

Basic Git Tutorial

Yue Wang

Parts of the material are credit to Ding Qian

Version Control System

- Better control of your source code with Git
 - Clear picture of your development
 - Never worried about writing wrong code
 - Easier coordination with others
 - <http://git-scm.com/>

Installation

- Linux (Ubuntu)
 - `sudo apt-get install git`
- Mac
 - Install `homebrew` first from <http://brew.sh>
 - `brew install git`
- Windows
 - Install `Cygwin` first, then you have a UNIX shell
 - Install git for windows (<http://gitforwindows.org/>)
- Your own way...

Initialization

- Create a local git repository : **git init**

```
qding at shb118 in ~/Workspace/csci4140
$ git init demo
Initialized empty Git repository in /home/qding/Workspace/csci4140/demo/.git/

qding at shb118 in ~/Workspace/csci4140
$ cd demo

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ █
```

same as

```
qding at shb118 in ~/Workspace/csci4140
$ mkdir demo

qding at shb118 in ~/Workspace/csci4140
$ cd demo

qding at shb118 in ~/Workspace/csci4140/demo
$ git init
Initialized empty Git repository in /home/qding/Workspace/csci4140/demo/.git/
```

Working with Git

- **git status**

- Check the status of your git repository
- Use this command before critical changes like commit or push!!!

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

Working with Git

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ vim hello.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ ls
hello.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ make hello
cc      hello.c      -o hello

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ ls
hello hello.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ ./hello
Hello World
```

```
// hello.c
#include <stdlib.h>
#include <stdio.h>

int main()
{
    printf ("Hello World\n");
    return 0;
}
```

Working with Git

- The status of each file
 - unstaged/untracked: file change not tracked by git
 - tracked/staged: file change tracked by git

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       hello
#       hello.c
nothing added to commit but untracked files present (use "git add" to track)
```

Working with Git

- `git add <file_path>`
 - To add `unstaged/untracked` file to git repository
 - `hello.c` become `tracked/staged` now

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git add hello.c

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   hello.c
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       hello
```


Working with Git

- `git add <dir_path>`
 - Add all file (recursively) in the directory to git repository
 - hello.c and hello become **tracked/staged** now

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git add .

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   hello
#       new file:   hello.c
#
```

Working with Git

- `git commit -m "commit message"`
 - Make the changes persistent as a commit

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git commit -m "this is the first commit"
[master (root-commit) 05247e3] this is the first commit
2 files changed, 9 insertions(+)
create mode 100755 hello
create mode 100644 hello.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
nothing to commit, working directory clean
```

Working with Git

- **git log**
 - Check your commit history

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git log
commit 05247e3272450e5e22a8d91c46de26fd85cd9b1f
Author: DING QIAN <qding@cse.cuhk.edu.hk>
Date: Tue Jan 7 11:53:58 2014 +0800

    this is the first commit
```

commit
hash key

author and
date info

commit
message

Working with Git

- `git mv <old_file> <new_file>`
 - Rename `tracked` file or move to another directory
- `git rm <file_path>`
 - Delete `tracked` file

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git mv hello.c sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git rm hello
rm 'hello'

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:      hello
#       renamed:      hello.c -> sample.c
#
```

Similar to `git add`, the changes made by `git mv/rm` is `staged` by default, you can commit them directly

Working with Git

- Commit again, now we have two commits

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git commit -m "rename srouce and delete exec"
[master 7934b48] rename srouce and delete exec
2 files changed, 0 insertions(+), 0 deletions(-)
delete mode 100755 hello
rename hello.c => sample.c (100%)

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git log
commit 7934b483a721aa649810b6ec00822f0f086688a1
Author: DING QIAN <qding@cse.cuhk.edu.hk>
Date: Tue Jan 7 12:05:06 2014 +0800

    rename srouce and delete exec

commit 05247e3272450e5e22a8d91c46de26fd85cd9b1f
Author: DING QIAN <qding@cse.cuhk.edu.hk>
Date: Tue Jan 7 11:53:58 2014 +0800

    this is the first commit
```

Working with Git

- Let's modify sample.c

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ vim sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   sample.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
// hello.c
#include <stdlib.h>
#include <stdio.h>

int main()
{
    printf ("Hello World\n");
    printf ("Hello CUHK\n");
    return 0;
}
```

Similar to create a new file, after modifying a existing file in the repository, the change is **unstaged** by default, you need to use **git add** again to let git know you want to stage the change.

Working with Git

- Let's commit the modified sample.c

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git add sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   sample.c
#

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git commit -m "add one more printf in sample.c"
[master 0d4d4fe] add one more printf in sample.c
1 file changed, 1 insertion(+)
```

Working with Git

- Sometimes, you made some changes to your code, but you regretted. So how to rollback?
 - Case 1: the changed is **unstaged**

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ vim sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   sample.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```


Working with Git

- Sometimes, you made some changes to your code, but you regretted. So how to rollback?
 - Case 1: the changed is **unstaged**
 - **git checkout -- <file_path>**
 - This let you rollback to the latest commit

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git checkout -- sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
nothing to commit, working directory clean
```

Working with Git

- Sometimes, you made some changes to your code, but you regretted. So how to rollback?
 - Case 2: the changed is **staged**

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ vim sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git add sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
# Changes to be committed
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   sample.c
#
```

Working with Git

- Sometimes, you made some changes to your code, but you regretted. So how to rollback?
 - Case 2: the changed is **staged**
 1. `git reset HEAD <file_path>`
 2. `git checkout -- <file_path>`
 - You need one additional step to **unstage** first

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git reset HEAD sample.c
Unstaged changes after reset:
M      sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git checkout -- sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
nothing to commit, working directory clean
```

What if I don't
unstage???
Try it yourself.

Working with Git

- Sometimes, you made some changes to your code, but you regretted. So how to rollback?
 - Case 3: You want to rollback to previous commit, not the latest one

1. `git checkout <commit_hash> <file_path>`

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git log
commit 0d4d4fe6bc18ea95641232528faf4c7eea5944cb
Author: DING QIAN <qding@cse.cuhk.edu.hk>
Date: Tue Jan 7 13:04:48 2014 +0800
```

add one more printf in sample.c

```
commit 7934b483a721aa649810b6ec00822f0f086688a1
Author: DING QIAN <qding@cse.cuhk.edu.hk>
Date: Tue Jan 7 12:05:06 2014 +0800
```

rename srouce and delete exec

```
commit 05247e3272450e5e22a8d91c46de26fd85cd9b1f
Author: DING QIAN <qding@cse.cuhk.edu.hk>
Date: Tue Jan 7 11:53:58 2014 +0800
```

this is the first commit

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git checkout 7934b483a721aa649810b6ec00822f0f086688a1 sample.c
```

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
```

```
$ cat sample.c
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf ("Hello World\n");
```

```
    return 0;
```

```
}
```

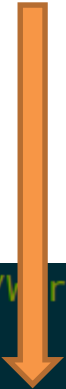
Working with Git

- Sometimes, you made some changes to your code, but you regretted. So how to rollback?
 - Case 3: You want to rollback to previous commit, not the latest one
 - By checkout, the changes are staged already

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git checkout 7934b483a721aa649810b6ec00822f0f086688a1 sample.c
```

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ cat sample.c
#include <stdlib.h>
#include <stdio.h>
```

```
int main()
{
    printf ("Hello World\n");
    return 0;
}
```

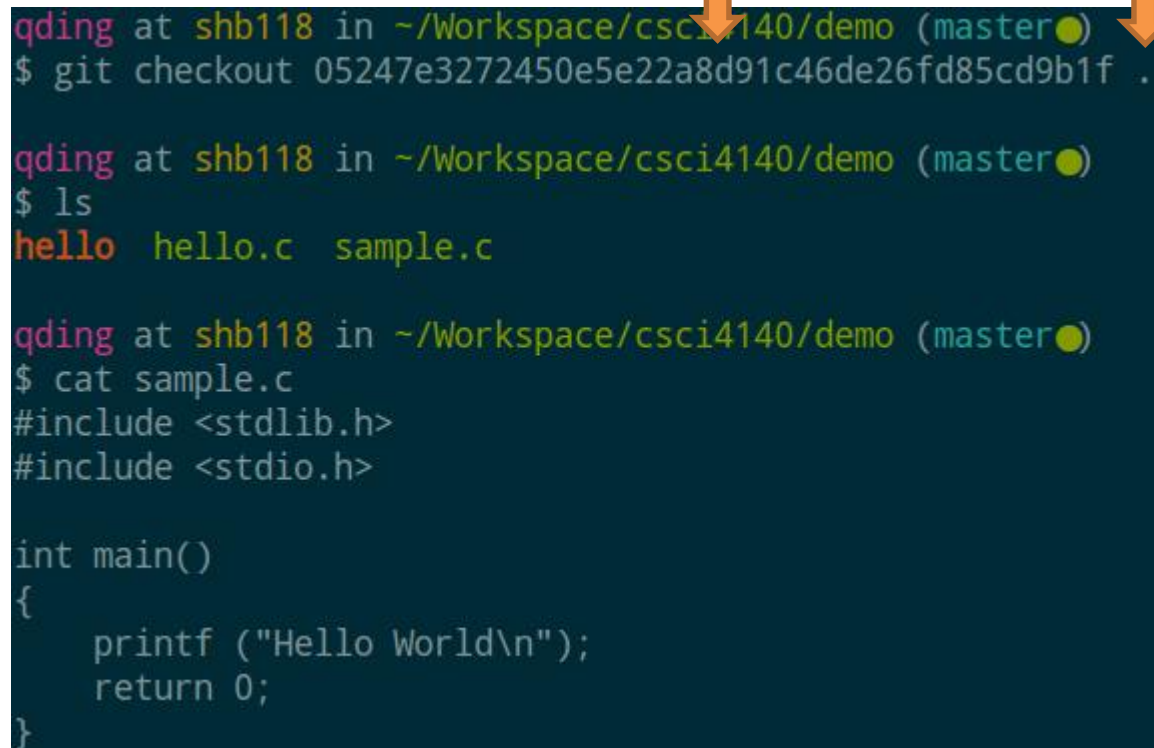


```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   sample.c
#
```

Working with Git

- Sometimes, you want to rollback your entire directory to a previous commit

– `git checkout <commit_hash> <dir>`



```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git checkout 05247e3272450e5e22a8d91c46de26fd85cd9b1f .

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ ls
hello hello.c sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ cat sample.c
#include <stdlib.h>
#include <stdio.h>

int main()
{
    printf ("Hello World\n");
    return 0;
}
```

Don't forget
this dot, it
specifies the
directory you
want to rollback

Synchronize with remote server

- Clone a remote git repository tracking central server: **git clone <url>**

```
qding at shb118 in ~/Workspace/csci4140
$ git clone ssh://52ca85355973ca990000042e@php-zie.rhcloud.com/~/.git/php.git/
Cloning into 'php'...
remote: Counting objects: 25, done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 25 (delta 1), reused 20 (delta 0)
Receiving objects: 100% (25/25), 19.48 KiB, done.
Resolving deltas: 100% (1/1), done.

qding at shb118 in ~/Workspace/csci4140
$ cd php

qding at shb118 in ~/Workspace/csci4140/php (master)
$ ls
libs  misc  php  deplist.txt  README

qding at shb118 in ~/Workspace/csci4140/php (master)
$ █
```

Synchronize with remote server

- Once you commit your changes, you can push your local commits to the remote server by
– **git push**

```
qding at shb118 in ~/Workspace/csci4140/php (master)
$ git push
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 372 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Not stopping cartridge zend because hot deploy is enabled
remote: Building git ref 'master', commit bcef8a4
remote: Preparing build for deployment
remote: Deployment id is 719a42d6
remote: Activating deployment
remote: Not starting cartridge zend because hot deploy is enabled
remote: Result: success
remote: Activation status: success
remote: Deployment completed with status: success
To ssh://52ca85355973ca990000042e@php-zie.rhcloud.com/~/.git/php.git/
a84e249..bcef8a4  master -> master
```


Synchronize with remote server

- If another developer also tracking this repo, he can get your update via

– `git pull`

```
qding at shb118 in /tmp/php (master)
$ git pull
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
Unpacking objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
From ssh://php-zie.rhcloud.com/~git/php
     bcef8a4..62ca251  master      -> origin/master
Updating bcef8a4..62ca251
Fast-forward
 php/index.php | 2 ++
 1 file changed, 2 insertions(+)
```

Advanced Techniques

- `git diff <file_path>`
 - Get the difference between your working copy (unstaged) and the staged copy
 - You can try `git add <file_path>` and then `git diff <file_path>` to see what will happen

```
qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git diff sample.c
diff --git a/sample.c b/sample.c
index 481e18e..3923151 100644
--- a/sample.c
+++ b/sample.c
@@ -4,7 +4,7 @@
 int main()
 {
     printf ("Hello World\n");
-    printf ("Hello CUHK\n");
+    printf ("Hello CUHK CSE\n");
     return 0;
 }
```

Advanced Techniques

- **.gitignore file**
 - Let git ignores and never tracks particular file types or files
 - Create a hidden file named “.gitignore” under your repository
 - Each line in the .gitignore file named a type or a file that will be ignored

Advanced Techniques

- **.gitignore file**

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
```

```
$ ls
```

```
sample  sample.c
```

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
```

```
$ git status
```

```
# On branch master
```

```
# Untracked files:
```

```
#   (use "git add <file>..." to include in what will be committed)
```

```
#
```

```
#       sample
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
```

```
$ echo "sample" > .gitignore
```

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
```

```
$ git status
```

```
# On branch master
```

```
# Untracked files:
```

```
#   (use "git add <file>..." to include in what will be committed)
```

```
#
```

```
#       .gitignore
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

Advanced Techniques

- **.gitignore file**

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git add .gitignore

qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git commit -m "add .gitignore file"
[master 58f2118] add .gitignore file
1 file changed, 1 insertion(+)
create mode 100644 .gitignore

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ ls
sample sample.c

qding at shb118 in ~/Workspace/csci4140/demo (master)
$ git status
# On branch master
nothing to commit, working directory clean
```

```
./.gitignore file
*.o
img/
sample
*.c
```

Advanced Techniques

- Config your git with your username, email, customized log style
 - `git config --global user.name "John"`
 - `git config --global user.email "abc@example.com"`
 - `git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --"`

```
qding at shb118 in ~/Workspace/csci4140/demo (master●)
$ git lg
* 0d4d4fe - (HEAD, master) add one more printf in sample.c (66 minutes ago) <DING QIAN>
* 7934b48 - rename srouce and delete exec (2 hours ago) <DING QIAN>
* 05247e3 - this is the first commit (2 hours ago) <DING QIAN>
```

These short hash values can also
be used with `git checkout`

Basic Branching and Merging

Let's go through a simple example of branching and merging with a workflow that you might use in the real world. You'll follow these steps:

1. Do some work on a website.
2. Create a branch for a new story you're working on.
3. Do some work in that branch.

At this stage, you'll receive a call that another issue is critical and you need a hotfix. You'll do the following:

1. Switch to your production branch.
2. Create a branch to add the hotfix.
3. After it's tested, merge the hotfix branch, and push to production.
4. Switch back to your original story and continue working.

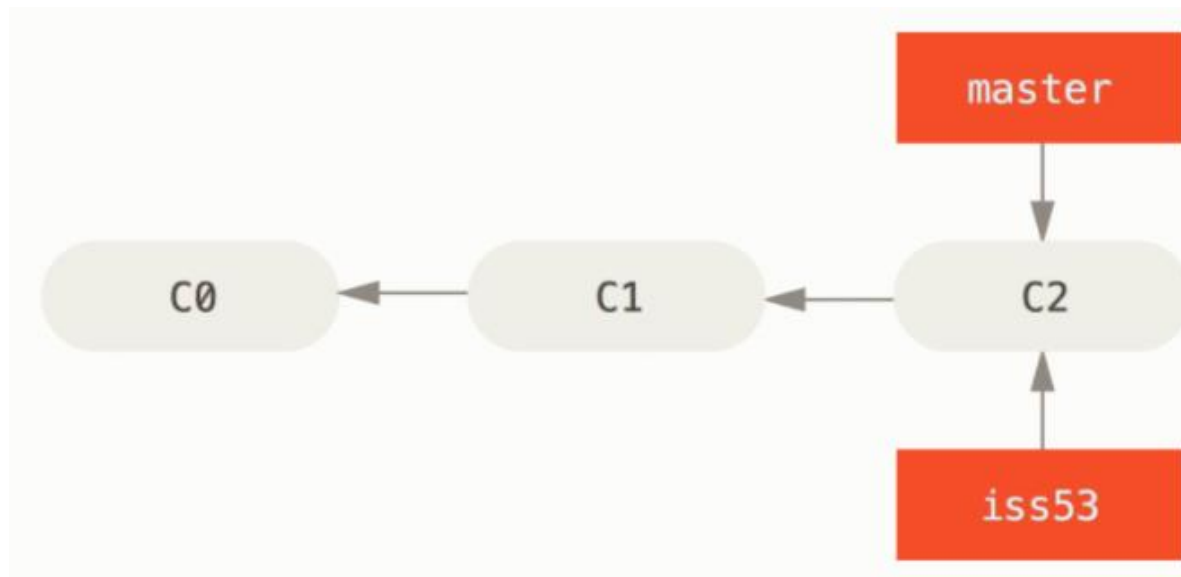
Basic Branching and Merging

- If you want to create a new branch and switch to it at the same time
 - `git checkout -b [branch name]`
 - `git branch [branch name]`, `git checkout [branch name]`

```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (master)
$ git checkout -b iss53
Switched to a new branch 'iss53'
```


Basic Branching and Merging

- Assume the current commit flow is as following:

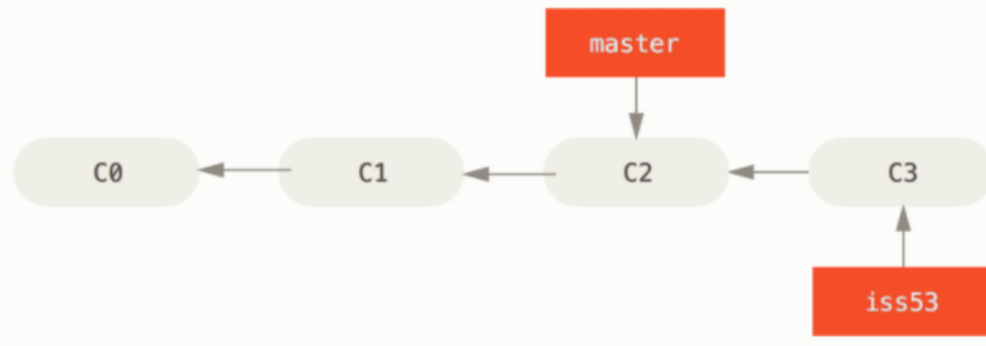


Basic Branching and Merging

```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ vim c3

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git add c3
warning: LF will be replaced by CRLF in c3.
The file will have its original line endings in your working directory.

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git commit -m "the fix in the branch [iss53], c3"
[iss53 a28457e] the fix in the branch [iss53], c3
1 file changed, 1 insertion(+)
create mode 100644 c3
```



```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git lg
* a28457e - (HEAD -> iss53) the fix in the branch [iss53], c3 (69 seconds ago) <yuewang>
* 047050f - (master) the third commit c2 (8 minutes ago) <yuewang>
* 92373f6 - the second commit c1 (8 minutes ago) <yuewang>
* 4a2f93e - the first commit c0 (9 minutes ago) <yuewang>
```

Basic Branching and Merging

- Return back to the master branch and create a new branch hotfix and work on it

```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git checkout master
Switched to branch 'master'

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (master)
$ git checkout -b 'hotfix'
Switched to a new branch 'hotfix'

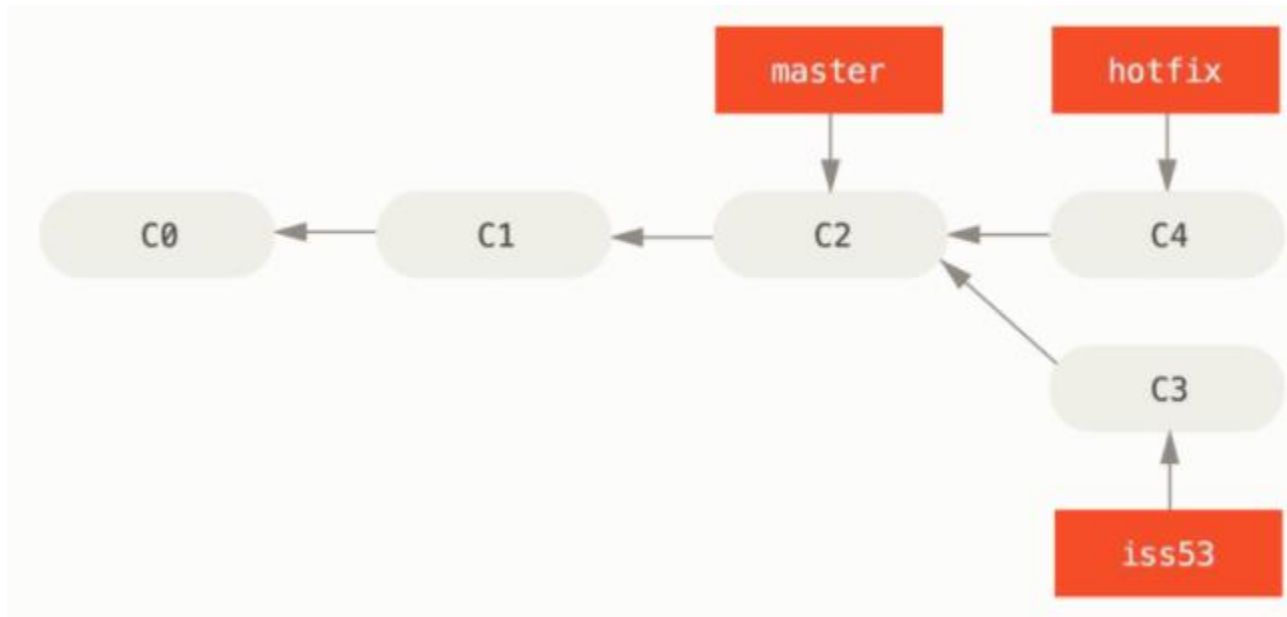
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (hotfix)
$ vim c4

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (hotfix)
$ git add c4
warning: LF will be replaced by CRLF in c4.
The file will have its original line endings in your working directory.

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (hotfix)
$ git commit -m "the commit in [hotfix], c4"
[hotfix ee2f22e] the commit in [hotfix], c4
1 file changed, 1 insertion(+)
create mode 100644 c4
```

Basic Branching and Merging

- Now the snapshot of the commit is like:



Basic Branching and Merging

- We want to merge the branch hotfix into the master branch
 - `git merge [branch name]`

No conflict!
Merge successfully

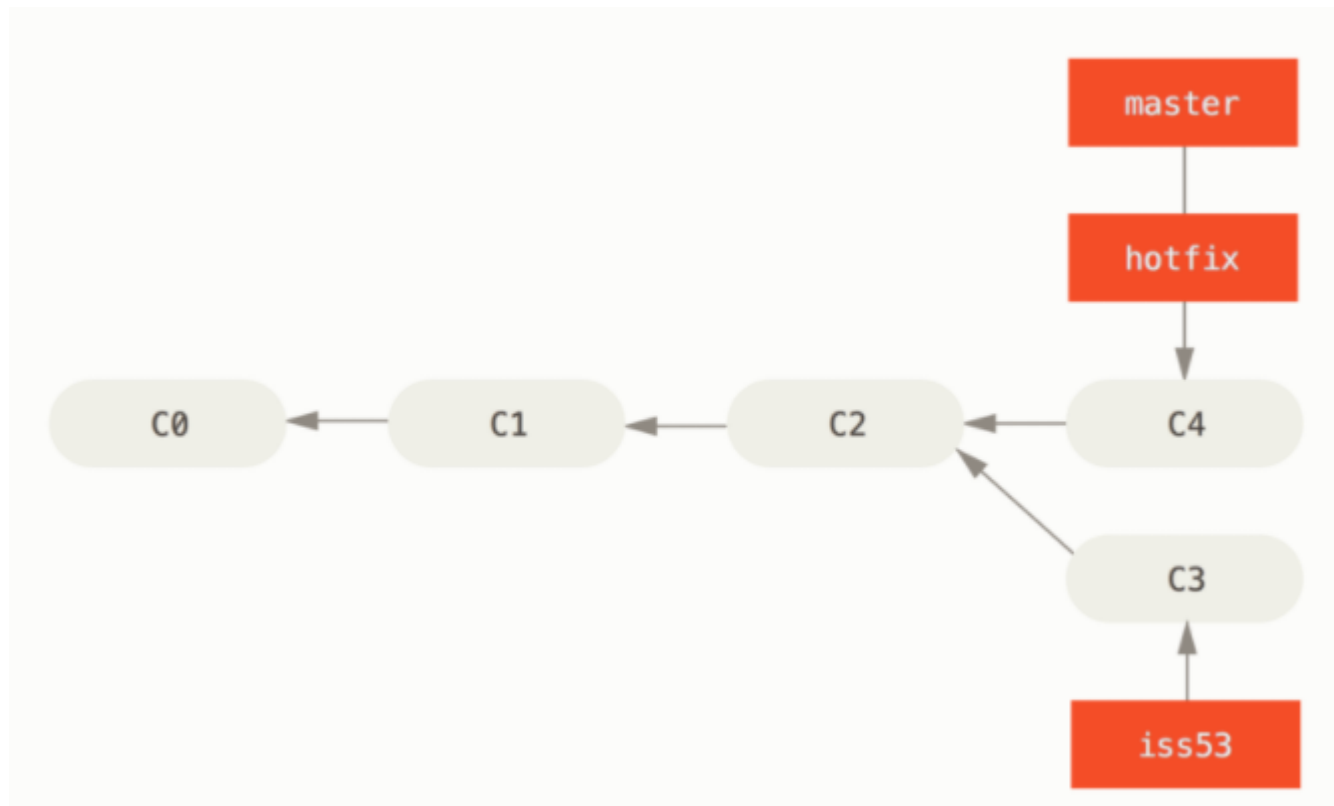


```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (hotfix)
$ git checkout master
Switched to branch 'master'

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (master)
$ git merge hotfix
Updating 047050f..ee2f22e
Fast-forward
 c4 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 c4
```

Basic Branching and Merging

- Now the snapshot of the commit is like:



Basic Branching and Merging

- We can delete the useless branch hotfix now
 - `git branch -d [branch name]`

```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (master)
$ git branch -d hotfix
Deleted branch hotfix (was ee2f22e).
```

Basic Branching and Merging

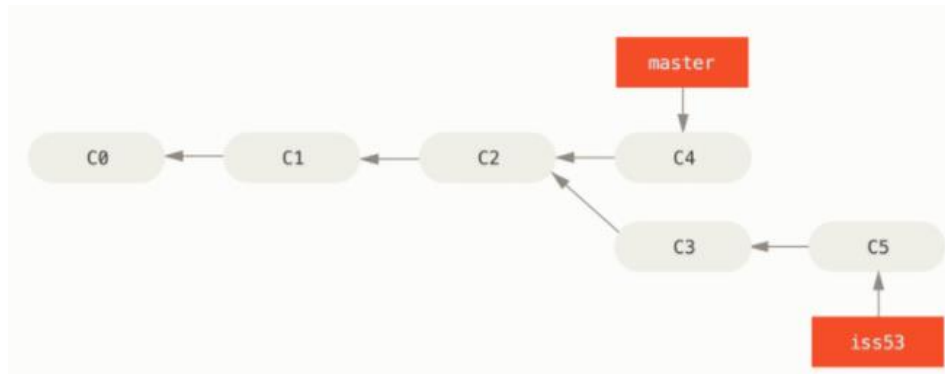
- Switch to iss53 and work on it

```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ vim c2

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git add c2
warning: LF will be replaced by CRLF in c2.
The file will have its original line endings in your working directory.

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git commit -m 'revise c2 in [iss53], c5'
[iss53 df0eace] revise c2 in [iss53], c5
1 file changed, 1 insertion(+)
```

- Now the snapshot of the commit is like:



Basic Branching and Merging

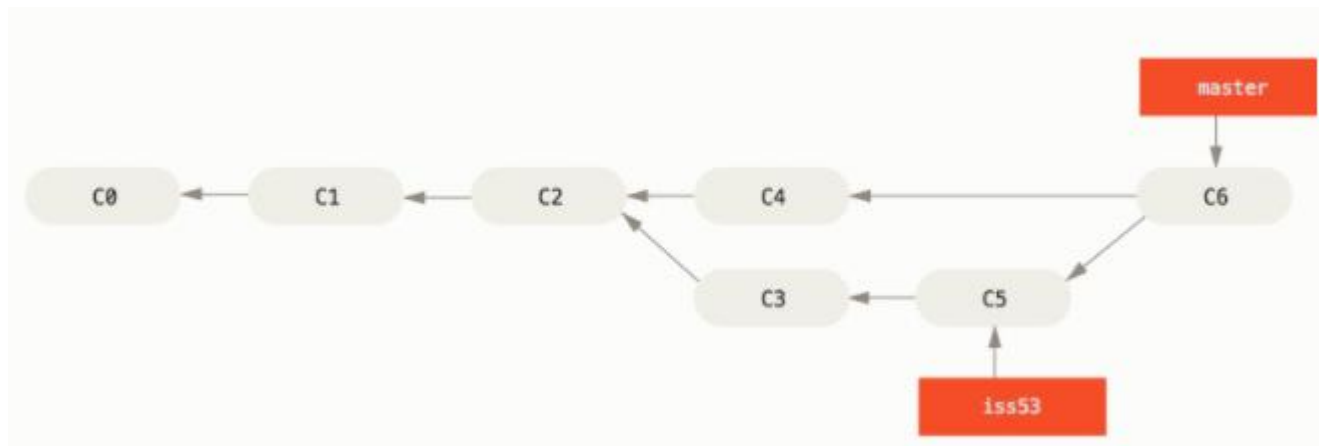
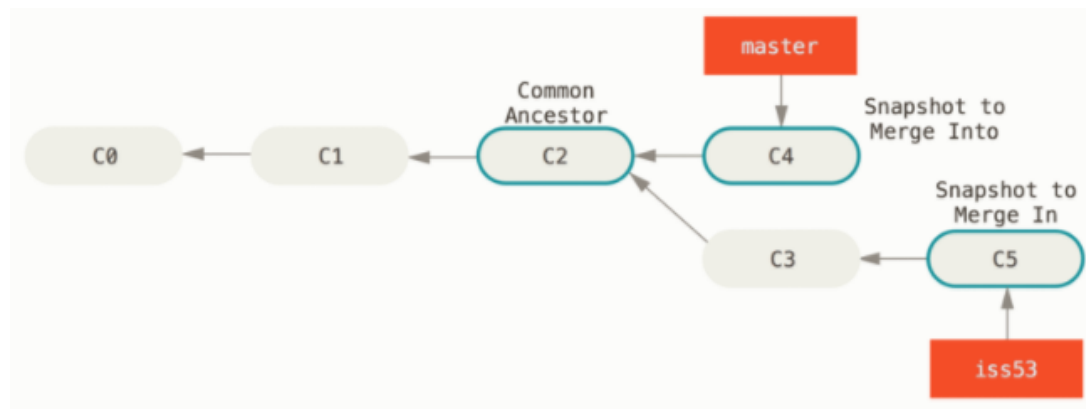
- Merge iss53 into the master branch

```
yuewang@PC90083 MINGW64 ~/workspace/branch_demo (iss53)
$ git checkout master
Switched to branch 'master'

yuewang@PC90083 MINGW64 ~/workspace/branch_demo (master)
$ git merge iss53
Merge made by the 'recursive' strategy.
 c2 | 1 +
 c3 | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 c3
```

Basic Branching and Merging

- The snapshot will be changed in the way as:



Basic Branching and Merging

- Conflicts may happen when merge
 - Assume the first commit in the master is to create a readme file as “a”
 - The second commit: create a branch[op1] to revise the readme as “b”
 - The third commit: return to master and revise the readme as “c”
 - Finally merge [op1] into master
- What will happen?

Basic Branching and Merging

- You need to resolve the conflict in person

```
yuewang@PC90083 MINGW64 ~/workspace/conflict_demo (master)
$ git merge op1
Auto-merging readme
CONFLICT (content): Merge conflict in readme
Automatic merge failed; fix conflicts and then commit the result.
```

```
yuewang@PC90083 MINGW64 ~/workspace/conflict_demo (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

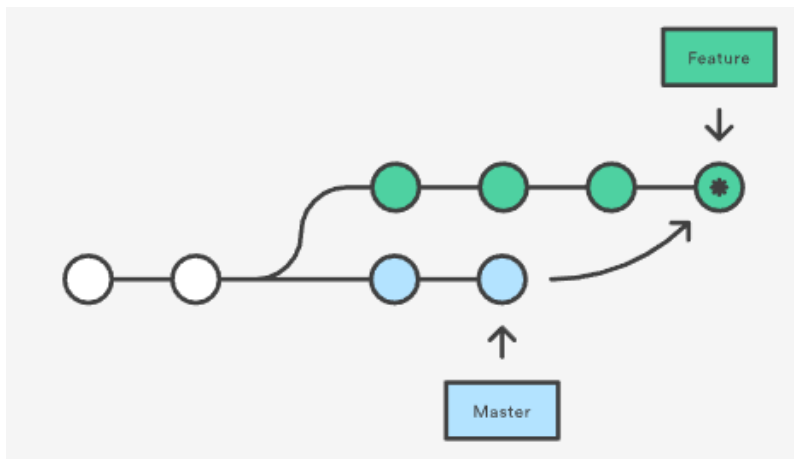
        both modified:   readme

no changes added to commit (use "git add" and/or "git commit -a")
```

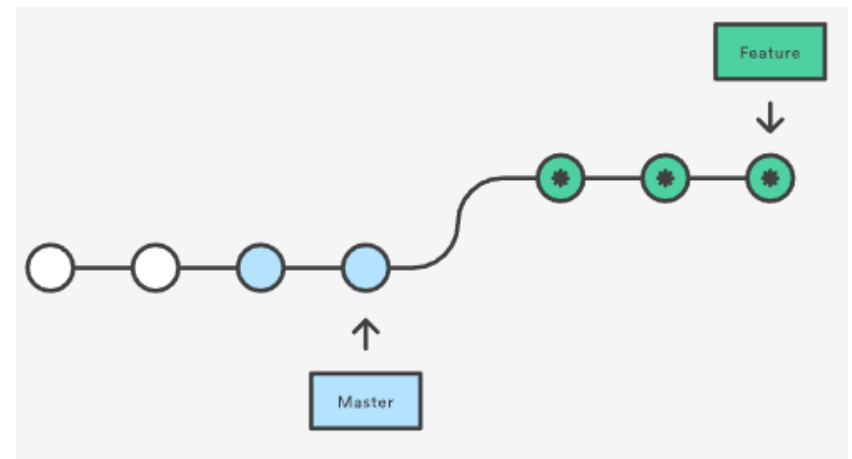
```
<<<<<<< HEAD
c
=====
b
>>>>>>> op1
~
```

Merging vs. Rebasing

- Tradeoff between safe and clean



merge



rebase

Advanced Techniques

- A successful Git branching model
 - <http://nvie.com/posts/a-successful-git-branching-model/>
- Basic branch and merge operation
 - <http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging>
- Git tutorial reference
 - <http://www.cs.columbia.edu/~sedwards/classes/2013/4840/git-tutorial.pdf>

What is github?

- www.github.com
- Largest web-based git repository hosting service
 - hosts ‘remote repositories’
- Allows for code collaboration with anyone online
- Adds extra functionality on top of git
 - UI, documentation, bug tracking, feature requests, pull requests, and more!
- Free public repository for everyone and free private repository for students



GitHub
Student Developer Pack