Advanced Git

Irina Gulina

October 23, 2019

Agenda

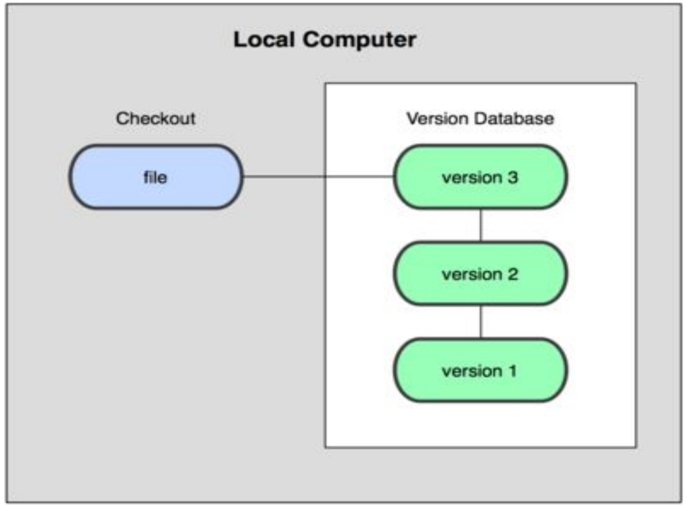
- [recap] What is Git?
- [recap] Git Basics
- Branching
- Collaborating
- Handy Git tools and commands
- Local and public troubles
- Git Etiquette

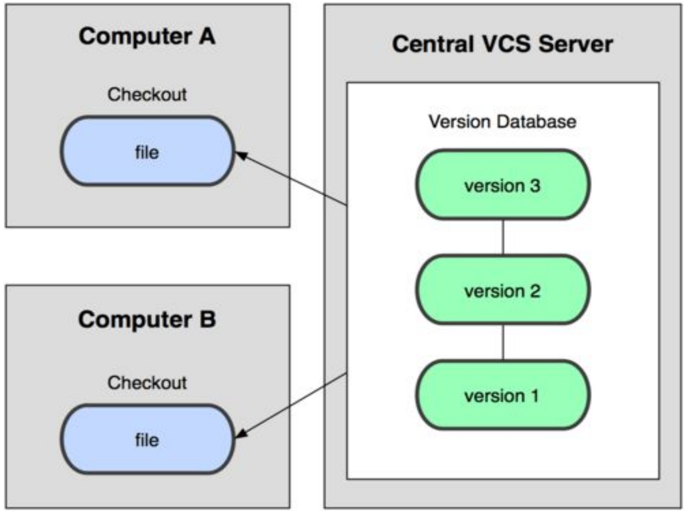
http://bit.ly/devconf19-gtw

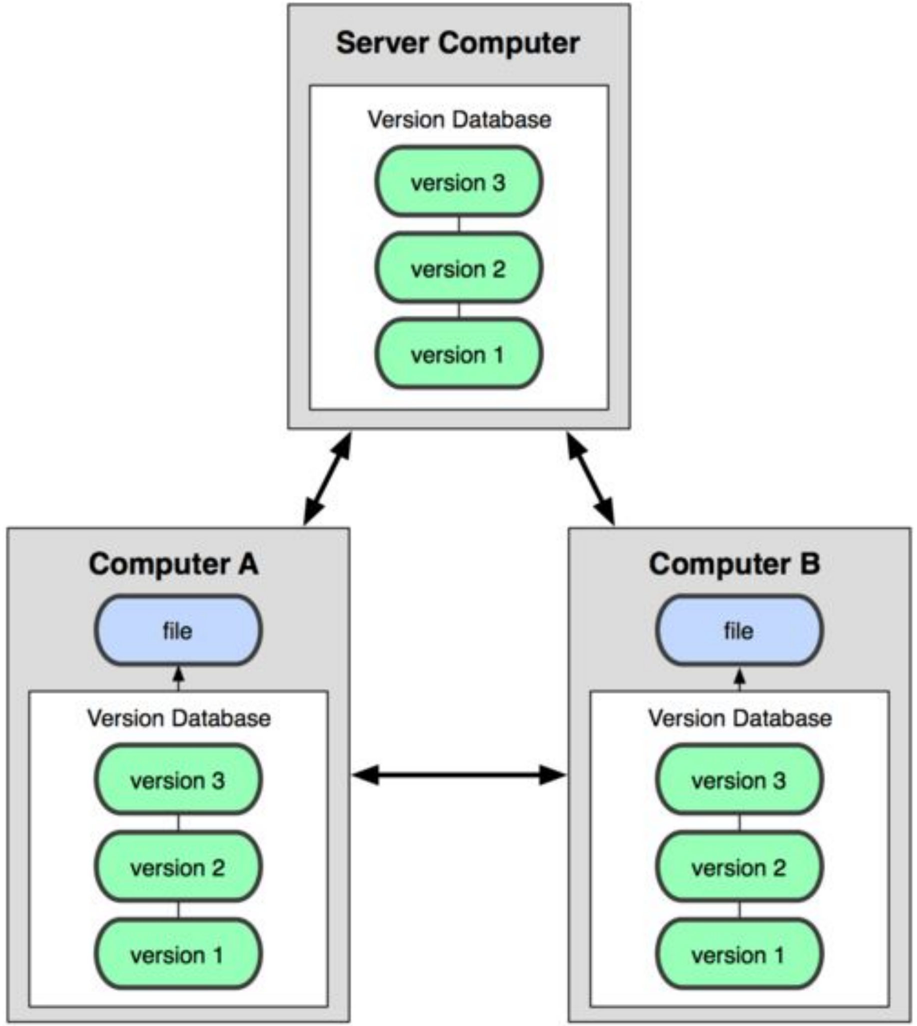
What is Git?

Distributed version control system for managing source code, i.e. it’s a system to

- record and save each file change
- restore a previous version of your code at any time


Local Computer
Checkout
file
Version Database
version 3
version 2
version 1


Computer A
Checkout
file
Central VCS Server
Version Database
version 3
version 2
version 1
Computer B
Checkout
file


Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1

What problem does it solve?

- Keep track of code history
- Collaborate on code as a team
- See who made which changes

Basic Git workflow

- Modifying files in the working tree
- Staging changes in index
- Committing files to a repository

What Git commands do you know?

Do you know how to ...

- Create a new repository locally?
- Clone an existing remote repository?
- Check status of your changes?
- Record changes locally?
- Commit changes to a remote repository?
- Find info about Git commands?

Git Basic Commands

- `help`

- `init`
- `clone`

- `config`

- `add`
- `status`
- `diff`
- `commit`
- `reset`
- `mv`
- `rm`

- `branch`
- `checkout`
- `merge`
- `log`
- `stash`

- `fetch`
- `pull`
- `push`
- `remote`

Git help

Documentation www.git-scm.com/docs

```
$ git help
$ git help <command>
```

Git Branching

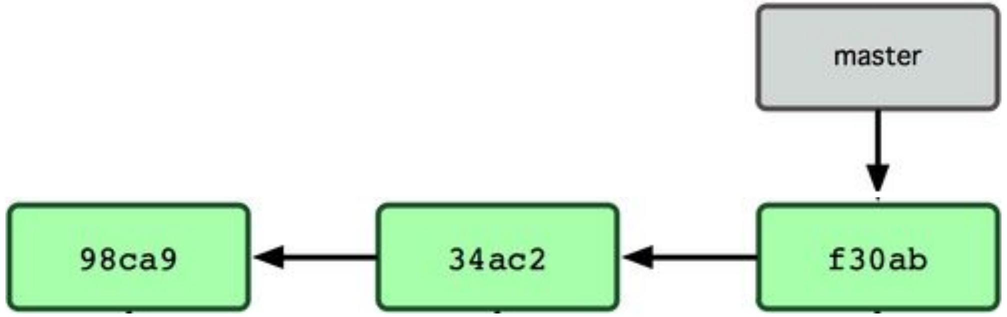
Branching

- Default branch
- Create a new branch
- Switch branches
- Work in parallel on different branches
- Merge branches
- Delete a branch
- Rename a branch
- *Stash changes

- Resolve merge conflicts
- Rebase a branch

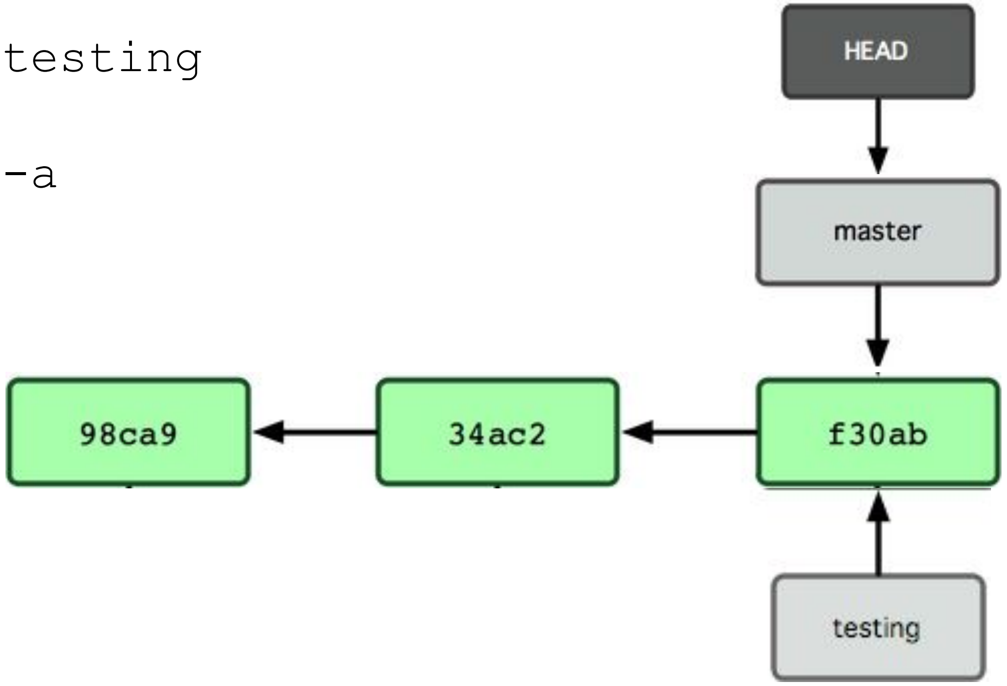
Branching

```
$ git branch
$ git branch -v
```



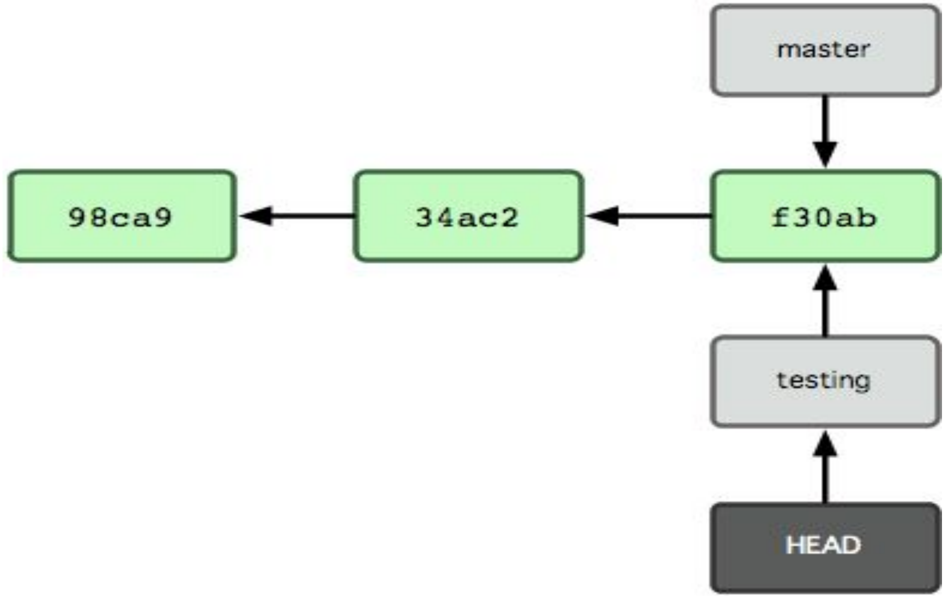
Create a branch

```
$ git branch testing
$ git branch
$ git branch -a
```



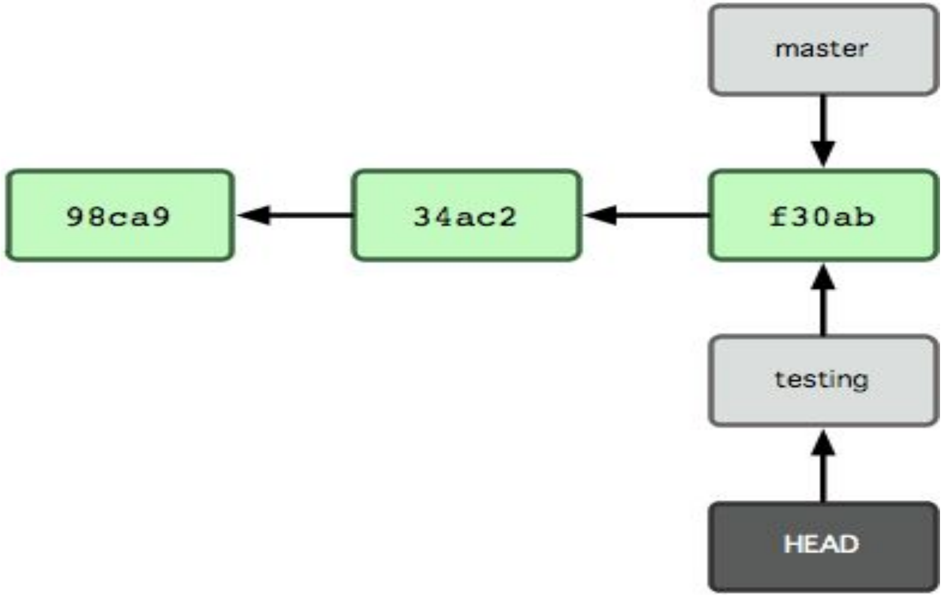
Switch a branch

```
$ git checkout testing
$ git branch
```



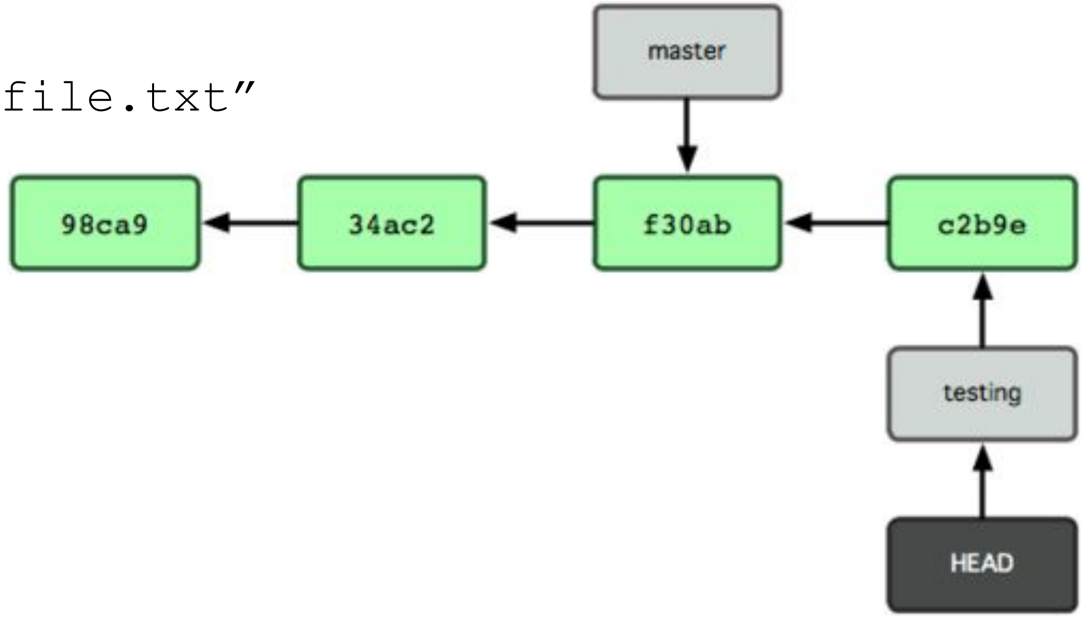
Create and switch

```
$ git checkout -b testing
```



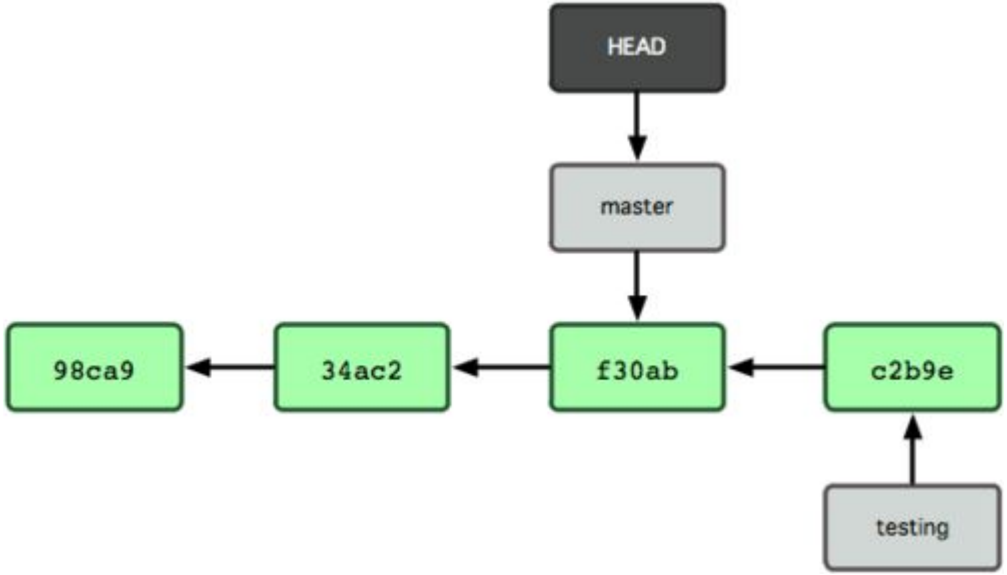
Work in parallel

```
$ touch file.txt
$ git commit -a -m “add file.txt”
```



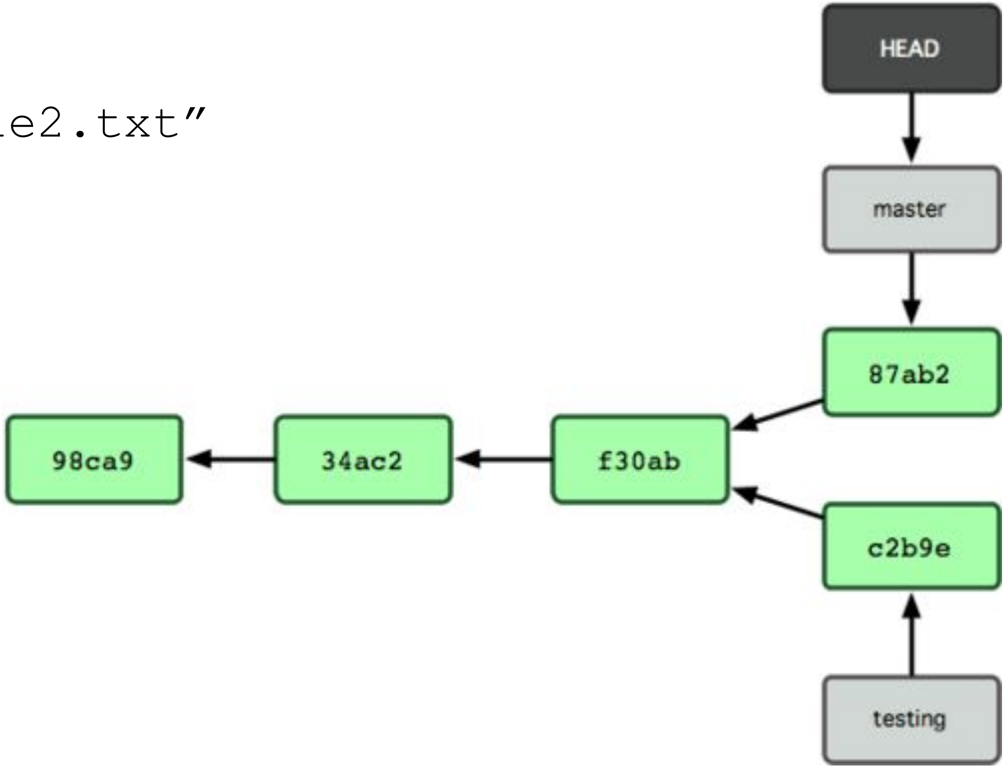
Work in parallel

```
$ git checkout master
```



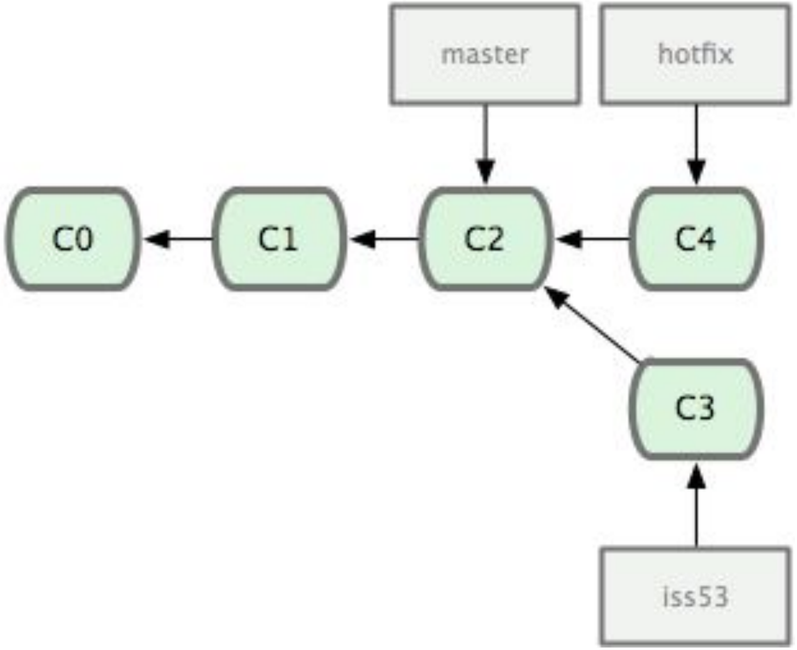
Work in parallel

```
$ touch file2.txt
$ git commit -a -m "add file2.txt"
```



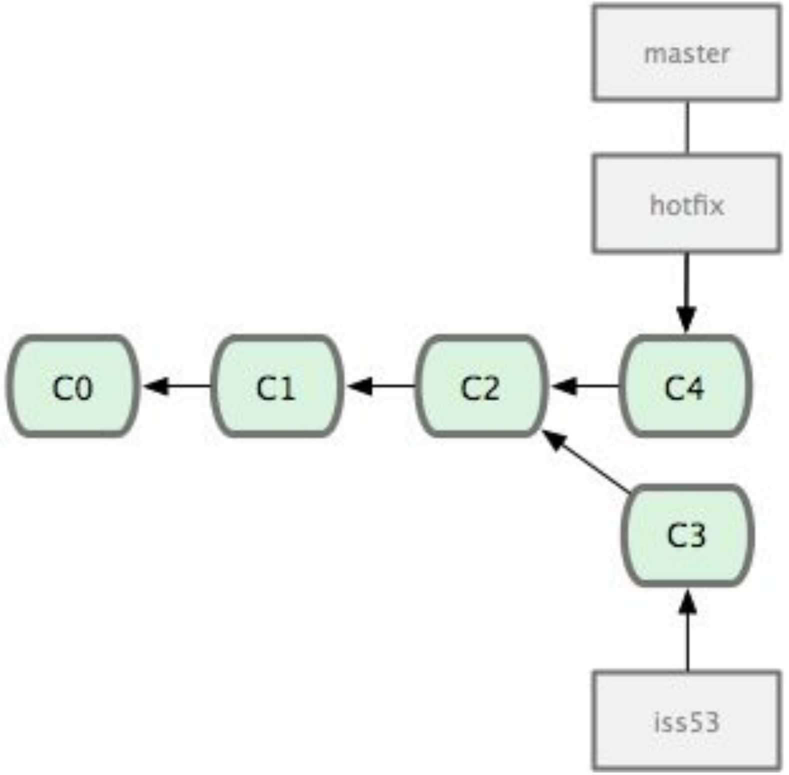
Merge branches

```
$ git checkout master
$ git merge hotfix
```



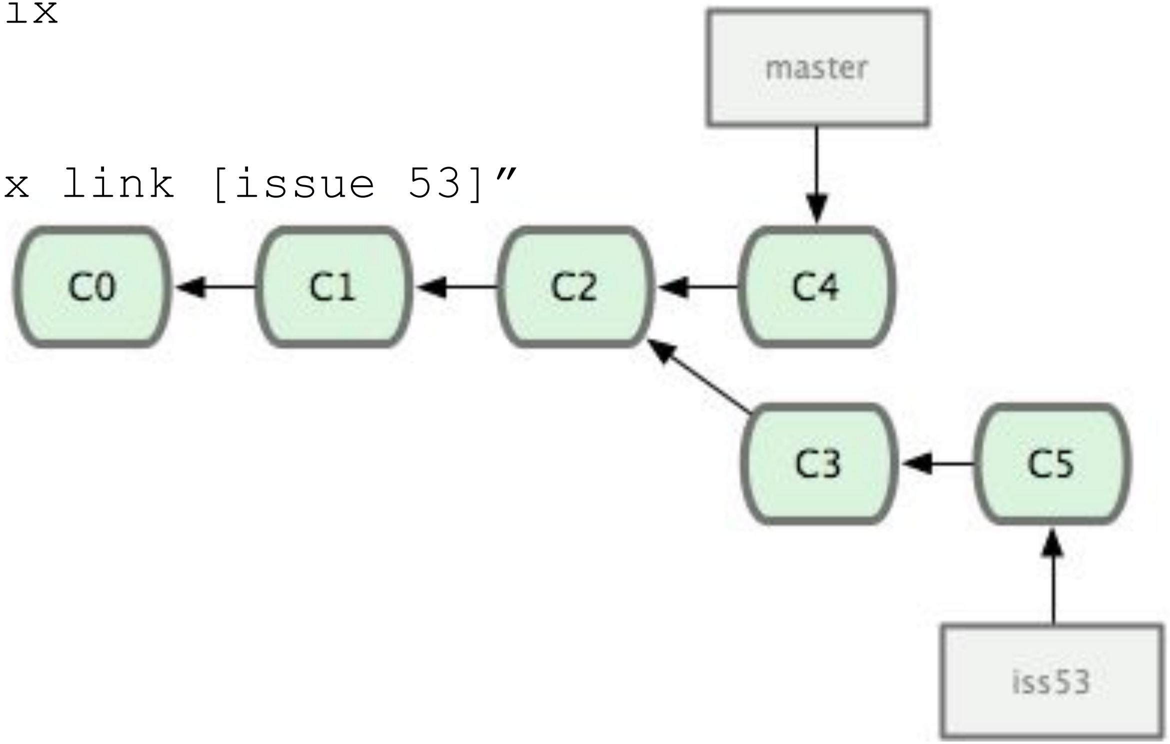
Merge branches

```
$ git checkout master
$ git merge hotfix
```



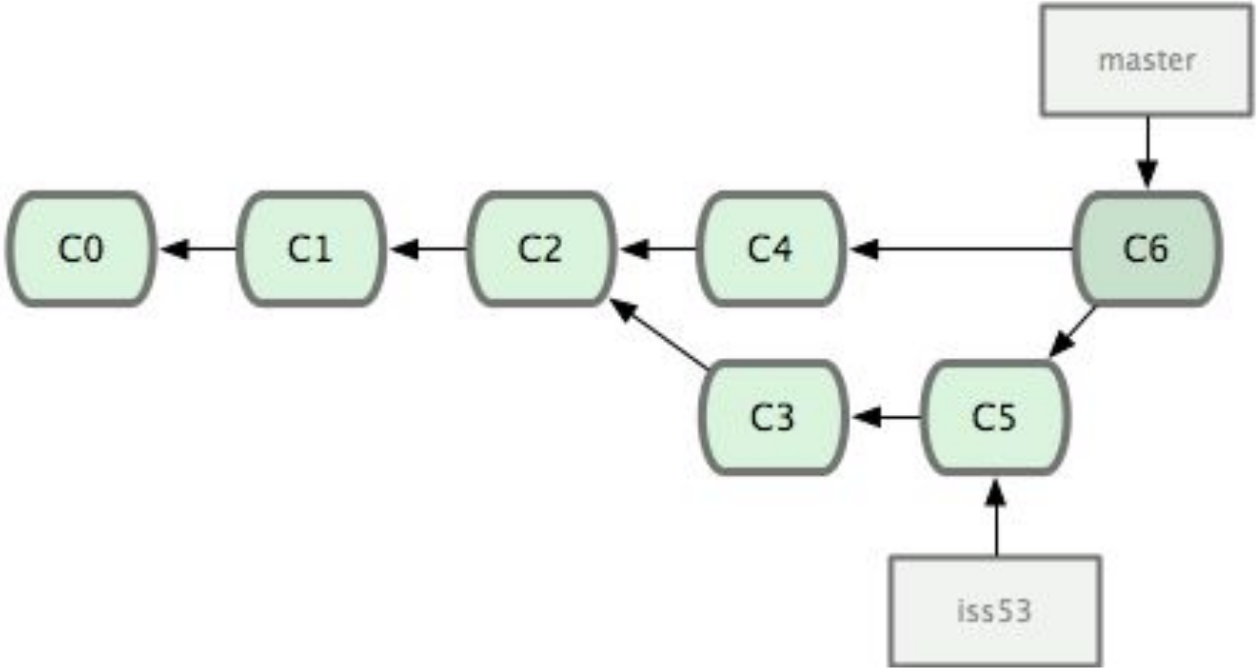
Merge branches

```
$ git branch -d hostfix
$ git checkout iss53
$ vi index.html
$ git commit a -m “fix link [issue 53]”
```



Merge branches

```
$ git checkout master
$ git merge iss53
```



Merge strategies

```
$ git merge -s recursive branch1 branch2
$ git merge -s resolve branch1 branch2
$ git merge -s octopus branch1 branch2 branch3 branchN
$ git merge -s ours branch1 branch2 branchN
$ git merge -s subtree branchA branchB
```

Merge conflicts

- Git fails to start the merge
  ```
  error: Entry '' not uptodate. Cannot merge. (Changes in
  working directory)
  ```

- Git fails during the merge
  ```
  error: Entry '' would be overwritten by merge. Cannot
  merge. (Changes in staging area)
  ```

Create a merge conflict

- Create a Git repo
- Add some text into a file
- Commit the change

- Create a new branch
- Overwrite text in that file and commit it

- Updata the same file again on master, commi it

- Try to merge those two branches

Resolve a merge conflict

- Identify the conflict
- Inspect it
- Make changes
- Stage those changes

```
$ git status
$ git log --merge
$ git diff

$ git mergetool
```

```
$ git checkout
$ git reset --mixed
```

```
$ git merge --abort
$ git reset
```

Delete a branch

- You can’t remove a branch you checked out at
- You can remove a merged branch
- You can remove a branch with unstaged changes
- Sometimes you need to apply force

```
$ git branch -d branch_name
$ git branch -D branch_name
```

Stash changes

- Stashing your changes
- Re-applying your stashed changes
- Stashing untracked and ignored files
- Multi stashing
- Viewing stash diff
- Create a branch from stash
- Cleaning up your stash

```
$ git stash
$ git stash pop
$ git stash apply
$ git stash -u
$ git stash -a
$ git stash list
$ git stash pop stash@{2}
$ git stash show
$ git stash show -p
```

$ git stash drop stash@{1}
$ git stash clear

Collaborating

Collaborating

- Add remote repositories

```
$ git remote add origin <url>
```

- Download remote content

```
$ git fetch origin
$ git fetch --all
$ git fetch --dry-run
$ git fetch branch_name
$ git merge origin/master

$ git pull
$ git pull --verbose
```

- Upload local content to a remote repository

```
$ git push
$ git push --all
$ git push --force
```

How to find things

Revision selectors

1. `1c002dd4b536e7479fe34593e72e6c6c1819e53b`

2.
```
$ git log --oneline

1c002dd changed the version number
085bb3b removed unnecessary test code
a11bef0 first commit
```

Revision selectors

3. `$ git reflog`

```
734713b HEAD@{0}: commit: fixed refs handling, added gc
d921970 HEAD@{1}: merge phedders/rdocs: Merge made by the
'recursive' strategy.
1c002dd HEAD@{2}: commit: added some blame stuff
1c36188 HEAD@{3}: rebase -i (squash): updating HEAD
95df984 HEAD@{4}: commit: # This is a combination of two
```

Revision selectors

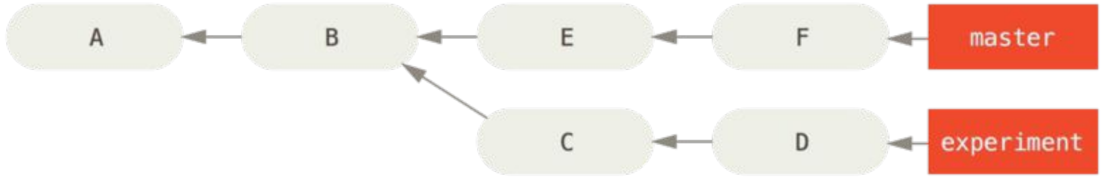
4.
```
$ git show master@{yesterday}
$ git show master@{2.months.ago}
```

Revision selectors

5. Ancestry references (`^ ~`)
   - `ca82a6d^`
   - `ca82a6d^^`
   - `HEAD`
   - `HEAD^`
   - `HEAD^2` (is it the same as `ca82a6d^^` ?)
   - `HEAD~` (is it the same as HEAD^ ?)
   - `HEAD~2` (is it the same as HEAD^2 ?)
   - `HEAD~3^2` (is it valid?)

Revision selectors

6. Ranges of commits



How to show commits on experiment branch, which are not on master?
The opposite?
How to show local commits which are not on origin remote?

Solution:

```
$ git log master..experiment
$ git log experiment..master
```

Revision selectors

7. Multiple points

   How to see what commits are in any of several branches, that aren't in the branch you're currently on?

   How to see all commits on A and B, which are not on C?

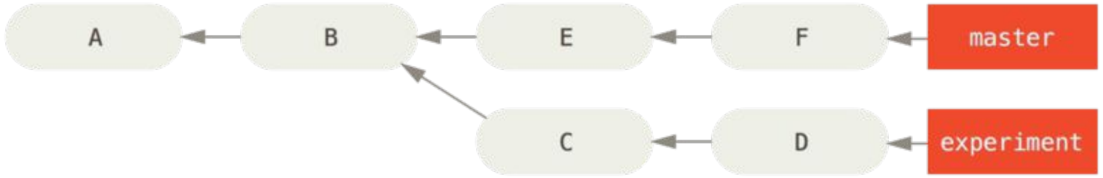
   More than two references can be specified.

Solution:

```
$ git log A B ^C
$ git log A B --not C
```

Revision selectors

8. Scenario:



How to see what commits are in either of two branches, but not on both of them? i.e. E, F, C, D?

Solution:

```
$ git log master...experiment
```

Git Log Searching

Problem:

How to find specific commits by the content of their messages or even the content of the diff they introduce?

Solution:

```
$ git log -S calc --oneline
  61e3ce7 add a new function

$ git log -S x1 --oneline
  18f3671 change params in search
  a5c51ae edit search func
  7a7c4f7 add search func
```

Git Log Searching

Problem:

How to show the history (all commits) of a function or line of code in a codebase?

Solution:

```
$ git log -L :add:<file>
```

Local troubles

Git cardinal rule

You have a great freedom

to rewrite your history ***locally***

Undoing local changes, not committed

Steps to reproduce:

The cat walked across your keyboard, while you were making coffee. You have not noticed and saved the changes, then saw them with `git diff`.

Solution:

```
$ git checkout -- <file>
```

Changing the last local commit

1. How to modify the last commit message

Solution:

```
$ git commit --amend
```

Changing the last local commit

2. How to modify the content of the last commit

Solution:

Make changes
Stage those changes

```
$ git commit --amend
```

or

```
$ git commit --amend --no-edit
```

Don’t amend your last commit if you have already pushed it!

Undo the last local commit(s)

Solution:

```
    $ git reset <last good commit>
```

or

```
    $ git reset --hard <last good commit>
```

Find and restore a deleted file

Scenario:

A file was deleted and this change was committed. More commits were added. How to find a commit deleting that file and restoring it?

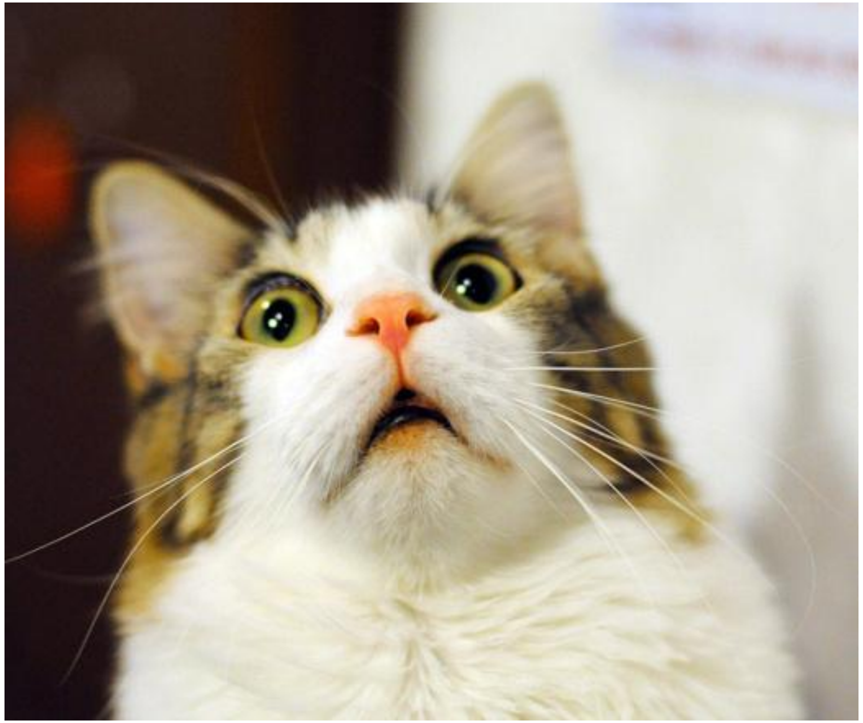
Solution:

1)
```
$ git rev-list -n 1 HEAD -- path/to/file
$ git checkout <commit>^ -- path/to/file
```

Delete and restore all files

Scenario:

```
$ rm -r *
```



Redo after undo the last local commit(s)

Scenario:

You made some commits, then did a `git reset --hard` to “undo” them, and then you want those changes back. There are several possible solutions, it depends on what you want to accomplish.

Solution:

```
    $ git reflog

 -  $ git reset --hard <commit>
 -  $ git checkout <commit> -- <filename>
 -  $ git cherry-pick <commit>
```

Revert a single file to a specific commit

Scenario:

Some changes on a file were committed multiple times. Then, an author wants to restore that file to a specific commit

Solution:

```
$ git log
$ git diff <commit>
$ git checkout <commit> -- filename
```

or if one commit before a specific one:

```
$ git checkout <commit>~1 filename
```

Stop tracking a tracked file

Scenario:

A log file was accidently added (by commit) to the repository. Since then Git reports there are unstaged changes in that file even though there is `*.log` entry in `.gitignore`.

Solution (remove a file from a git repo, but not locally)

```
$ git rm --cached file.log
```

for a single directory

```
$ git rm --cached -r logs
```

Question: How to remove multiple files?

Multiple undo/redo of several local commits

Scenario:

There is a dozen or so commits, but only some of them are needed to be pushed, others changed or deleted

Solution:

`$ git rebase -i HEAD~5` don’t include any commit you’ve already pushed. Notice the order of commits.

- Reordering commits
- Squashing
- Splitting

Fix an earlier local commit

Scenario:

A file was not included in an earlier commit.

Solution:

```
$ git add <file>
$ git commit --fixup <earlier-commit>
$ git rebase -i --autosquash <even-more-earlier-commit>
```

07

Removing a file from every commit

Scenario:

Remove a file (e.g. with a sensitive info) from the entire history.

Solution:

```
$ git filter-branch --tree-filter ‘rm -f id_rsa’ HEAD

Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)
Ref 'refs/heads/master' was rewritten
```

Moving local commits between branches

Scenario:

Commits were made on a `master` branch, but they should be on another branch instead

Solution:

```
$ git branch feature
```

What is the difference with `git checkout -b feature`?

```
$ git reset --hard origin/master
$ git checkout feature
```

How to avoid it?

Outdated branch

Scenario:

You commited changes to one `feature` branch based on master which was pretty far behind remote `master`. You wish your `feature` branch be up-to-date with the remote `master` and your commits be on top of that.

Solution:

```
$ git checkout master
$ git pull
$ git checkout feature
$ git rebase master
```

Restore a deleted branch

Scenario:

You deleted a branch in your Git repository, but want it back.

Solution:

Find a SHA of that branch from terminal history or `git reflog`

```
$ git checkout -b <branch> <SHA>
```

Save changes without committing

Scenario:

You made some code changes, but it’s not a good time to commit. You need to switch branches to fix an urgent bug. How to save your work?

Solution:

```
$ git stash
```

Find the commit, that introduced a bug

Scenario:

You created several commits, but from some certain point the application gets broken. It’s unclear what it caused and which commit introduced the bug.

Solution:

```
$ git bisect
```

Public troubles

Undo a commit, pushed

Steps to reproduce:

```
$ touch file.txt
$ git add file.txt
$ git commit -m “Something terribly wrong”
$ git push origin master
```

Undo a commit, pushed

Solution:

Find SHA hash of that commit.

```
$ git revert <commit>
$ git push
```

It’s the safest scenario, it doesn’t alter history!

How to restore orphaned or deleted commits

Steps to reproduce:

```
$ git reset --hard HEAD~1

$ git push --force
```

How to restore orphaned or deleted commits

Solution:

- Find SHA hash of that commit.
- Create a new branch with that commit as the head of the branch
  ```
  $ git branch my-new-branch <commit>
  ```
- Ensure all changes are on that branch
- Merge changes to master

Edit the message of older or multiple commit(s), pushed

Solution:

1)
```
$ git commit --amend
$ git push --force
```

2)
```
$ git rebase -i HEAD~5
```
# Display last 5 commits

or `git rebase -i <commit>`

Replace `pick` with `reword` in opened editor

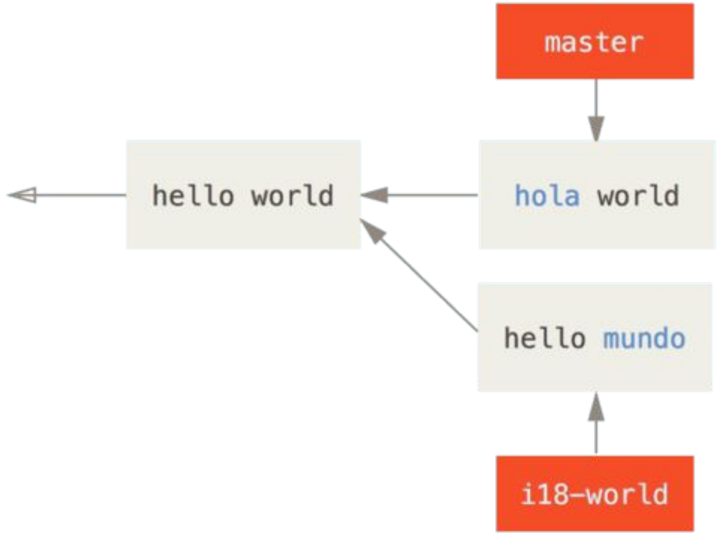
Edit the commit messages

Save and close the file

```
$ git push --force
```

Avoid repeated merge conflicts

Scenario:



Solution:

```
$ git rerere
```

Rename a branch

Scenario:

You made a spelling mistake in a branch name. Instead of `bugfix-15631` you named it `idontknow`. Maybe you were hungry that moment. Now you want to rename it.

Solution:

```
$ git branch -m <old-branch> <new-branch>

$ git push origin :<old-branch>
$ git push origin --set-upstream <new-branch>
```

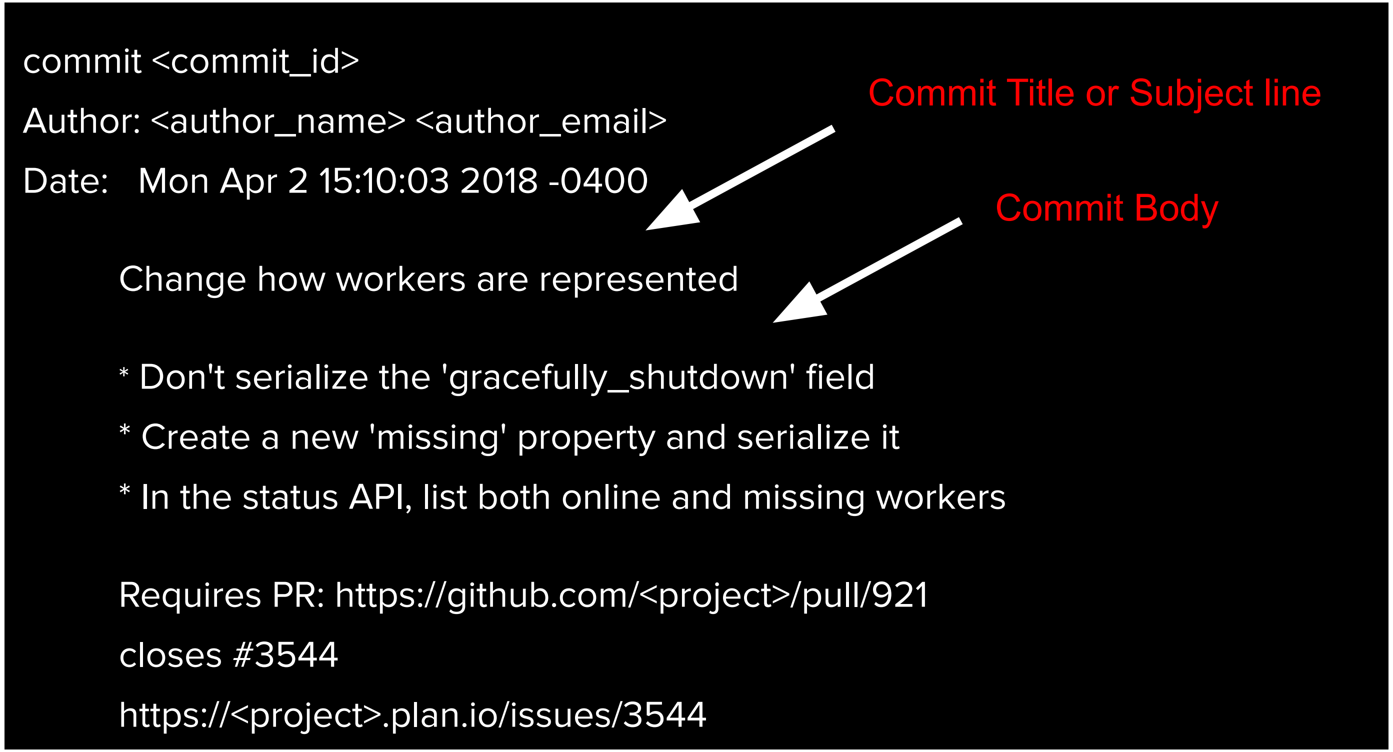
Git Etiquette

Poor quality code can be refactored.
A terrible commit message lasts **forever**.

What is a commit message

- Title/Subject line
- Body

Commit message example



```
commit <commit_id>
Author: <author_name> <author_email>
Date:   Mon Apr 2 15:10:03 2018 -0400

        Change how workers are represented

        * Don't serialize the 'gracefully_shutdown' field
        * Create a new 'missing' property and serialize it
        * In the status API, list both online and missing workers

        Requires PR: https://github.com/<project>/pull/921
        closes #3544
        https://<project>.plan.io/issues/3544
```

Usage of a commit title

- git log --pretty=oneline
- git rebase --interactive
- merge.summary
- git shortlog
- git format-patch, git send-email, …
- reflogs
- Gitk
- GitHub user interface

Commit history

```
$ git log --oneline
cf2***e some updates
7ae***f some structure changes
10e***d todo
1b4***1 improved
hj3***b docs
47a***m some updates
871***a little bit reworked and added specific part for docker
type
```

What constitutes a good commit title?

- git commit -m "Fix login bug"
- git commit or git commit --verbose

```
Redirect user to the requested page after login

https://link/to/issue/tracker
```

What constitutes a good commit title?

- Capital letter, 50/72, no punctuation in the end

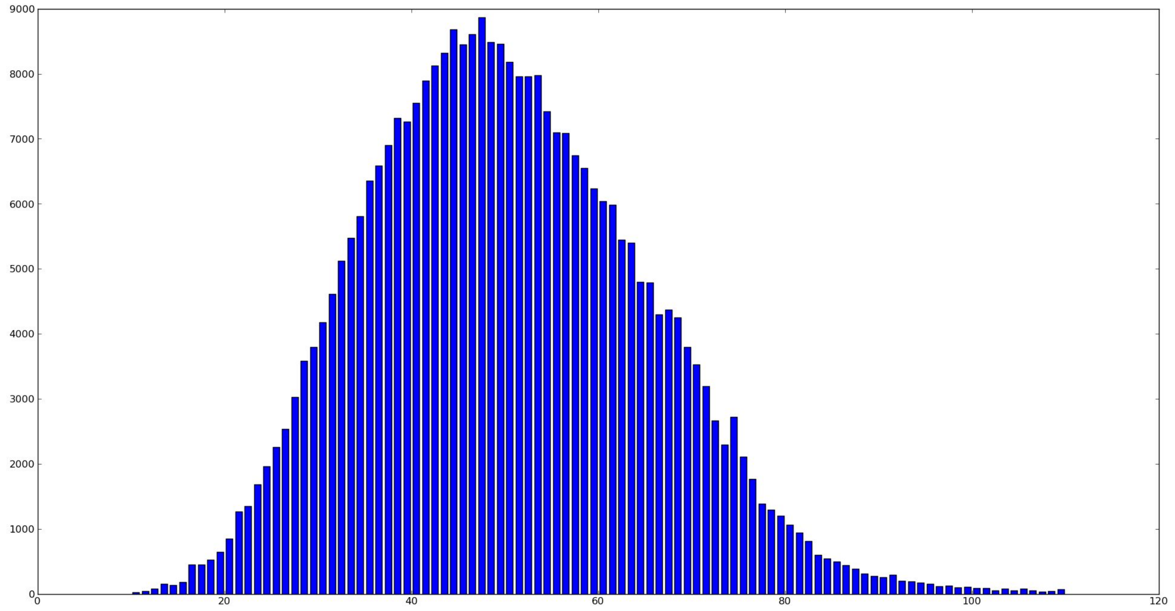
```
$ git commit
A brief summary of the commit

A paragraph describing what changed and its
impact."
```

What constitutes a good commit title?

- compare to the linux kernel contributors

```
$ git shortlog |\
  grep -e '^ ' |\
  sed 's/[[:space:]]\+\(.*\)$/\1/' |\
  awk '{lens[length($0)]++;} END {for (len in lens)
print len, lens[len] }' |\
  sort -n
```

What constitutes a good commit title?



What constitutes a good commit title?

- Present Tense and Imperative Mood

```
cf****e Adds unit tests
7a****f Fixed unit tests
10****d Update unit tests
1b****1 Removing unit test
```

“If accepted, this commit will <your commit message goes here>.”

What constitutes a good commit title?

- Reference to an issue

```
Redirect user to the requested page after login

https://link/to/issue/tracker
```

What constitutes a good commit title?

- Clear Title - What is commit about?
- Present Tense and Imperative Mood
- Clear Body - Why is it needed?
- 50/72
- Reference to an issue

Git push

IF YOU DO FORCE PUSH....

May the force stay with you.

Submitting a PR

Why do we use PR workflow

- Share changes
- Get review and feedback
- Encourage quality

What constitutes a good PR?

- Complete piece of work
- Adds value in some way
- Solid title and body
- Clear commit history
- Small

Contributors

Before submitting a PR

- Follow the repo’s conventions
- Double check your code (and ToDos)
- Add docs
- Keep changes small
- Separate branch
- Be clear and specific
- Check your ego and be polite

Contributors

After submitting a PR

- Check your ego and be polite
- Ensure your branch merge and tests pass
- Use --amend, --fixup or rebase -i
- Don’t merge your own PR

WIP PR?

- Don’t overuse WIP label
- Remove WIP label when ready
- “This is ready for review, please.”

Reviewing a PR

Reviewers

- Be kind and polite
- Check commit history
- Don’t fix issues
- Ensure the branch can be merged
- CI Tests pass
- **Don’t merge WIPs**
- Squash
- Delete branch

Thank you!

igulina@redhat.com