**GIT PEROSNAL NOTE**

*Author: Minping Yang*

* **Install Git first:**

1. For Linux

$ git

The program 'git' is currently not installed. You can install it by typing:

sudo apt-get install git

For Ubuntu Linux

$ sudo apt-get install git

If the version of Debian or Ubuntu Linux is too old

$ sudo apt-get install git

2.For mac

1st way: use [homebrew](http://brew.sh/) install git

2nd way: use Xcode (down load from apple store)

then, “Xcode”->“Preferences” --- choose “DOWNLOADS”in pop-up window, choose “Command Line Tools”, and click install

3.For windows

use [msysgit](https://git-for-windows.github.io/) to download windows version of Git

how to open the application?

Start – Git – Git Bash

Finally configure personal info by using command line

$git config --global user.name "Your Name"

$git config --global user.email [email@example.com](mailto:email@example.com)

\*global means that configure the computer

* **Create a local repository**

$ mkdir learngit

$ cd learngit

$ git init

Initialized empty Git repository in /Users/yangminp/learngit/.git/

##If .git directory was hidden,

$ ls –ah

create a readme.txt

$ git add readme.txt

$ git commit -m "wrote a readme file"

[master (root-commit) cb926e7] wrote a readme file

1 file changed, 2 insertions(+)

create mode 100644 readme.txt

* **Time Machine**

If you made some changes, using

$ git status

if you forgot what the changes we made before

$ git diff readme.txt

once we confirmed changes we made, then

$ git add readme.txt (Or add . means add all changes)

git commit –m “update readme.txt”

* **Change back the Version**

Let us check the log of changes of readme.txt

V1.0：wrote a readme file

Git is a version control system.

Git is free software.

V2.0：add distributed

Git is a distributed version control system.

Git is free software.

V3.0：append GPL

Git is a distributed version control system.

Git is free software distributed under the GPL.

Using

$ git log

commit 3628164fb26d48395383f8f31179f24e0882e1e0

Author: Michael Liao <askxuefeng@gmail.com>

Date: Tue Aug 20 15:11:49 2013 +0800

append GPL

commit ea34578d5496d7dd233c827ed32a8cd576c5ee85

Author: Michael Liao <askxuefeng@gmail.com>

Date: Tue Aug 20 14:53:12 2013 +0800

add distributed

commit cb926e7ea50ad11b8f9e909c05226233bf755030

Author: Michael Liao <askxuefeng@gmail.com>

Date: Mon Aug 19 17:51:55 2013 +0800

wrote a readme file

$ git log --pretty=oneline

3628164fb26d48395383f8f31179f24e0882e1e0 append GPL

ea34578d5496d7dd233c827ed32a8cd576c5ee85 add distributed

cb926e7ea50ad11b8f9e909c05226233bf755030 wrote a readmefile

$ git reset –-hard HEAD^

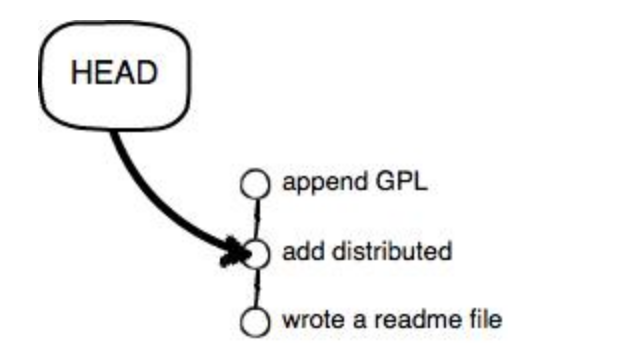
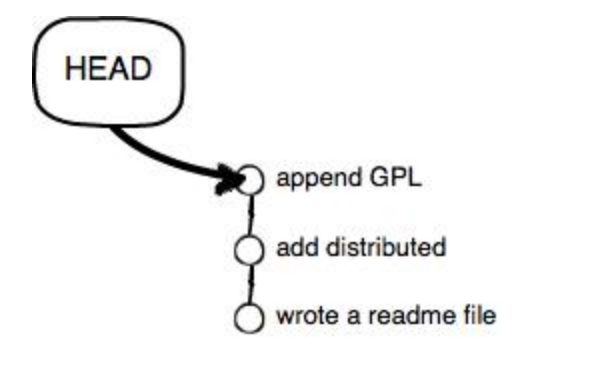
HEAD^ means last version

HEAD^^ means last second version

HEAD~100 means last 100th version

OR $ git reset --hard 3628164

HEAD is now at 3628164 append GPL



If forget the id of version

$ git reflog

ea34578 HEAD@{0}: reset: moving to HEAD^

3628164 HEAD@{1}: commit: append GPL

ea34578 HEAD@{2}: commit: add distributed

cb926e7 HEAD@{3}: commit (initial): wrote a readme file

Recap:

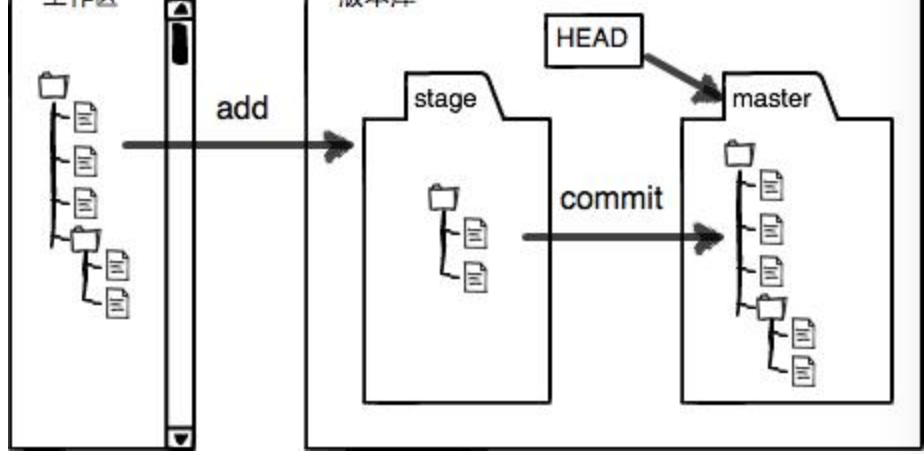
* Head point the current version
* Before reset the version, using git log to check the history of from latest to initialisation to ensure which version you will return back
* If want to go to future,

git relog to check the history of commands we used and is used for check the ID of commit, if we forgot

* **Working Directory and Repository (index/stage(represents temporary area))**

Local disc is the working directory.

|Working directory| |-------------Repository-------------------|



After $ git commit –m “”

Working directory and stage is clean

* **Manage Modification**

Git tracks the changes not files

Do some changes in readme.txt

$ git add readme.txt (---------stage)

Do some changes again in readme.txt (-------working directory)

$ git commit –m “git tracks changes”

$ 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: readme.txt

#

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

Recap:

1st modification 🡺 git add 🡺 2nd modification 🡺 git commit

So, only the changes in stage has been committed not in working directory

$git diff HEAD – readme.txt # view the changes between working directory and repository

diff --git a/readme.txt b/readme.txt

index 76d770f..a9c5755 100644

--- a/readme.txt

+++ b/readme.txt

@@ -1,4 +1,4 @@

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

-Git tracks changes.

+Git tracks changes of files.

Tips ： Git add every time while make changes

* **Undo and Modification**

$ cat readme.txt

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

My stupid boss still prefers SVN.

Delete “stupid” by hand

$ 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: readme.txt

#

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

discard the changes in working directory

$ git checkout -–readme.txt

\*\*\*\*\*\*\*\* $git checkout without -- , use for switching another branch

TWO CASES of changes

1st readme.txt was changed and was not added into stage, undo and modify back to

# Changes not staged for commit:

$ git checkout -–readme.txt

2nd readme.txt was changed and was also added into stage

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: readme.txt

$git reset HEAD readme.txt back to working directory

$git checkout –-readme.txt

$git status

On branch master

nothing to commit (working directory clean)

* **Delete file**

$git add test.txt

$git commit –m “add test.txt”

$rm test.txt # delete in working directory

TWO CASES：

1st delete from repository

$git rm test.txt

rm ‘test.txt’

$git commit –m “remove test.txt”

1 file changed, 1 deletion(-)

delete mode 100644 test.txt

2nd delete mistakenly

git checkout -– test.txt

* **Remote Repository**

CASE1: You want to link local repository to a new remote repository

$ssh-keygen –t rsa –C [youremail@example.com](mailto:youremail@example.com)

enter until the end

log on your account in GitHub website

click “Account settings” ---- “SSH key” ------“Add SSH Key”

Title is filled whatever. Key is filled the content of id\_rsa.pub

Use command line to copy the ssh key.



In the gitHub, create a new one. Once the new repository is created,



copy the command

$ git remote add origin [git@github.com:yangminp/learngit.git](mailto:git@github.com:yangminp/learngit.git)

origin represents the remote repository

$ git push -u origin master

#push all the local things to remote repository

#Actually, $git push origin master,

could upload the latest change to GitHub.

#SSH warning

There will be a warning since of the first time of using Git clone

*The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.*

*RSA key fingerprint is xx.xx.xx.xx.xx.*

*Are you sure you want to continue connecting (yes/no)?*

Confirm the if the figure print is really from GitHub, which can be check by [GitHub](https://help.github.com/articles/github-s-ssh-key-fingerprints/)

CASE2: You want to create a new remote repository without linking the local disc

$ git clone git@github.com:michaelliao/gitskills.git

Cloning into 'gitskills'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (3/3), done.

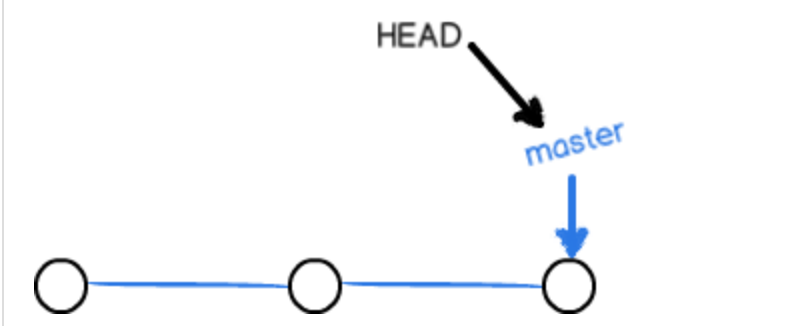
$ cd gitskills

$ ls

README.md

* **Manage branch**

Create and combine branch



master should be used to upload to remote repository

Head should be used to current branch

#Create dev branch and switch to dev branch

$ git checkout -b dev

Switched to a new branch 'dev'

==== $ git branch dev

$ git checkout dev

Switched to branch 'dev'

$git branch #to view current branches

#do some changes in readme.txt on dev branch

$git add readme.txt

$git commit –m “branch test”

[dev fec145a] branch test

1 file changed, 1 insertion(+)

#Switch back to master branch

$ git checkout master

Switched to branch 'master'

# Now merge the dev branch back to master

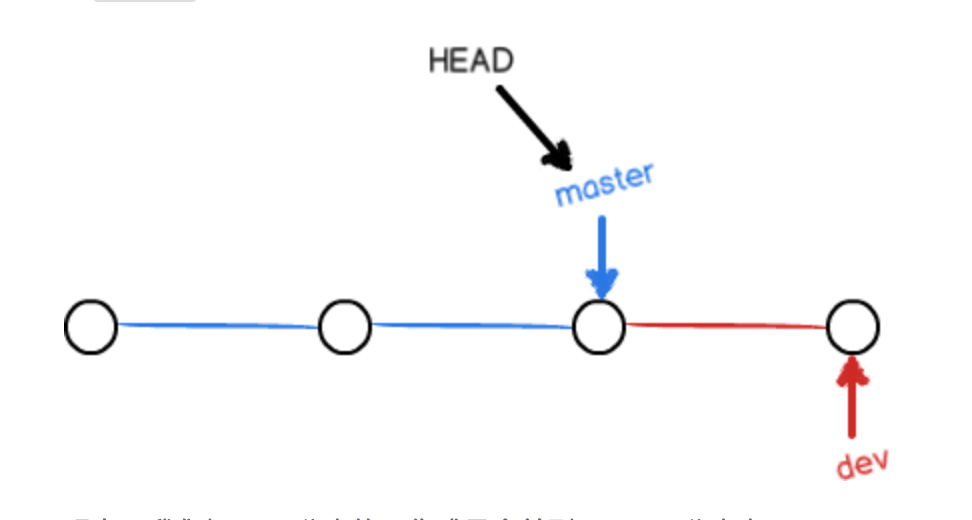
$git merge dev

Updating d17efd8..fec145a

Fast-forward

readme.txt | 1 +

1 file changed, 1 insertion(+)

 Fast forward: link master branch to dev directly

After branching, then delete dev branch

$git branch –d dev

Deleted branch dev (was fec145a)

$git branch

\*master

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

* **Deal conflicts**

$ git checkout -b feature1

Switched to a new branch 'feature1'

#do some changes in readme.txt

commit on the branch of feature

$ git add readme.txt

$ git commit -m "AND simple"

[feature1 75a857c] AND simple

1 file changed, 1 insertion(+), 1 deletion(-)

switch to master branch

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 1 commit.

# do some changes in readme.txt again

$ git add readme.txt

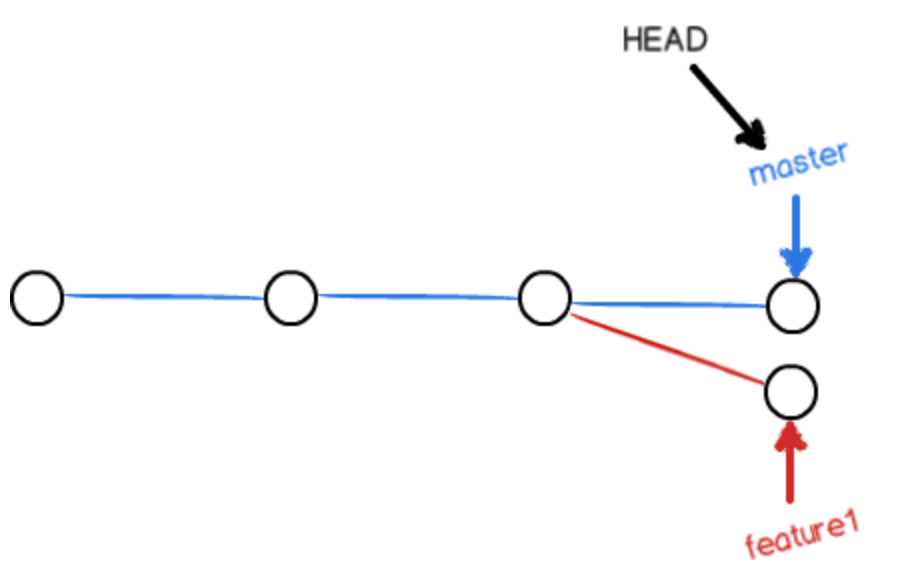
$ git commit -m "& simple"

[master 400b400] & simple

1 file changed, 1 insertion(+), 1 deletion(-)

Now both of master and feature has new different commit

In this case, Git cannot fast-forward merge.

****

$ git merge feature1

Auto-merging readme.txt

CONFLICT (content): Merge conflict in readme.txt

Automatic merge failed; fix conflicts and then commit the result.

$git status #tell us where the conflict exists

# On branch master

# Your branch is ahead of 'origin/master' by 2 commits.

#

# Unmerged paths:

# (use "git add/rm <file>..." as appropriate to mark resolution)

#

# both modified: readme.txt

#

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

Meantime, we can have a look “the file” which exist conflicts

*Git is a distributed version control system.*

*Git is free software distributed under the GPL.*

*Git has a mutable index called stage.*

*Git tracks changes of files.*

*<<<<<<< HEAD*

*Creating a new branch is quick & simple. (master)*

*=======*

*Creating a new branch is quick AND simple. (feature1)*

*>>>>>>> feature1*

modify the conflict file directly

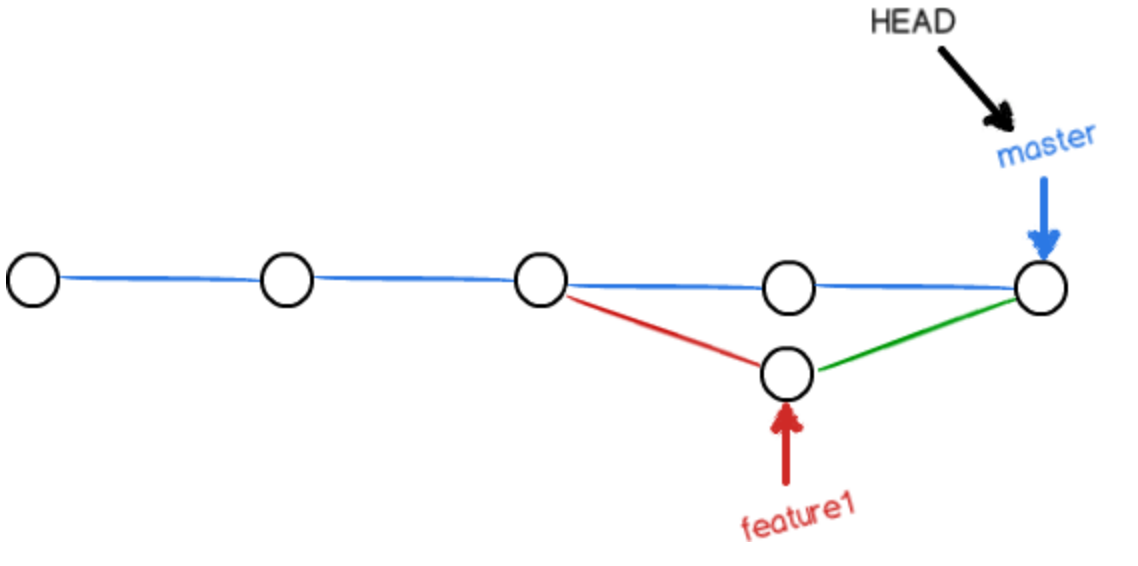
“Creating a new branch is quick and simple.”

$ git add readme.txt

$ git commit -m "conflict fixed"

[master 59bc1cb] conflict fixed

Now the branches became like this,



$ git log --graph --pretty=oneline --abbrev-commit

\* 59bc1cb conflict fixed

|\

| \* 75a857c AND simple

\* | 400b400 & simple

|/

\* fec145a branch test

...

Finally, delete the “feature1” branch

Deleted branch feature1 (was 75a857c).

Summary:

If merge branch failed, then deal conflicts before add-commit

* **Strategies of managing branches**

The disadvantage of fast-forward merge is that the info of sub-branch will be lost.

If forbid forcibly Fast-forward merge, Git will create a new commit while merging.

Then, the branch info can be viewed from branch history

--no-ff git merge:

$git checkout –b dev

#do some changes in readme.txt, and add-commit

$ git add readme.txt

$ git commit -m "add merge"

[dev 6224937] add merge

1 file changed, 1 insertion(+)

$ git checkout master

Switched to branch 'master'

Notice: --no—ff means forbid Fast-forward

$ git merge --no-ff -m "merge with no-ff" dev

Merge made by the 'recursive' strategy.

readme.txt | 1 +

1 file changed, 1 insertion(+)

add –m , since merging of this time we need to create a new commit

then,

$ git log --graph --pretty=oneline --abbrev-commit

\* 7825a50 merge with no-ff

|\

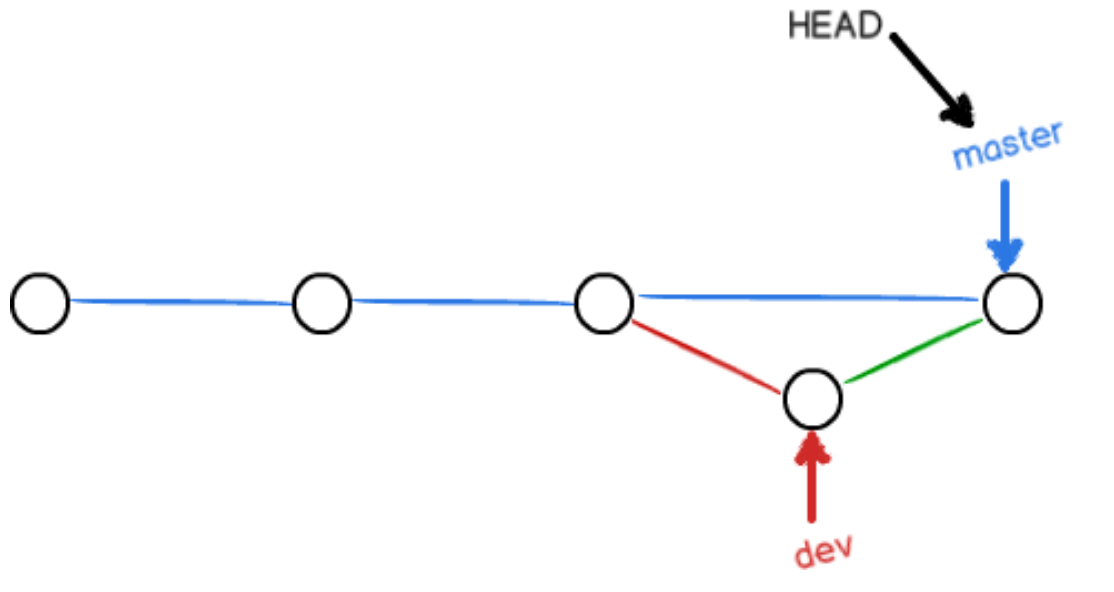
| \* 6224937 add merge

|/

\* 59bc1cb conflict fixed

...

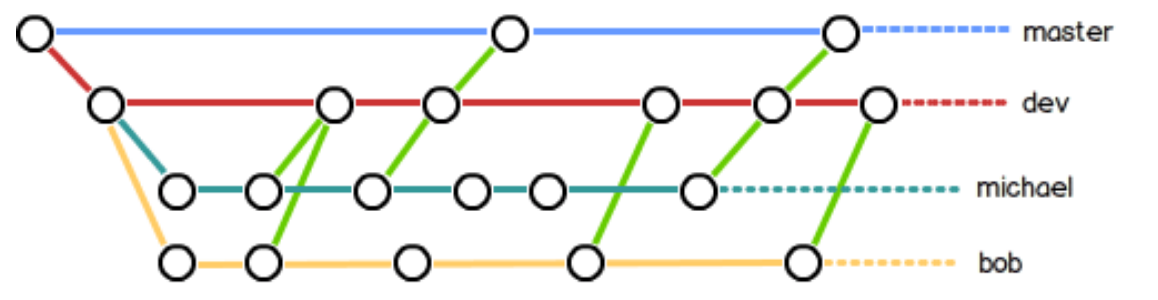
as we can see from above, the merge will become like this:



In real world, we should follow these rules to use Git:

Firstly, master should be real steady and only is used to push new versions instead of being worked on.

Generally, for example, the v1.0 is released, then push from dev to master branch. Things like this:



* **Branches of bugs**

Every bug can be fixed by creating a new branch. After fixing the bug, merging the branches, then delete the temporary branch.

For example, you have a task of fixing issue-101, naturally, you can just create a branch called “issue-101” to fix it, but, hold on, the work on the dev branch have not been submitted yet.

$ git status

# On branch dev

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: hello.py

#

# 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: readme.txt

#

Currently, the work was just done partly which can be add-commit now, probably, it will take one more day to finish. However, you have to fix the bug within 2 hours

Fortunately, stash is provided by Git, which is used to “save and hide” for current the process of work. The status of work could be recovered later.

$ git stash

Saved working directory and index state WIP on dev: 6224937 add merge

HEAD is now at 6224937 add merge

Now, $git status can be used to check the working directory which is empty (Except the files which is not administrated by Git), then we can feel free to create an issue-01 branch to fix bugs.

Firstly, we should make sure on which branch we will fix the bug. Assuming, we need fix the bug on the branch of master, then create a temporary branch on master branch

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 6 commits.

$ git checkout -b issue-101

Switched to a new branch 'issue-101'

#Now, fix the bug, change “Git is free software” into “Git is a free software”, then add-commit

$ git add readme.txt

$ git commit -m "fix bug 101"

[issue-101 cc17032] fix bug 101

1 file changed, 1 insertion(+), 1 deletion(-)

Finish debugging, then switch to master branch, finally, delete “issue-101” branch:

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 2 commits.

$ git merge --no-ff -m "merged bug fix 101" issue-101

Merge made by the 'recursive' strategy.

readme.txt | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

$ git branch -d issue-101

Deleted branch issue-101 (was cc17032).

Congratulations, it only take 5 minutes to fixing the bug instead on 2 hours. Now, it is time to continue working back dev branch

$ git checkout dev

Switched to branch 'dev'

$ git status

# On branch dev

nothing to commit (working directory clean)

$ git stash list

stash@{0}: WIP on dev: 6224937 add merge

As we can see, the things still exist. There are two ways to recover data.

1st $git stash apply but after date is recovered, the content of stash still exists. $git stash drop to delete.

2nd $git stash pop recovers and deletes data of stash

$ git stash pop

# On branch dev

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: hello.py

#

# 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: readme.txt

#

Dropped refs/stash@{0} (f624f8e5f082f2df2bed8a4e09c12fd2943bdd40)

$git stash list (be used to view the content of stash)

u can use stash many times, $git stash list to check before u want to recover the particular one.

$ git stash apply stash@{0}

**Recap:**

Create a temporary branch to fix bug, then merge with master. Finally, delete the temporary branch

If current work has not been done, then $git stash current working directory----🡪 fix bug -----🡪 git stash pop

* **Feature branches**

There is always lots of new features needed to be added during development.

Every time, we don’t want testing code messes master branch up. So, create a new feature branch on which develop new feature branch🡺 merge 🡺 delete feature branch

Now, we have a new task that develop a new feature which is called “Vulcan”, and it is used for create the new version of spacecraft.

Let’s start，

$ git checkout -b feature-vulcan

Switched to a new branch 'feature-vulcan'

------------------Development done after 5 minute------------------

$ git add vulcan.py

$ git status

# On branch feature-vulcan

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: vulcan.py

#

$ git commit -m "add feature vulcan"

[feature-vulcan 756d4af] add feature vulcan

1 file changed, 2 insertions(+)

create mode 100644 vulcan.py

####switch to dev, then start merge

$ git checkout dev

Similar to bug branch, 🡪merge 🡪 delet

But, the new feature has to be deleted since the lack of money

$ git branch -d feature-vulcan

error: The branch 'feature-vulcan' is not fully merged.

If you are sure you want to delete it, run 'git branch -D feature-vulcan'.

$ git branch -D feature-vulcan

Deleted branch feature-vulcan (was 756d4af).

Delete forcibly is used to delete the branch before merging

* **Cooperate with others**

Actually, Git can link local master branch and remote master branch, however, the remote repository’s name is origin

To view the info of remote repository,

$git remote

origin

or

git remote –v

origin git@github.com:yangminp/learngit.git (fetch)

origin git@github.com:yangminp/learngit.git (push)

#if there is not permit of punishing, then the address of pushing will not be shown

**Pushing-branch**

Pushing-branch should push local branches to remote branches by assigning the local branch

$ git push origin master

if we want to push other branch, like “dev” brach

$ git push origin dev

However, it is not necessary to push all branches

Master branch must be pushed

Dev also

Bug not

Feature depends on u

**Fetch-branch**

Team-work, everyone will push things on master and dev.