

How not to be a Git

Tips and tricks for a good workflow

Who am I?

- Adam Jimerson
- Software Engineer @ Code Journeymen
- GDG Gigcity Organizer
- vendion@gmail.com
- <https://google.com/+AdamJimerson>



What is a Git?

1. A distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.
2. A mild profanity with origins in British English for a silly, incompetent, stupid, annoying, senile, elderly or childish person.

Still lost?

- [Code School + Github's 'Try Git'](#) (interactive)
- [Bitbucket Git Tutorials](#)
- [Pro Git Book](#) or [Online version](#) (more recent)

Lets start with tips



Listing tracked files

List all tracked files

```
$ git ls-files
```

List all tracked files in the current branch

```
$ git ls-tree -r <branch> --name-only
```

Ignoring tracked files

First we need to remove the file from Git

```
$ git rm --cached <filename>
```

Then add the file to the ignore file

```
$ echo 'filename' >> \ $projectRoot/.  
gitignore
```

Ignoring tracked files

To tell git to ignore changes to a file, but not delete it, run:

```
$ git update-index --assume-unchanged  
\  
<filename>
```


Ignoring files

Use Global Gitignore files

```
$ git config --global core.excludesfile \ ~/.  
gitignore_global
```

Good starter: <https://gist.github.com/octocat/9257657>

Ignoring files for a repo

Add the file(s) name to `.git/info/exclude`

NOTE: This only affects that repository, and should only be used for files you don't want in the repos ignore file.

Always name remotes

When doing pushes or pulls always name the remote server and branch.

```
$ git pull <remote> <branch>
```

```
$ git push <remote> <branch>
```

With Git version 2.x this becomes even more important.

But that is hard!

- That is extra typing that I have to do!
- I only ever work with one remote/branch anyways!
- etc...

Solution

```
function current_branch() {  
  ref=$(git symbolic-ref HEAD 2> /dev/null) || \  
  ref=$(git rev-parse --short HEAD 2> /dev/null) || return  
  echo ${ref#refs/heads/}  
}
```

```
# these aliases take advantage of the previous function  
alias ggpull='git pull origin $(current_branch)'  
alias ggpur='git pull --rebase origin $(current_branch)'  
alias ggpush='git push origin $(current_branch)'
```

Autocorrect

```
$ git plush origin master  
git: 'plush' is not a git-command.  
See 'git --help'.
```

```
Did you mean this?  
push
```



To have Git fix this

```
$ git config --global \ help.  
autocorrect = 1
```

Removing whitespace

Create a `$HOME/.config/git/attributes` file and add:

** filter=trimWhitespace*

Removing whitespace

Next we need to tell Git about this filter

```
$ git config --global \ filter.  
trimWhitespace.clean \ trim_whitespace
```

Removing whitespace

Now create the “trim_whitespace” command

```
#!/usr/bin/env ruby
lines = STDIN.readlines
lines.each do |line|
  puts line.rstrip
end
```

Prettier log output

Add the following to `$~/.gitconfig` under the `[alias]` section

```
Lg = Log --color --graph \ --pretty=format:'%  
Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%  
cr) %C(bold blue)<%an>%Creset' --abbrev-commit  
--
```

What does that do

```
0 vendion 0.00, 0.01, 0.05 1:*
* 2a87baa - (HEAD, origin/master, origin/HEAD, master) Major update to how things are started (13
* a4321ad - Updating gitconfig file (3 weeks ago) <Adam Jimerson>
* d02be8a - Remove TrueNorth shortcut (3 weeks ago) <Adam Jimerson>
* 5f9c1fd - Have go build work on a directory level (3 weeks ago) <Adam Jimerson>
* 562b846 - Add oauth key for Github-issues.vim (3 weeks ago) <Adam Jimerson>
* 7130d36 - Remove quicktask.vim plugin (3 weeks ago) <Adam Jimerson>
* e336a47 - Use the correct rule to force Chrome app menu to float (3 weeks ago) <Adam Jimerson>
* d01a320 - Define the <C-d> mapping always (4 weeks ago) <Adam Jimerson>
* adf0b12 - Updating Vim Spell file (4 weeks ago) <Adam Jimerson>
* 5292ce4 - Try to target the Chrome App Menu (4 weeks ago) <Adam Jimerson>
* af206a3 - Force Chrome windows into floating mode (4 weeks ago) <Adam Jimerson>
* 2dd3570 - Only set vimpager as the PAGER if it exists (4 weeks ago) <Adam Jimerson>
* 2521020 - Adding new fpath directory in OSX (4 weeks ago) <Adam Jimerson>
* 8289cc8 - Updating vim spell files (5 weeks ago) <Adam Jimerson>
* f414f05 - Adding new plugins (5 weeks ago) <Adam Jimerson>
* c2a3ad9 - Work around vim's syntax failing every now and then (5 weeks ago) <Adam Jimerson>
* 76c4c50 - Remove settings for doxygen (5 weeks ago) <Adam Jimerson>
* aa054e5 - Updating spell files (5 weeks ago) <Adam Jimerson>
* a689cc7 - Tell Git to ignore vim session files (6 weeks ago) <Adam Jimerson>
* e0df789 - Set ANDROID_HOME environment var (6 weeks ago) <Adam Jimerson>
* 9b936ab - Updating Vim's spellfiles (6 weeks ago) <Adam Jimerson>
* b156071 - Updating Vim spellfiles (7 weeks ago) <Adam Jimerson>
* c50bd61 - Merge branch 'CtrlSpace-integration' (7 weeks ago) <Adam Jimerson>
|
| * 033ff55 - Set custom settings for CtrlSpace (7 weeks ago) <Adam Jimerson>
| * 0527f49 - Add CtrlSpace to Vim's plugins (8 weeks ago) <Adam Jimerson>
| * beef710 - Update Vim spellfile (7 weeks ago) <Adam Jimerson>
|
| * b1d477c - Set PAGER to vimpager (8 weeks ago) <Adam Jimerson>
* 2d2f2ee - Updating spell files (9 weeks ago) <Adam Jimerson>
* 6697c66 - Settings for YAML files (3 months ago) <Adam Jimerson>
* 0104415 - Adding Twig plugin for VIM (3 months ago) <Adam Jimerson>
* cb8dc27 - Change keymap for pdv (3 months ago) <Adam Jimerson>
:
```

Another log

```
alias gLogf='git log --graph --color'
```

What does that do?

```
⓪ □ vendion □ 0.05, 0.03, 0.05 1:*
* commit 2a87baade907d9dcb74c144ae7e9c583df651471
  Author: Adam Jimerson <vendion@gmail.com>
  Date: Mon Aug 4 21:55:44 2014 -0400

    Major update to how things are started

    Signed-off-by: Adam Jimerson <vendion@gmail.com>
* commit a4321ad07a0c6a8f03afe549237f3ac50d197ac2
  Author: Adam Jimerson <vendion@gmail.com>
  Date: Mon Jul 28 10:52:08 2014 -0400

    Updating gitcofig file

    Updating the Git config file.
* commit d02be8a43364c4f57e45b324b31cfa2d914cd561
  Author: Adam Jimerson <vendion@gmail.com>
  Date: Mon Jul 28 10:24:13 2014 -0400

    Remove TrueNorth shortcut

    Signed-off-by: Adam Jimerson <vendion@gmail.com>
* commit 5f9c1fd11add1c6dd29096668fa4be80dc101992
  Author: Adam Jimerson <vendion@gmail.com>
  Date: Sat Jul 26 23:30:46 2014 -0400

    Havge go build work on a directory level

    Signed-off-by: Adam Jimerson <vendion@gmail.com>
* commit 562b846b2bc2421868bdb60249079291e02d9a6d
:
```

Handling multiple emails

- What if you have repos you need associated with different email addresses?
- Edit *.git/config* file for each repository manually
- Create a Git command to set email addresses for you.

Profile command

In the global Git config file add the following under the `[alias]` tag

```
workprofile = config user.email      \"  
adam@codejourneymen.com\"
```

Then run

```
$ git workprofile
```


Speed up slow net

If you have problems with slow network connections. Edit `~/.ssh/config` add:

ControlMaster auto

ControlPath /tmp/%r@%h:%p

ControlPersist yes

Stop working around Git

Git implements several commands that interact with the filesystem as well as its own tracking info.

- `mv => git mv`
- `cp => git cp`
- `rm => git rm`

Moving files

```
$ git mv <oldFilename> <newFilename>
```

is the same as

```
$ mv <oldFilename> <newFilename>
```

```
$ git add <newFilename>
```

Copying files

```
$ git cp <original> <copy>
```

is the same as

```
$ cp <original> <copy>
```

```
$ git add <copy>
```

Removing files

```
$ git rm <filename>
```

is the same as

```
$ rm <filename>
```

```
$ git rm <filename>
```

Recovering/Restoring Files

Discarding changes

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

Rolling a file back

```
$ git checkout master~N <file>
```

Working on all files with a certain extension

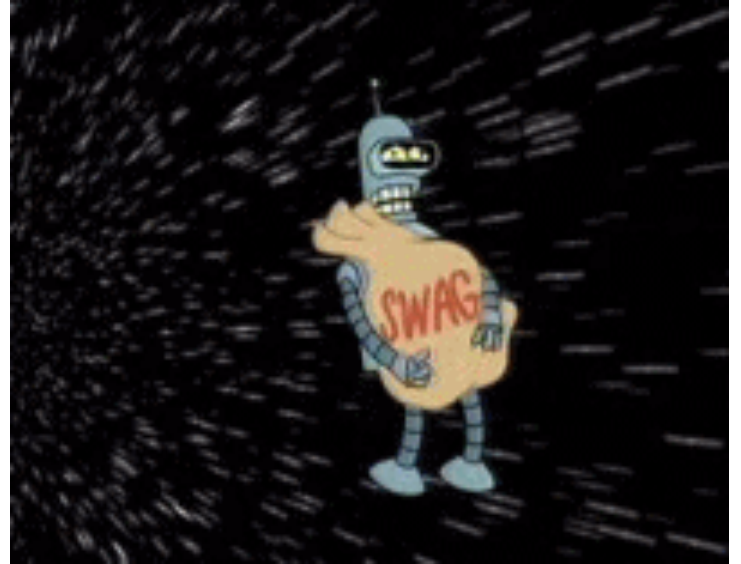
```
$ git checkout -- '*.php'
```

And now for something completely different...



Branching

- How to work with branches.
- Why you should work with branches.



What is a branch anyways?

A branch is a copy of the code base, where changes can be made that doesn't affect copies.

*Very simple explanation



Listing branches

Using the branch command with no arguments displays a list of branches and marks the current branch

```
$ git branch  
develop  
*master
```

Creating branches

Create a new branch by giving a single argument to branch

```
$ git branch <name>
```

Switching branches

To switch branches give the name of the branch as an argument to checkout

```
$ git checkout <branch_name>
```

Doing both at once

To create and switch to the branch

```
$ git checkout -b <name>
```

Deleting a branch

To delete a branch after it has been merged

```
$ git branch -d <name>
```

To delete a branch without merging

```
$ git branch -D <name>
```

Recovering deleted branch

```
$ git reLog
```

```
793d399 HEAD@{0}: rebase finished: returning to refs/heads/develop
```

```
793d399 HEAD@{1}: rebase: checkout feature/test2
```

```
2d1a343 HEAD@{2}: checkout: moving from feature/test2 to develop
```

```
793d399 HEAD@{3}: checkout: moving from feature/test1 to feature/test2
```

```
$ git checkout -b <branch> HEAD@{N}
```

Working with branches

- Separate code changes when adding a feature or making a change.
- Easier context switches.

Squashing commits

Say you have two commits that really should have been one. What can you do?



Word of warning

Don't do the following if a push has been done between the commits being squashed/merged.

If you do try this things are guaranteed to break.

Word of warning



git commit --amend

```
$ git add file1 file2
```

```
$ git commit -m 'Adding some files'
```

```
...
```

```
$ ls
```

```
file1 file2 file3
```

git commit --amend

```
$ git add file3
```

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

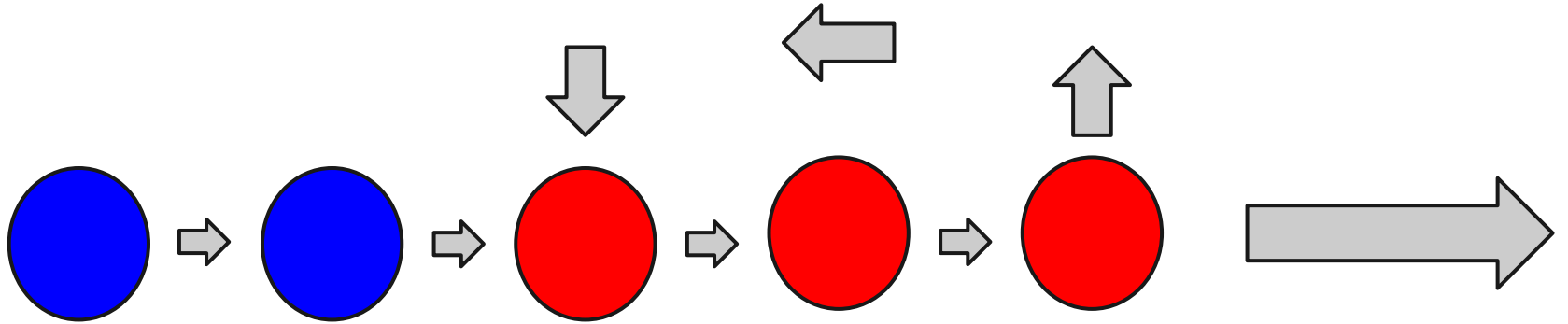
Merging commits

```
$ git rebase --interactive HEAD~2
```

Warning about rebase

- Rebasing alters the history of the repository.
- Constantly mixing merges and rebases can cause issues with upstream repos.

Yay visuals!



Rebasing commits

```
1 [ ] vendion [ ] 0.02, 0.02, 0.05 [ ] [ ] 1:vim - rebase-merge* [ ]
git-rebase-todo+ [ ]
1 pick f3ec368 committing test4 file.
2 squash 2d1a343 adding missing file
3
4 # Rebase b6e4bbe..2d1a343 onto b6e4bbe
5 #
6 # Commands:
7 #   p, pick = use commit
8 #   r, reword = use commit, but edit the commit message
9 #   e, edit = use commit, but stop for amending
10 #   s, squash = use commit, but meld into previous commit
11 #   f, fixup = like "squash", but discard this commit's log message
12 #   x, exec = run command (the rest of the line) using shell
13 #
14 # These lines can be re-ordered; they are executed from top to bottom.
15 #
16 # If you remove a line here THAT COMMIT WILL BE LOST.
17 #
18 # However, if you remove everything, the rebase will be aborted.
19 #
20 # Note that empty commits are commented out
```

Merging branches

```
$ git checkout <branch to merge into>
```

```
$ git merge <branch to merge>
```

Merging branches

```
1  vendion  0.00, 0.01, 0.05  1:..ects/LearnGit*
vendion@Beyla ~/Projects/LearnGit <feature/test1>
└─> git checkout develop
Switched to branch 'develop'
vendion@Beyla ~/Projects/LearnGit <develop>
└─> git merge feature/test1
Updating 40e77c7..2d1a343
Fast-forward
 test1 | 1 +
 test2 | 1 +
 test3 | 1 +
 test4 | 1 +
 test5 | 1 +
 5 files changed, 5 insertions(+)
 create mode 100644 test4
 create mode 100644 test5
vendion@Beyla ~/Projects/LearnGit <develop>
└─>
```

Rebasing branches

```
$ git checkout <branch to merge into>
```

```
$ git rebase <branch to merge>
```

Rebasing branches

```
1 vendion 0.00, 0.01, 0.05 1:..ects/LearnGit*
└─> git checkout develop
vendion@Beyla ~/Projects/LearnGit <feature/test2>
└─> git checkout develop
Switched to branch 'develop'
vendion@Beyla ~/Projects/LearnGit <develop>
└─> git rebase feature/test2
First, rewinding head to replay your work on top of it...
Fast-forwarded develop to feature/test2.
vendion@Beyla ~/Projects/LearnGit <develop>
└─>
```

Merging vs Rebasing

- There are two camps about this matter.
- Merging keeps the commit structure (branch info) intact, but creates empty commits.
- Rebasing flattens the commit structure, and avoids creating empty commits.

Finding bugs (and who introduced them)

Useful Git tools:

- `git bisect`
- `git blame`

Git Bisect

```
$ git bisect start <bad> <good>
```

```
$ git bisect bad or $ git bisect good
```

```
$ git bisect reset
```

This is just the start of what bisect can do!

Git Blame

```
$ git blame <file or commit SHA>
```

Thank you!

