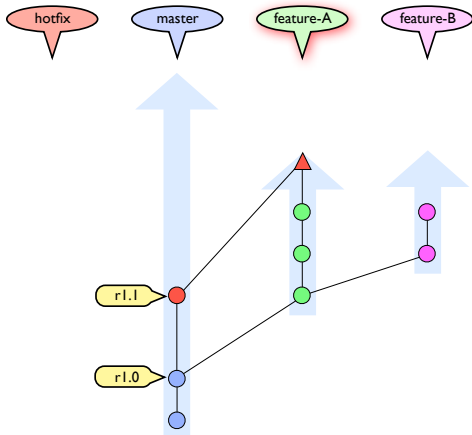


rebasing creates new copies of **every** commit between the base and the head!

DO use rebase if you are about to synchronise your work with a public repository.

DO NOT use rebase if the effected commits have already been published!

```
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.1
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git merge master
...resolve conflicts...
git commit -m 'merge from rl.1'
```

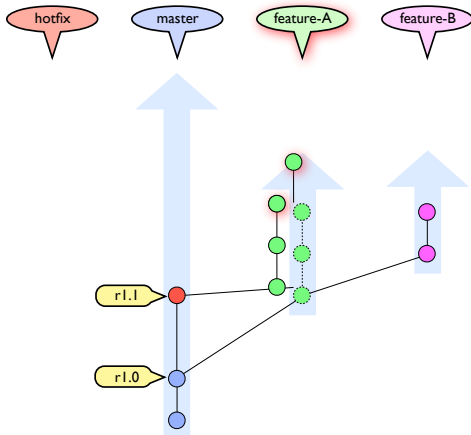


merging applies the changes from the source branch onto the head of the target branch.
Existing commits remain effective, the merged commits are duplicated.
DO use merge if your commits have been published!

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'

```

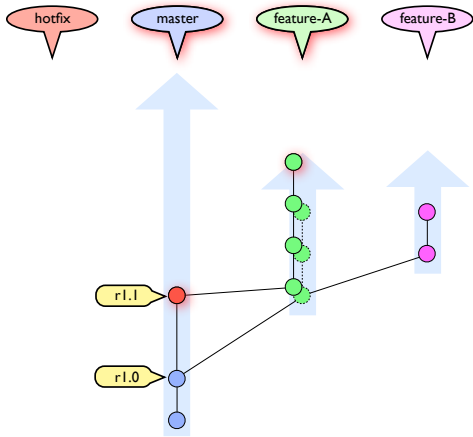


The original commits are no longer accessible via the branch.
Branches stemming from an original commit still reference it!

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master

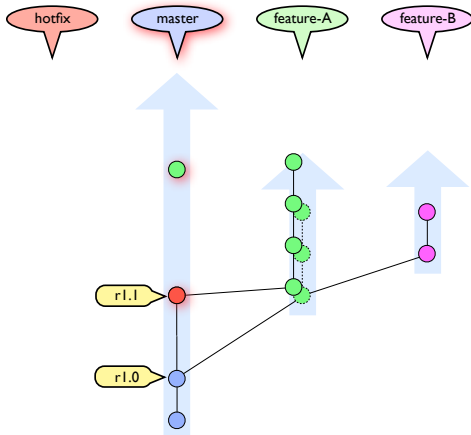
```



```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A

```

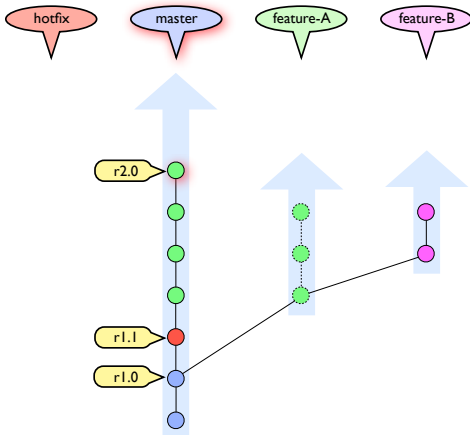


fast forward merge again

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'

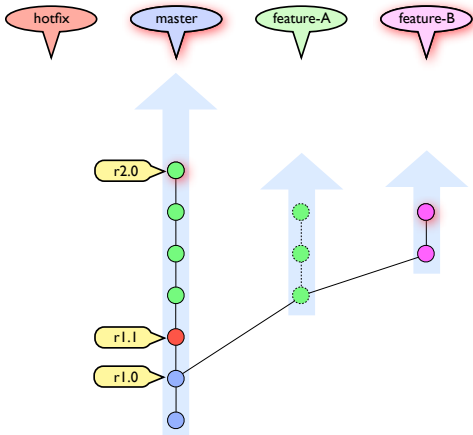
```




```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B

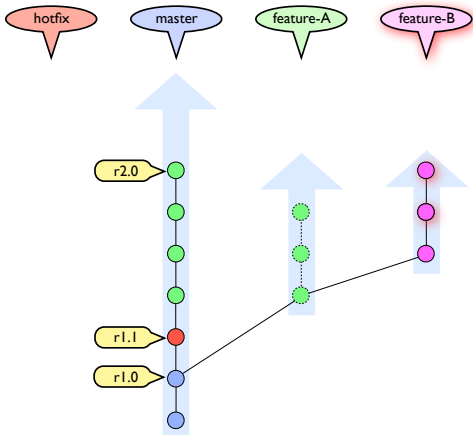
```



```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'

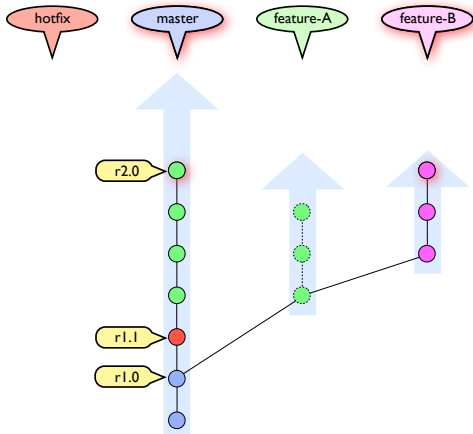
```



```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master

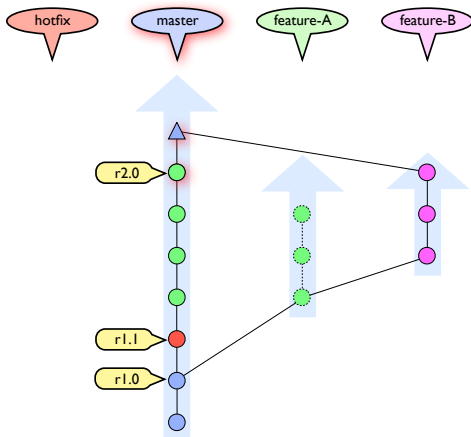
```



```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'

```



divergent branches require a new commit with 2 parents.

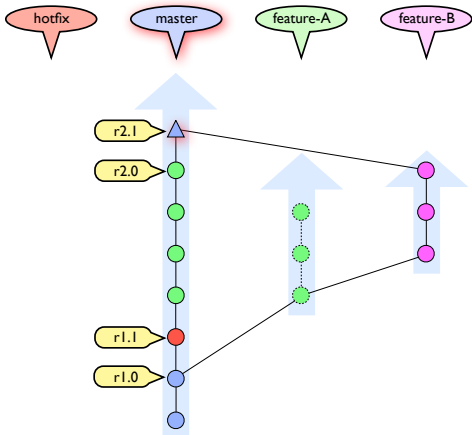
The new commit tracks conflict resolutions.

`git merge` automatically detects if a fast forward merge is possible or not.

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a r1.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'

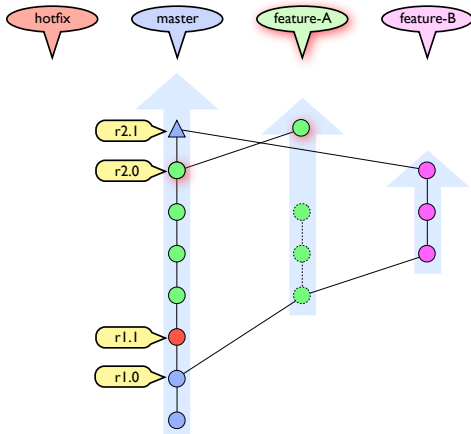
```




```

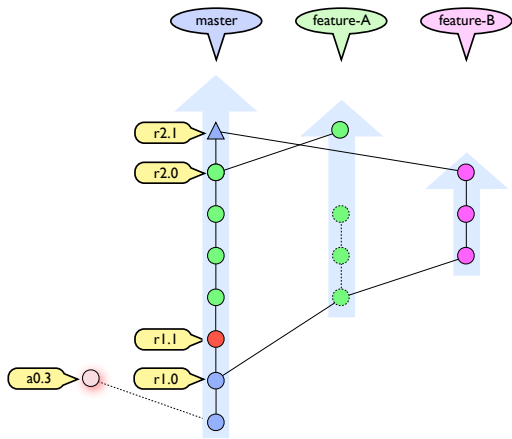
git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix rl.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a rl.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'
git checkout feature-A
git commit -a -m 'feature-A extension'

```



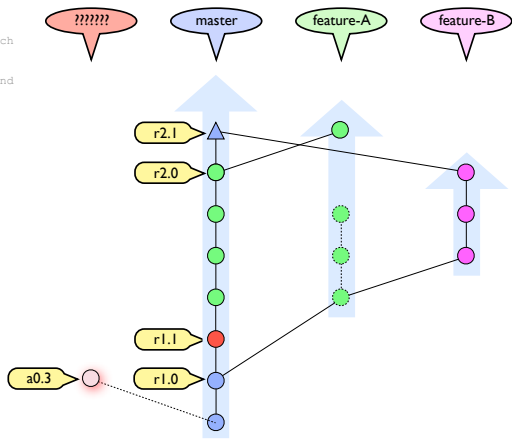
Lost your Head?


```
git checkout a0.3
```



```
git checkout a67061d
```

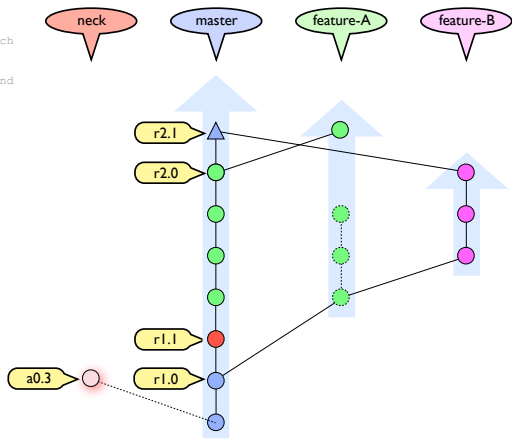
Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:
git checkout -b <new_branch_name>
HEAD is now at a67061d... add boo



```
git checkout a67061d
```

Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:

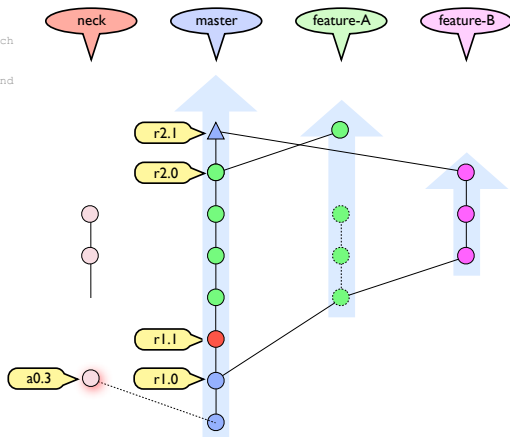
```
git checkout -b <new_branch_name>  
HEAD is now at a67061d... add boo
```



```
git checkout a67061d
Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:
  git checkout -b <new_branch_name>
HEAD is now at a67061d... add boo
```

...work work work...

```
git commit -a -m 'this is cool'
[detached HEAD a67061d] add foo.txt
git status
# Not currently on any branch.
nothing to commit (working directory clean)
Exit 1
git checkout -b neck
Switched to a new branch 'neck'
git status
# On branch neck
nothing to commit (working directory clean)
```



```
git checkout a67061d
```

Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:

```
git checkout -b <new_branch_name>  
HEAD is now at a67061d... add boo
```

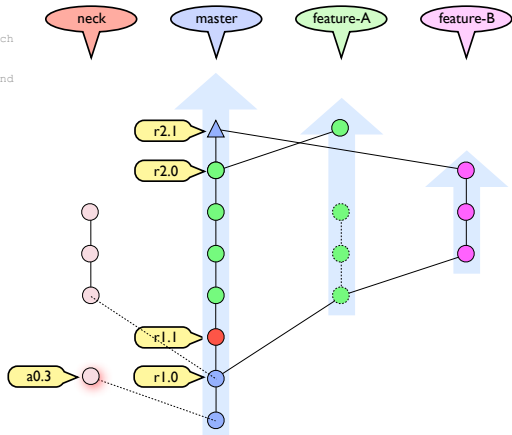
...work work work...

```
git commit -a -m 'this is cool'  
[detached HEAD a67061d] add foo.txt
```

```
git status  
# Not currently on any branch.  
nothing to commit (working directory clean)  
Exit 1
```

```
git checkout -b neck  
Switched to a new branch 'neck'
```

```
git status  
# On branch neck  
nothing to commit (working directory clean)
```



```
git checkout a67061d
```

Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:

```
git checkout -b <new_branch_name>  
HEAD is now at a67061d... add boo
```

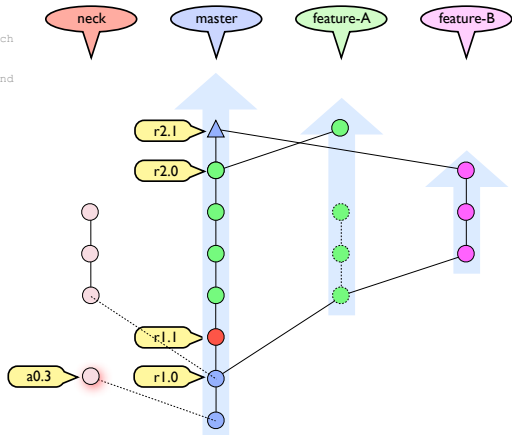
...work work work...

```
git commit -a -m 'this is cool'  
[detached HEAD a67061d] add foo.txt
```

```
git status  
# Not currently on any branch.  
nothing to commit (working directory clean)  
Exit 1
```

```
git checkout -b neck  
Switched to a new branch 'neck'
```

```
git status  
# On branch neck  
nothing to commit (working directory clean)
```



```
git checkout a67061d
```

Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:

```
git checkout -b <new_branch_name>  
HEAD is now at a67061d... add boo
```

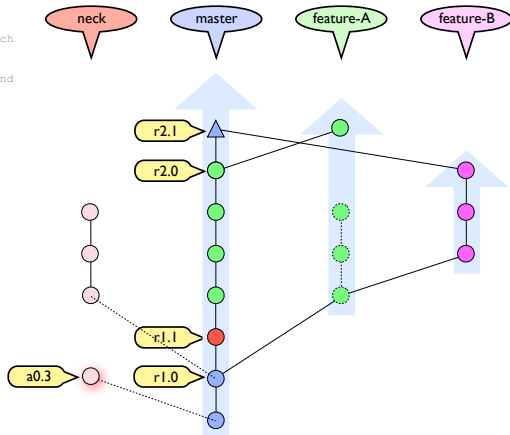
...work work work...

```
git commit -a -m 'this is cool'  
[detached HEAD a67061d] add foo.txt
```

```
git status  
# Not currently on any branch.  
nothing to commit (working directory clean)  
Exit 1
```

```
git checkout -b neck  
Switched to a new branch 'neck'
```

```
git status  
# On branch neck  
nothing to commit (working directory clean)
```



```
git checkout a67061d
```

Note: moving to 'a67061d' which isn't a local branch
If you want to create a new branch from this
checkout, you may do so
(now or later) by using -b with the checkout command
again. Example:

```
git checkout -b <new_branch_name>  
HEAD is now at a67061d... add boo
```

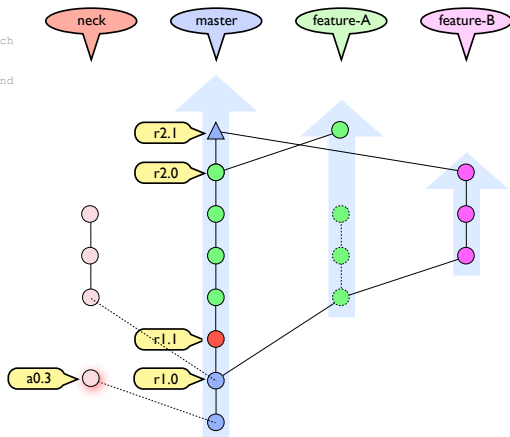
...work work work...

```
git commit -a -m 'this is cool'  
[detached HEAD a67061d] add foo.txt
```

```
git status  
# Not currently on any branch.  
nothing to commit (working directory clean)  
Exit 1
```

```
git checkout -b neck  
Switched to a new branch 'neck'
```

```
git status  
# On branch neck  
nothing to commit (working directory clean)
```

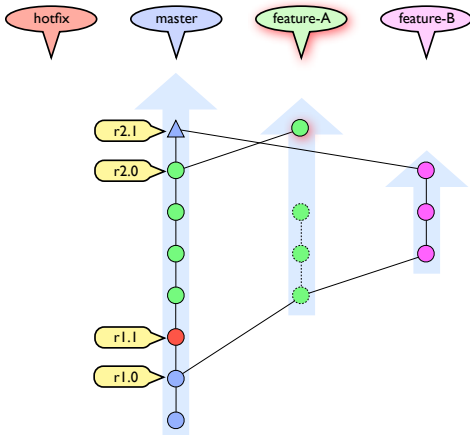


Multiple Repositories

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a r1.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'
git checkout feature-A
git commit -a -m 'feature-A extension'

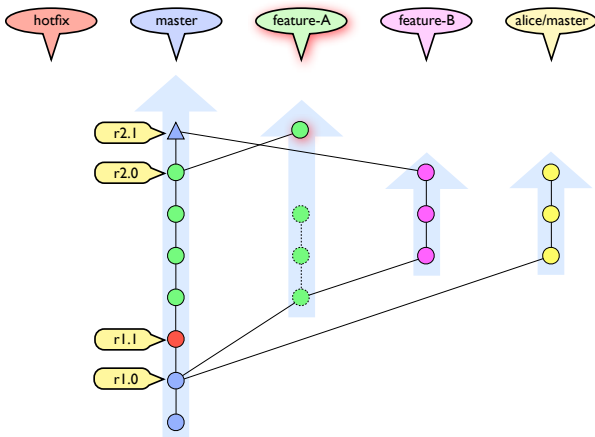
```



```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a r1.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'
git checkout feature-A
git commit -a -m 'feature-A extension'
git fetch alice

```

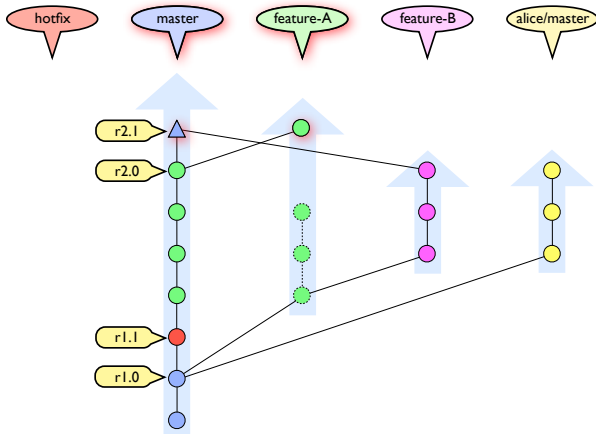


Fetching from a remote repository doesn't change your commit tree!

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a r1.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'
git checkout feature-A
git commit -a -m 'feature-A extension'
git fetch alice
git checkout master

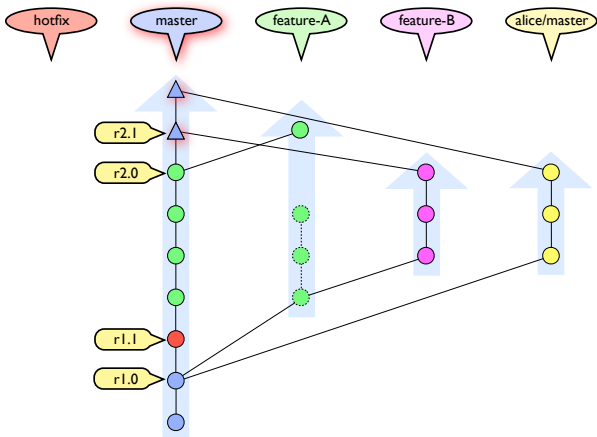
```



```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a r1.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'
git checkout feature-A
git commit -a -m 'feature-A extension'
git fetch alice
git checkout master
git merge alice/master
...resolve conflicts...
git commit -a -m 'merge alice/master into master'

```

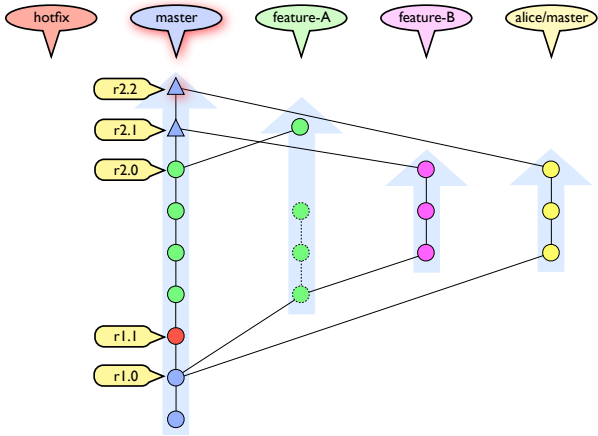


git pull combines fetch and merge!

```

git checkout master
git checkout -b feature-A
git commit -a -m 'basic feature A structure'
git checkout -b feature-B
git commit -a -m 'basic feature B structure'
git commit -a -m 'debug feature B'
git checkout feature-A
git commit -a -m 'finish feature A'
git commit -a -m 'debug feature A'
git checkout -b hotfix r1.0
git commit -a -m 'keep customer happy'
git checkout master
git merge hotfix
git tag -a r1.1 -m 'security update'
git checkout feature-A
git rebase master
...resolve conflicts...
git rebase --continue
git commit -a -m 'polish feature A'
git checkout master
git merge feature-A
git tag -a 2.0 -m 'new and improved release'
git checkout feature-B
git commit -a -m 'polish feature-B'
git checkout master
git merge feature-B
...resolve conflicts...
git commit -a -m 'merge feature-B into master'
git tag -a r2.1 -m 'wow release'
git checkout feature-A
git commit -a -m 'feature-A extension'
git checkout master
git fetch alice
git merge alice/master
...resolve conflicts...
git commit -a -m 'merge alice/master into master'
git tag -a r2.2 -m 'insecurity update'

```



Publishing your repository

```
local ~/project/ > ssh me@remote.com
```

```
remote ~/ > mkdir project.git
```

```
remote ~/project.git/ > cd project.git
```

```
remote ~/project.git/ > git init --bare
```

```
remote ~/project.git/ > logout
```

```
local ~/project/ > git remote add public_repo ssh://me@remote.com/~/project.git
```

```
local ~/project/ > git push public_repo release_branch
```

you should NOT publish your private directories (basic security)
create a **bare repository** on a public server
push only the branches your wish to publish

USB-Stick

```
~/project/ > git clone --bare . /Volumes/usb_stick/project.git
```

```
~/project/ > git remote add usb_stick /Volumes/usb_stick/project.git
```

```
~/project/ > git push usb_stick
```

USB stick, external disk

great for ad-hoc sharing

great for backup

treat like a public
repository

Which Repos Am I connected to?

```
~/project_dir/ > git remote -v  
public_repo      ssh://me@remote.com/~project.git (fetch)  
public_repo      ssh://me@remote.com/~project.git (push)  
usb_stick        /Volumes/usb_stick/project.git (fetch)  
usb_stick        /Volumes/usb_stick/project.git (push)
```

Updates From Multiple Repos

```
~/project/ > $EDITOR .git/config

...
[remote "steve"]
    url = ssh://steveserve.com/~Git/project.git
    fetch = +refs/heads/*:refs/remotes/steve/*
[remote "mac"]
    url = git@github.com:mac/project.git
    fetch = +refs/heads/*:refs/remotes/mac/*
[remotes]
    buddies = steve mac
...

~/project/ > git remote update buddies
Updating steve
...
Updating mac
...
```

Working With Others

Publish your changes via a bare repository

Never push to someone else's repository

Use `git remote update` to track multiple repositories

Use `git show-branch` or `git whatchanged` to see what's new

Rewriting History

What if you want to...

...find the commit that introduced a problem...

...remove some commits from the history...

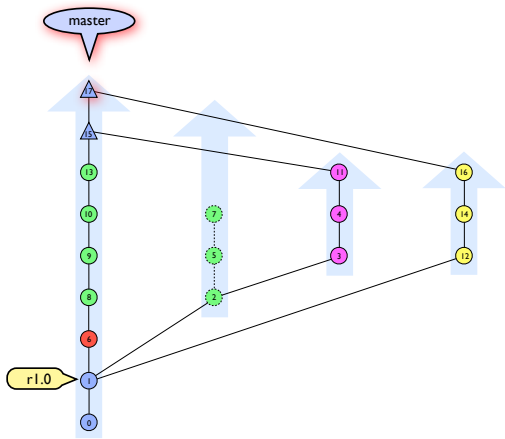
...add one or more commits from one branch to another...

...work on a branch for a long time...

finding bad commits

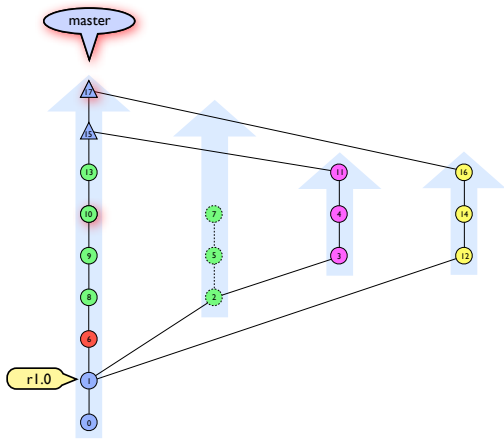
git bisect

`git checkout master`




```
git checkout master
git bisect start
git bisect bad master
git bisect good rl.0
```

*Bisecting: ## revisions left to test after this
[10] commit message*



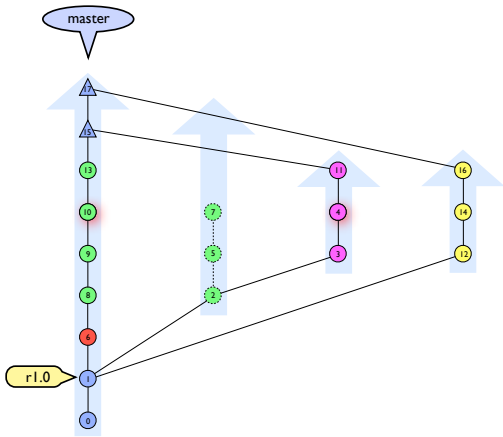
git has checked out a commit for you to test...
Is the problem currently checked out?
If not call `git bisect good`

```
git checkout master
git bisect start
git bisect bad master
git bisect good rl.0
```

```
Bisecting: ## revisions left to test after this
[10] commit message
```

```
git bisect good
```

```
Bisecting: ## revisions left to test after this
[4] commit message
```



git has checked out another commit for you to test...
Can't test this version? (doesn't compile?)
If so call `git bisect skip`

```
git checkout master
git bisect start
git bisect bad master
git bisect good rl.0
```

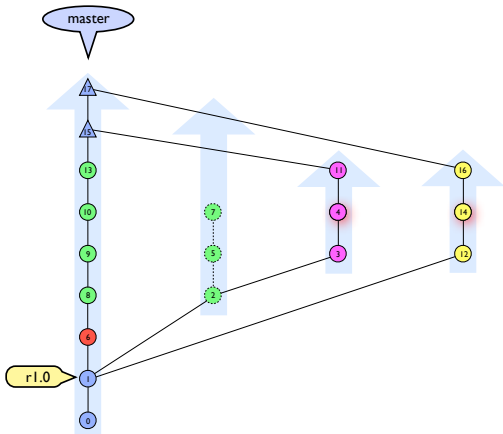
```
Bisecting: ## revisions left to test after this
[10] commit message
```

```
git bisect good
```

```
Bisecting: ## revisions left to test after this
[4] commit message
```

```
git bisect skip
```

```
Bisecting: ## revisions left to test after this
[14] commit message
```



Another commit for you to test...
Is the problem currently checked out?
If so call `git bisect bad`

```
git checkout master
git bisect start
git bisect bad master
git bisect good rl.0
```

```
Bisecting: ## revisions left to test after this
[10] commit message
```

```
git bisect good
```

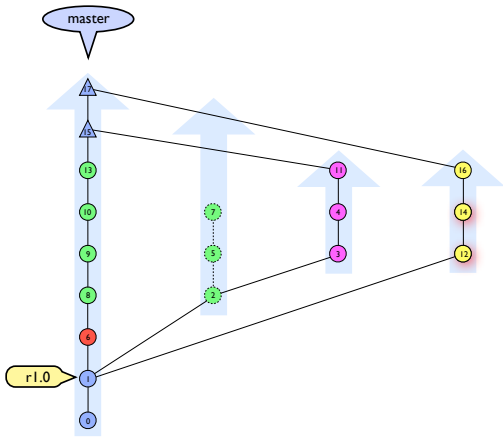
```
Bisecting: ## revisions left to test after this
[4] commit message
```

```
git bisect skip
```

```
Bisecting: ## revisions left to test after this
[14] commit message
```

```
git bisect bad
```

```
12 is the first bad commit
```



Now you know where the problem is.
Go back to your branch and fix it with a normal commit.

```
git checkout master
git bisect start
git bisect bad master
git bisect good rl.0
```

```
Bisecting: ## revisions left to test after this
[10] commit message
```

```
git bisect good
```

```
Bisecting: ## revisions left to test after this
[4] commit message
```

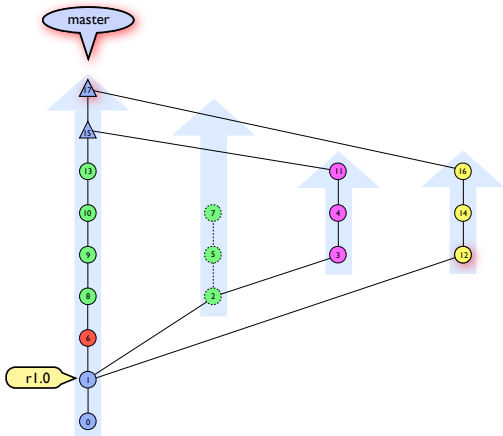
```
git bisect skip
```

```
Bisecting: ## revisions left to test after this
[14] commit message
```

```
git bisect bad
```

12 is the first bad commit

```
git bisect reset
```



```
git checkout master
git bisect start
git bisect bad master
git bisect good rl.0
```

```
Bisecting: ## revisions left to test after this
[10] commit message
```

```
git bisect good
```

```
Bisecting: ## revisions left to test after this
[4] commit message
```

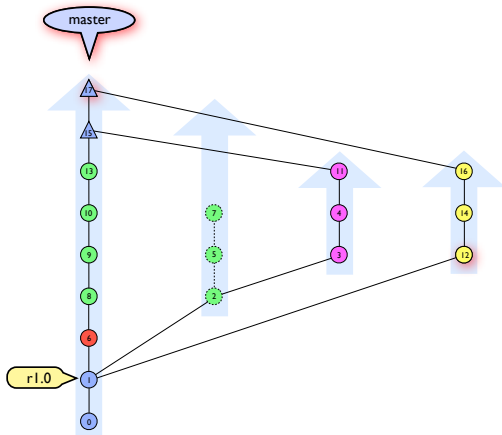
```
git bisect skip
```

```
Bisecting: ## revisions left to test after this
[14] commit message
```

```
git bisect bad
```

```
12 is the first bad commit
```

```
git bisect reset
```



git bisect automation

```
git bisect start bad_commit good_commit
```

```
git bisect run test_script options...
```

Test script exit codes:

```
exit 0      => good
```

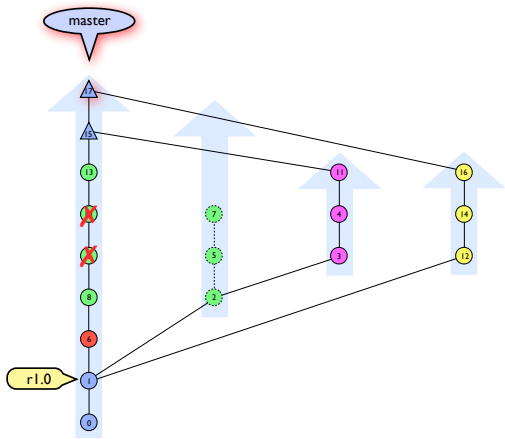
```
exit 125    => skip
```

```
exit 1 .. 127 => bad
```

removing bad commits

interactive rebasing

```
git checkout master  
git rebase --interactive r1.0
```



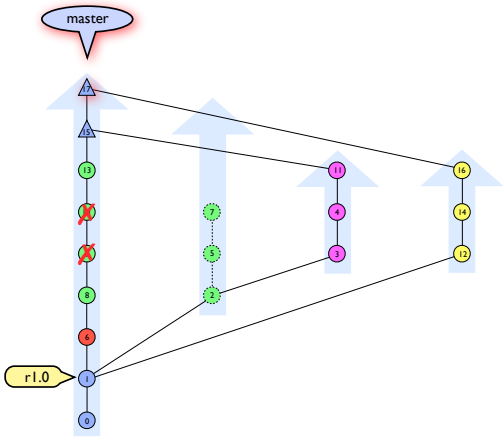
```
pick ca4f103 6 hotfix
pick 1f85820 8 feature_A - first try
pick 9b6e08e 9 feature_A - with signature
pick 7d86f88 10 feature_A with more detail
pick e29b897 2 feature_A - first try
pick 39f4215 3 first attempt at feature B
pick f4449ad 4 feature_B comments
pick 27c2b4c 11 feature_B fix wrong spell world
```

```
# Rebase 2aa3032..5af9beb onto 2aa3032
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Warning!

Newest commits are at the bottom!

(most other git output has the newest commit at the top!)



```

pick ca4f103 6 hotfix
pick 1f85820 8 feature_A - first try
pick 9b6e08e 9 feature_A - with signature
pick 7d86f88 10 feature_A with more detail
pick e29b897 2 feature_A - first try
pick 39f4215 3 first attempt at feature B
pick f4449ad 4 feature_B comments
pick 27c2b4c 11 feature_B fix wrong spellt world

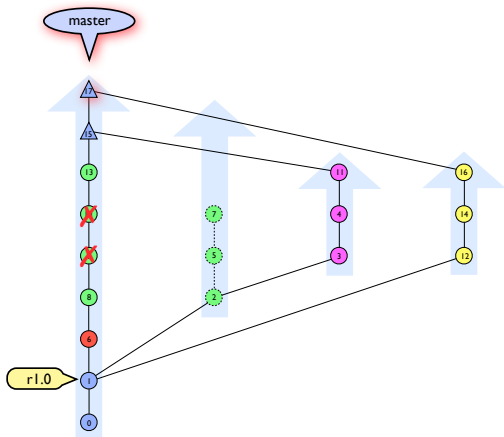
```

```

# Rebase 2aa3032...5af9beb onto 2aa3032
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#

```

Delete the lines of commits you don't want.
 Change pick to squash if you want merge commits,
 or edit if you want to split a commit into smaller commits.



```

pick ca4f103 6 hotfix
pick 1f85820 8 feature_A - first try

pick e29b897 2 feature_A - first try
pick 39f4215 3 first attempt at feature B
pick f4449ad 4 feature_B comments
pick 27c2b4c 11 feature_B fix wrong spellt world

```

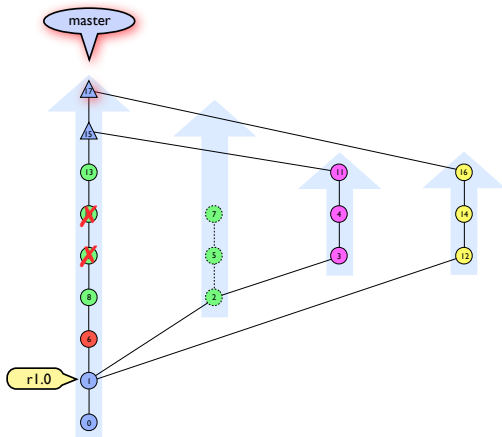
```

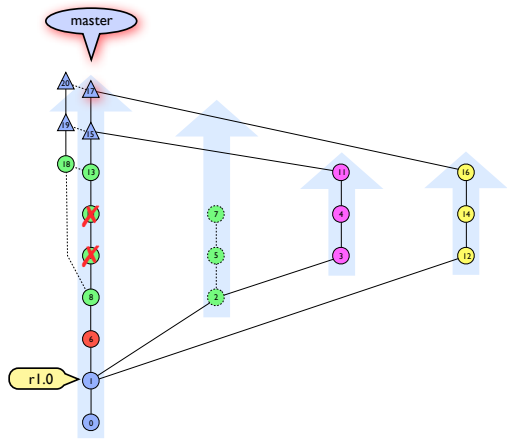
# Rebase 2aa3032...5af9beb onto 2aa3032
#
# Commands:
# p, pick = use commit
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#

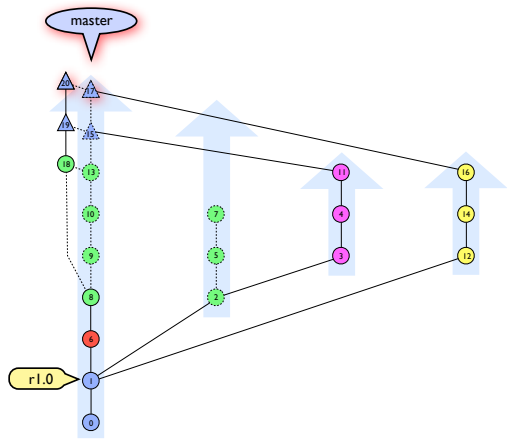
```

save the file and exit your editor...

git performs the rebase automatically







adding commits to other branches

cherry picking

Add just one commit to the current branch:

```
git cherry-pick sha1
```

rebasing onto another branch

Add a chain of commits, not the whole branch:

```
git rebase --onto target_commit first_commit last_commit
```

Many thanks to:

Patrick Stein (Venue)

Matthias Carell (Esteemed Critic)

CocoaHeads Munich (Avid Listeners)

