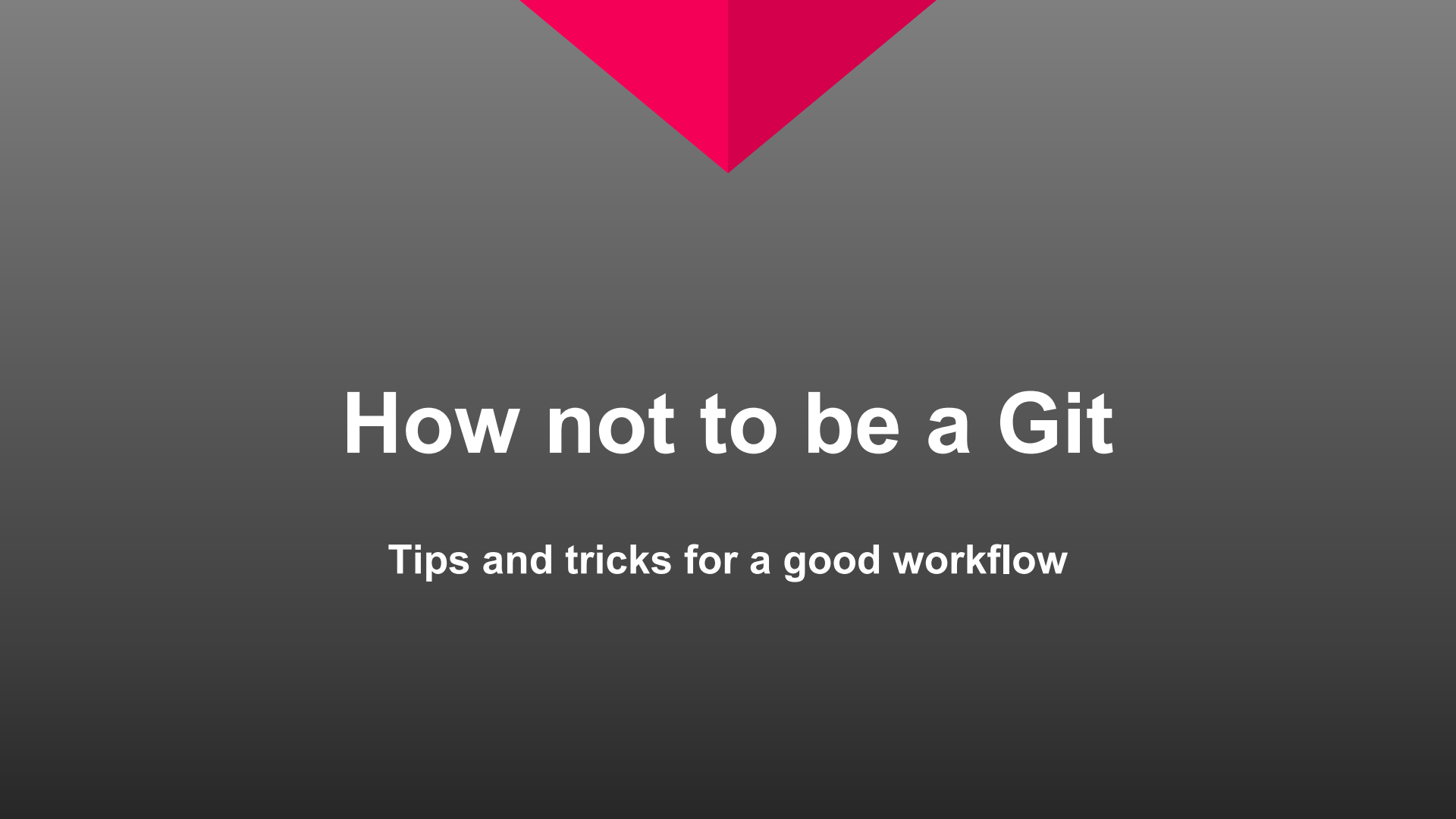

How not to be a Git
Tips and tricks for a good workflow

Who am I?

- Adam Jimerson
- System Architect @ Utiliflex
- PacBSD Developer
- vendion@gmail.com
- https://google.com/+AdamJimerson
- https://vendion.me

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)

Microsoft + GitHub

Still lost?

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

Let’s start with tips

Listing tracked files

List all tracked files

$ git ls-files

List all tracked files in a given 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

Gitignore templates: https://www.gitignore.io/

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>

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

```
* aba0786 - (HEAD -> master, origin/master, origin/HEAD) correctly target flexbox from view (46 minutes ago) <Lee Walker>

* 86c5c99 - Updated ...block--field-social-source.tpl.php (19 hours ago) <Adam Jimerson>
```

Another log

alias glogf='git log --graph --color'

What does that do?

```
*   commit 437a491f99f30e14ecb63af6f07e540af3fd9e00
|\  Merge: 770a155 defc9bd
| | Author: John Smith <jsmith@example.com>
| | Date:   Thu Aug 13 16:15:22 2015 -0400
| |
| |     Merge branch 'master' of
ssh://codeserver.dev.6ad151bf-f855-4e85-b698-52983a55a
2d2.drush.in:2222/~/repository
```

Diffing for Humans

Diff-So-Fancy: https://github.com/so-fancy/diff-so-fancy

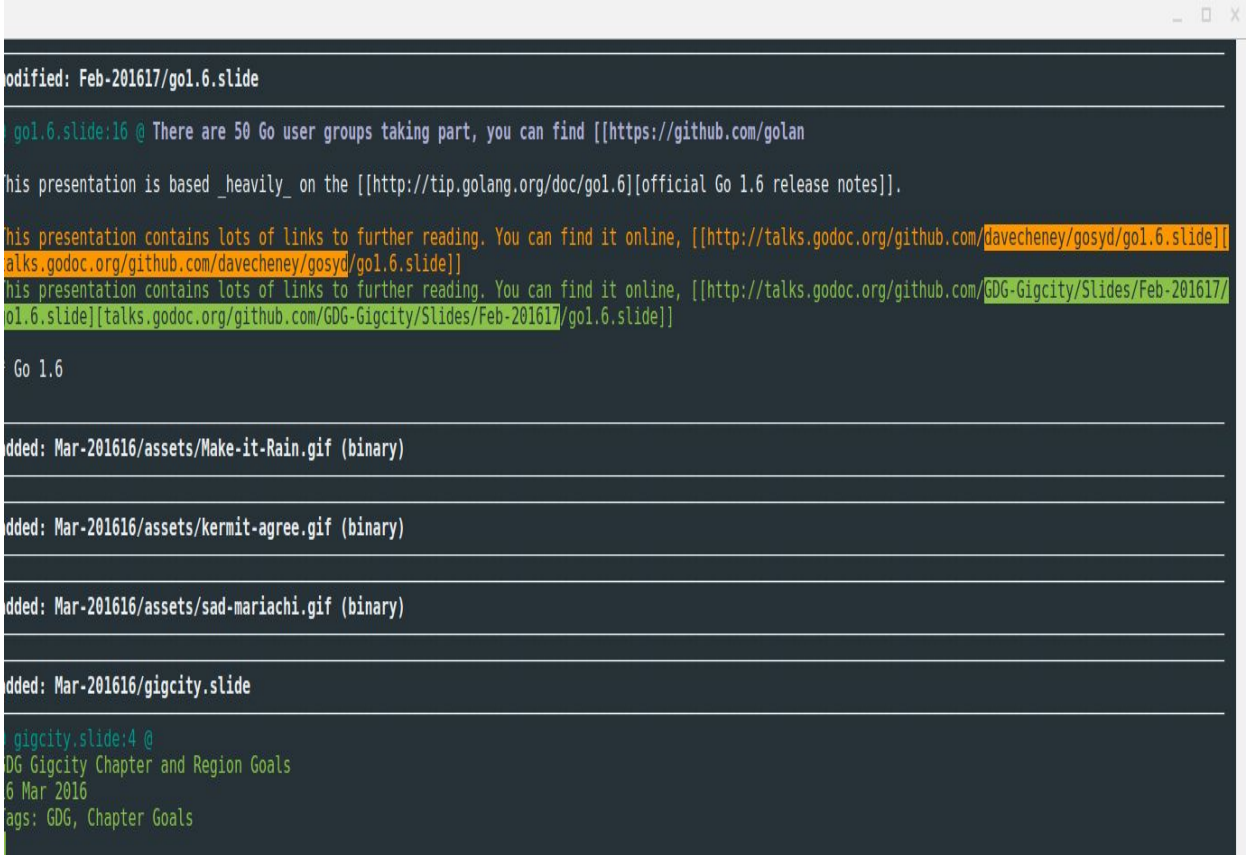
$ git diff --color | diff-so-fancy

OR

$ git config --global interactive.diffFilter = 'diff-so-fancy'

(Requires Git 2.[6,7,8])

What it looks like



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
- rm => git rm

Moving files

$ git mv <oldFilename> <newFilename>

is the same as

$ mv <oldFilename> <newFilename>

$ git add <newFilename>

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’*

Copying files from one branch to another

To copy files or directories from one branch to the current branch

$ git checkout <branch> -- <file>


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 reflog
```

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

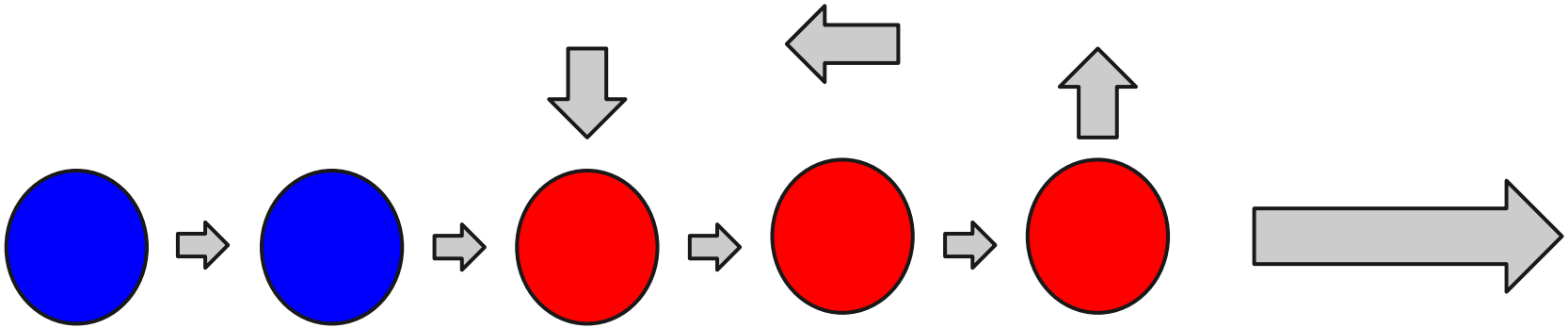
Merging commits

$ git rebase --interactive HEAD~2

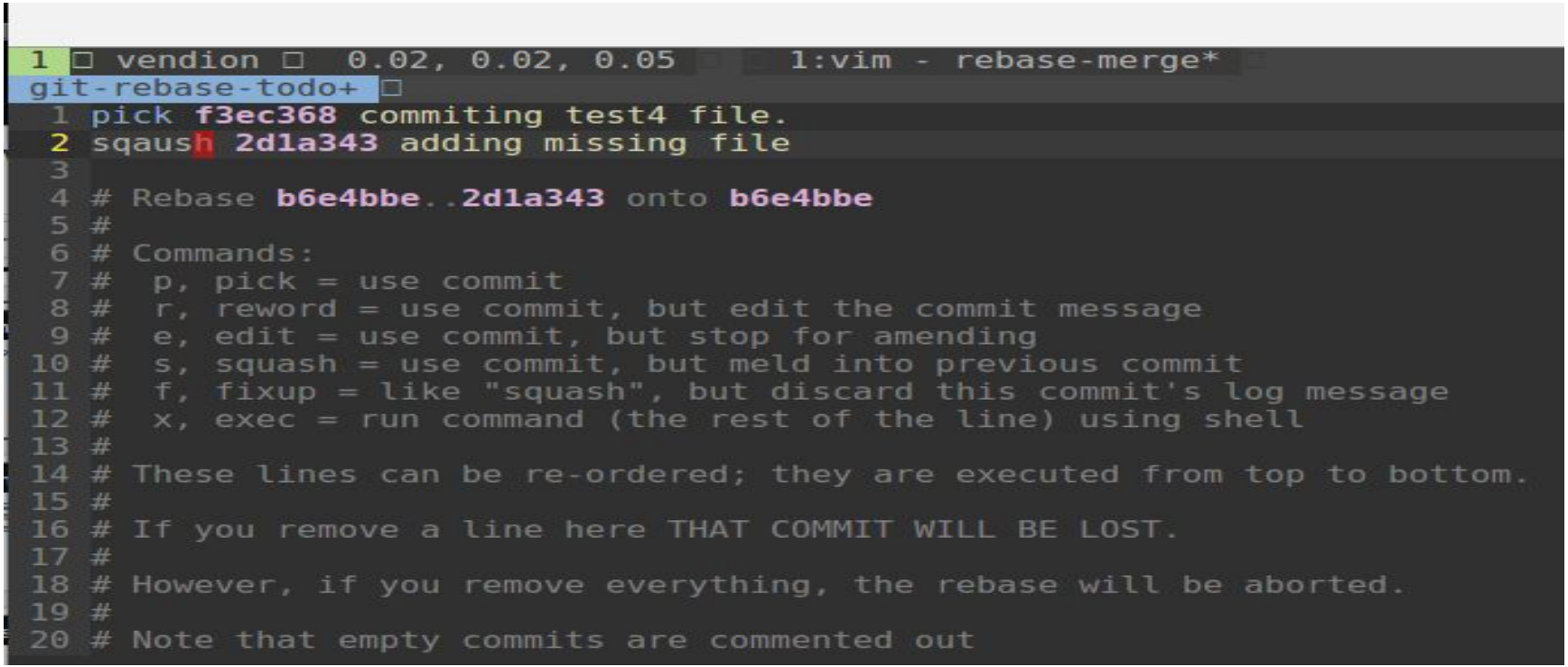
Warning about rebase

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

Yay visuals!



Rebasing commits

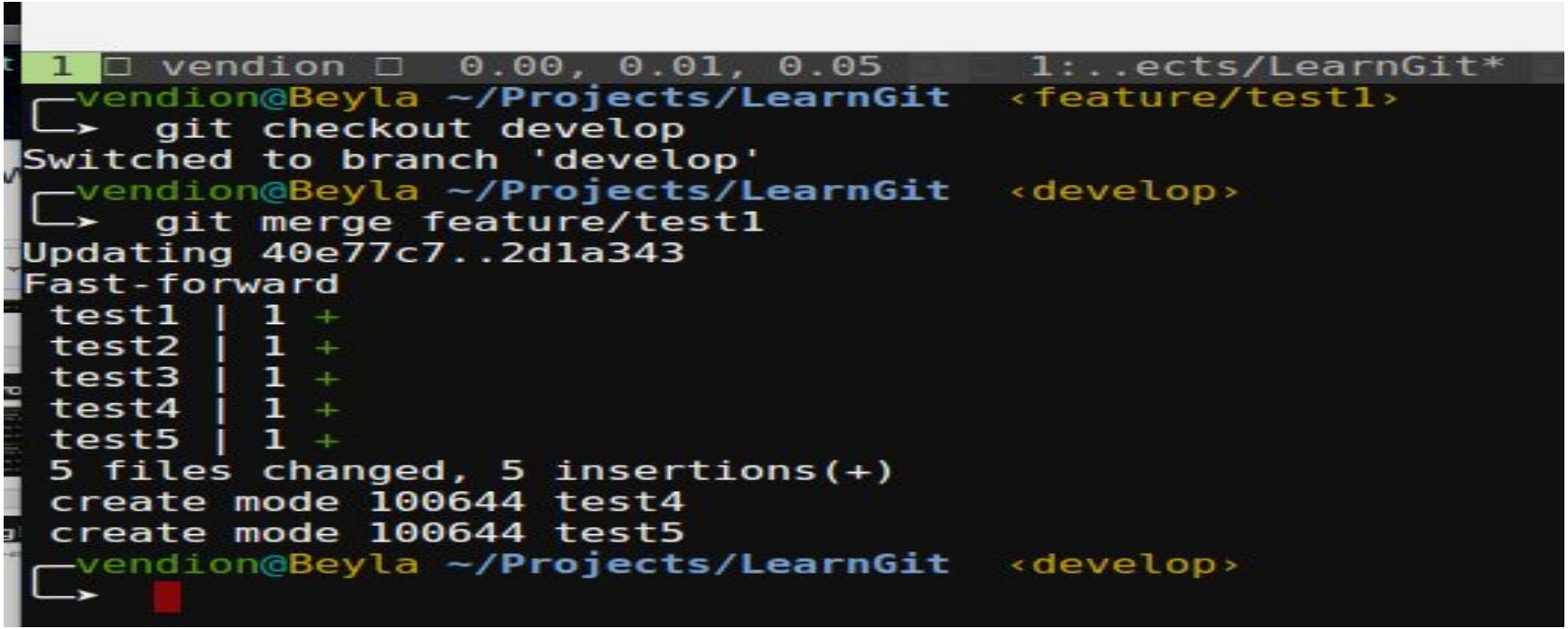


Merging branches

$ git checkout <branch to merge into>

$ git merge <branch to merge>

Merging branches

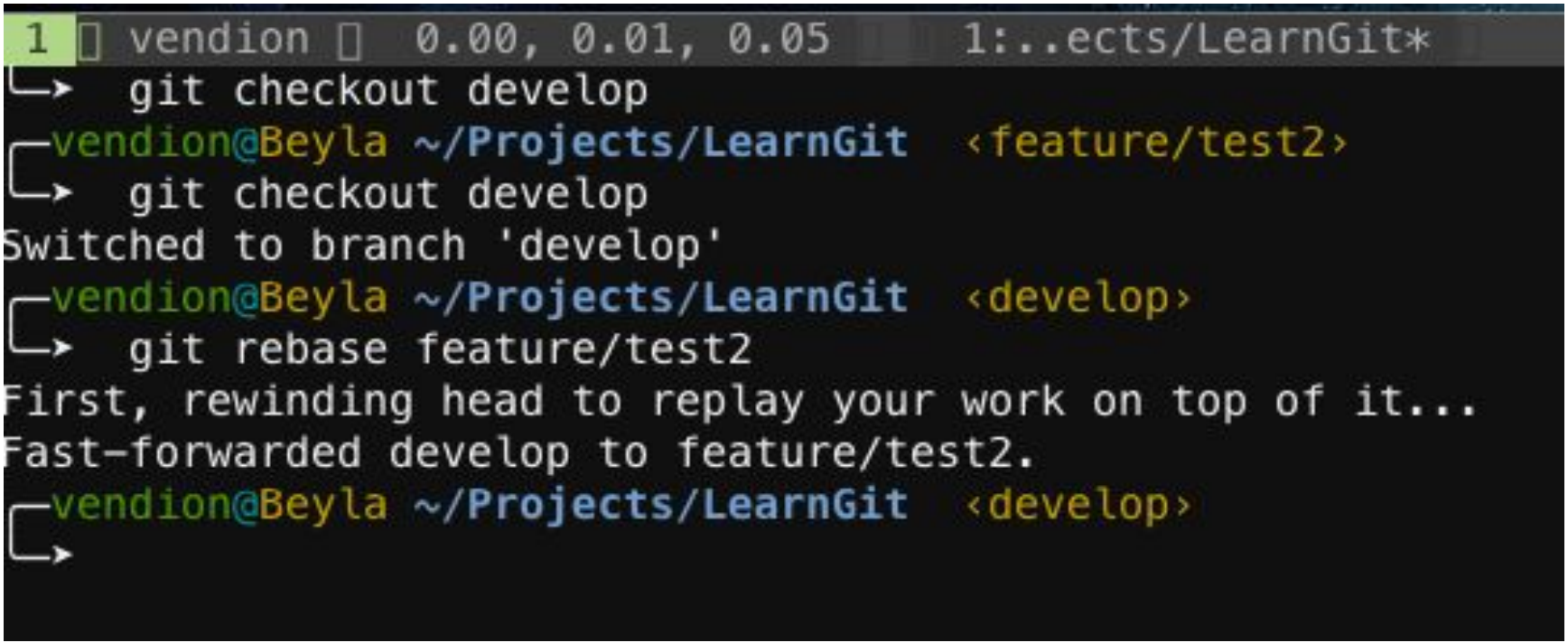


Rebasing branches

$ git checkout <branch to merge into>

$ git rebase <branch to merge>

Rebasing branches



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!