```
## git --learning --curve


#   difficulty
#
#                ----  nirvana
#              /    \
#             /      \
#            /        ------
#           | conflicts!?#  \
#           | rebase!#       \
#          /                  \
#         ? reset?!             -------
#        / undo?
#       / reset?!
#       | index?
#       | dag?
#   ----
#
#               time
#


# Disclaimer:
# - I'm not an expert
# - This should benefit beginners and
#     intermediates
# - This covers doubts I had and I was asked
#     however basic it sounds
# - Try to ask questions at the end
# - Try not to treat Git or SCM as secondary
```

```
## git log --gargantuan --intimidating --terrifying


# log has 100+ options
# git log man is -git

$ git log -git
          -tig
          -g -i -t
          -g -t -i
```

consoles

git bash
posh-git
conemu
tortoisegit
...

What I use?
git bash
scite text editor
winmerge2011 (diff, merge)

```
## configs


# system
# global
# local

# edit
$ git config --local -e &
$ git config --global -e &

# global config
# editor
# winmerge
# set some options in configs
# include path
```

```
## aliases


# The freedom to make my own ~~mistakes~~ aliases
# is all I ever wanted.
#               - Mance Rayder
#                 Game of Thrones

# git commit
$ git c

# git checkout
$ git co

# git status -s|--short
$ git s

# log aliases
# git log --pretty=twoline --graph \
#   --decorate --abbrev-commit --date=relative
$ git l -1

$ git alias alias
$ git alias conf

# alias with ! run at root level
$ cd to/some/dir
$ git l .
```

```
## data structures


special refs: head, fetch_head
    branch

refs: branch, remote, tag:
    commit

commit:          => hash
    (tree, hash)
    (parent, hash)[]
    author, name, email, date
    committer, name, email, date

tree:            => hash
    (type, hash, name)[]
    # where type is blob|tree

blob:            => hash
    binary

# cat
$ git cat-file -p <commit>|<tree>|<blob>
```

```
## DAG


#   root
#   o<---o<---A<---x master
#         \       /
#          `--B<-´
#             dev
#

# commits are shown as either of following:
# o
# *
# letters A, B, C ...

# merge commits are shown as:
# x
```

```
## workings


# work offline
# commit represents the whole repo at that point
# for diff, only the commits involved are required


#  o---A---o---o---B
```

```
## upstream


#           upstream
#
#                --o---B origin/master
#               /
#              /
#   ---o---o---A origin/master
#              master
#
#           downstream

$ git fetch

#           upstream
#
#                          origin/master
#
#   ---o---o---A----o---B origin/master
#             master
#
#           downstream

(master)
$ git reset --hard master@{u}
                   origin/master
```

```
## index


# commit in flight
# break a change into logical commits
# exclude a whole file or a *part* of it

# show not added to index
$ git difftool

# show added to index
$ git difftool --staged|--cached

# show both, not added and added to index
$ git difftool head

#                                          working dir
#  HEAD|          git diff head            |
#   ---o-----------------staged---------unstaged
#      | git diff --staged |  git diff  |
#
```

```
## reset


#                  topic
#   ---o---A-------B
#                  working dir

$ git reset --hard head~

#          topic
#   ---o---A-------B
#          working dir

# B is unstaged
$ git reset [--mixed] head~

#          topic
#   ---o---A-------B (unstaged)
#                  working dir

# B is staged
$ git reset --soft head~

#          topic
#   ---o---A-------B (staged)
#                  working dir


#        move      change     working
#        branch    index      dir
#  soft     Y
# mixed     Y        Y
#  hard     Y        Y         Y
```

```
## diff


# head / working dir
$ git diff
$ git difftool

# commit / working dir
$ git difftool master

# commit1 / commit2
$ git difftool master dev

# commit1 / commit2 for file.txt
$ git difftool master dev dir/file.txt
$ git difftool master dev -- "dir/file.txt"

# commit1:file1 / commit2:file2
$ git difftool dev:README.md master:rm.Extensions/Heap.cs

# diff of last 2 commits combined w/ head
$ git difftool head~2 head
# diff of last 2 commits combined w/ working dir
$ git difftool head~2
# diff of last 2 commits combined w/ staged working dir
$ git difftool head~2 --staged

# char level diff
$ git diff --color-words=.
```

```
## ancestry (parent) refs


# ^     parent by breadth
# ~     parent by depth

# ^     1st parent (same as ^1)
# ~     1st parent (same as ~1)
# ^2    2nd parent
# ~2    grandparent (1st parent's 1st parent)

# ^!    no parents (roughly, ^ for parent, ! for no)

# How to see a diff of a commit?
$ git diff 4a5b6c7~ 4a5b6c7
$ git diff 4a5b6c7^!
```

```
## commit ranges


# What's new on origin/master?
(master)
$ git log ..origin/master

# double dot, triple dot

#               dev..master = { d }
#
#   a---b---c---d master
#            \            dev...master = { d, e }
#             --e dev
#
#               master..dev = { e }
#
#               dev master = { a, b, c, d, e }

# .. in e, but not in d
$ git log d..e

# ... in d or in e but not both
$ git log d...e

# in d or e or both
$ git log d e
```

```
## commit --amend


$ git commit

#              topic
#   ---o<--o<--A
#

$ git commit --amend

#   ---o<--o<--A
#           \
#            A' topic
```

```
## stash


# -u includes untracked files
$ git stash save -u "comment"

# pop will remove if no conflicts, apply does not
$ git stash apply|pop

# v2.11+, no need to type stash@{1}
$ git stash apply|pop 1

#      // this is the stashed commit
#
#      *-.   WIP on dev
#      |\ \
#      | | |
#      | | * untracked files on dev
#      | |
#      | |
#      | * index on dev
#      |/
#      * (dev)        // point from where `git stash save -u` is ran
#

# use --index to restore index
$ git stash apply 3 --index
```

commit message convention

50/72:

```
    subject is 50 chars only

    body is 72 chars wide. this needs to wrap at 72 chars
    like so.
```

50/*:

```
    subject is 50 chars only

    body is freeform. this can go on and on and on and on and on and on.
```

$ git log --oneline -1
8624263 (HEAD -> dev) subject is 54 chars only

Please do not do this:

```
    my subject is more than 54 chars so i decided to break
    it down in this horrible, terrible, criminal way

    commit body.
```

$ git log --oneline -1
8624263 (HEAD -> dev) my subject is more than 54 chars so i decided to break
it down in this horrible, terrible, criminal way

There are exceptions:
- Auto-generated messages
- Quoted text

```
    Merge remote-tracking branch 'origin/really-long-branch-name' into next

    commit body.
```

Commits are forever, branches are not.
- Try not to make typos in commits
- Try not to make mistakes by putting incorrect ticket

```
## branch


# What branches is my branch merged into?
$ git fetch
$ git branch -r --contains origin/branch

# tracking branch info
$ git branch -v  # verbose
$ git branch -vv # very verbose

# force move a branch
$ git branch -f <branch> <commit|branch>
$ git branch -f dev origin/dev
```

```
## checkout/clean


# undo all unstaged changes
$ git checkout .

# undo a file's unstaged changes
$ git checkout file.txt

# delete untracked files, dirs, build files
$ git clean -f [-d] [-x]

# delete interactive
$ git clean -i

# alias: git undo
$ git reset --hard && git clean -f

# to delete dirs also
$ git undo -d
```

```
## merge


#   ---o---o master
#       \
#        --o
#          dev

(master)
$ git merge dev

#   ---o---o---x master
#       \     /
#        --o--
#          dev

(master)
$ git merge pu maint

#   ---o---o---x master
#      |\     /|
#      | --o-- | pu
#       \     /
#        `-o--' maint

# Linux kernel has ~~66~~ branches merged together
# in 2014
```

```
## --ff (fast-forward) merge


#   ---o master
#       \
#        --o---o
#              dev
#

(master)
$ git merge dev

#   ---o
#       \      master
#        --o---o
#              dev
#

# Where was master previously?
```

```
## --no-ff merge


#   ---o master
#       \
#        --o---o
#              dev

(master)
$ git merge dev --no-ff

#   ---o---------x master
#       \       /
#        --o---o
#              dev

# .gitconfig
[merge]
    ff = false

# --no-ff flag default
$ git merge dev

# for ff merge
$ git merge dev --ff

# .gitconfig
[branch "master"]
   mergeOptions = --no-ff
# same for next, dev
```

```
## merge conflicts


# There are 3 things certain in life.
(death)
$ git merge taxes
<<<<<<< HEAD
death
||||||| merged common ancestors
merge conflicts
=======
taxes
>>>>>>> taxes


(master)
$ git merge dev

# conflicts
$ git status -s
UU a.txt # conflict (Unmerged, Updated)
 M b.txt # modified
 A c.txt # added
 R d.txt # renamed
 D e.txt # deleted

# get current|other branch's copy
$ git difftool --ours|--theirs file.txt

# get current branch's copy
$ git checkout --ours   file.txt
               --theirs
```

```
# undo merge, even a successful merge
$ git reset --merge orig_head
# use again to go back to merge commit

[alias]
    undomerge = reset --merge orig_head
$ git undomerge

# How to see resolved conflicts before push?
$ git show head
diff --cc file.txt
- my change
 -other change
++resolved change

# this does not work
$ git difftool head^!
```

rebase

```
#              master
#   ---o---o---o
#       \
#        A---B topic

(topic)
$ git fetch
$ git rebase master

#              master
#   ---o---o---o
#       \       \
#        A---B   A'---B' topic
```

```
## why rebase?


#                       master
#   ---o---o---o---o---x
#       \       \     /
#        A---B---x---C topic

(topic)
$ git commit
$ git commit
$ git merge master
$ git commit
(master)
$ git merge topic

# merge commits do not add value
# difficult to read history

(topic)
$ git rebase master
#                   master
#   ---o---o---o---o
#       \           \
#        A---B---C   A'---B'---C' topic
```

```
## rebase -i


# Commit often, perfect later.

#               master
#   ---o---o---o
#               \
#                A---B---C---D topic

(topic)
$ git rebase -i master|head~4

pick A
pick B
pick C
pick D

# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous
commit
# f, fixup = like "squash", but discard this commit's
log message
# x, exec = run command (the rest of the line) using
shell
# d, drop = remove commit
```

```
# scenario 1: edit, pick, drop

edit A
pick B
drop C
drop D

stopped at A...
(some changes...)
$ git add .
$ git rebase --continue

#              master
#   ---o---o---o
#              |\
#              | A---B---C---D
#              \
#               A'--B' topic


# scenario 2: reorder commits

pick A
pick C
pick B
pick D

#              master
#   ---o---o---o
#              |\
#              | A---B---C---D
#              \
#               A'--C'--B'--D' topic
```

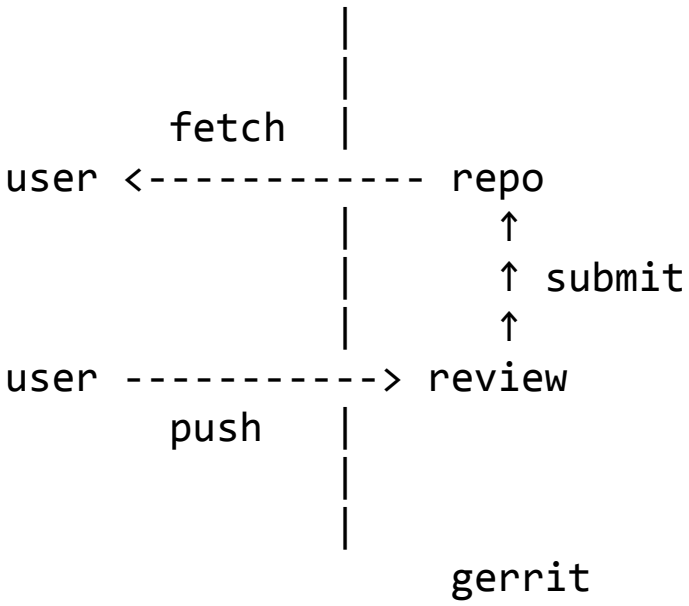
```
# scenario 3: pick, fixup, reorder, drop

pick A
fixup C
pick B
#edit D (dropping D by commenting)

# results in
AC (commits A+C)
B  (commit B)

#               master
#   ---o---o---o
#              |\
#              | A---B---C---D
#               \
#                AC'--B' topic
```

```
## push after rebase


(topic)
$ git push
$ git fetch
$ git rebase master

#               master
#   ---o---o---o
#       \       \
#        \       A'---B' topic
#         \
#          A---B topic{-1}
#                origin/topic

$ git push topic
rejected   (non-fast-forward)
do a pull/merge first

$ git push -f

#               master
#   ---o---o---o
#       \       \
#        \       A'---B' topic
#         \               origin/topic
#          A---B
```

```
# golden rule of rebase


# I normally don't rebase a branch,
# but when I do, I do master.
#     - The Most Interesting Rebaser in the World
#       #yolo


# don't rebase public branches

#      last    hotfix
#   ---o---o---o
#       \
#        A---B master (me, others)

# I rebase master
(master)
$ git rebase hotfix
$ git push -f

#              hotfix      origin/master
#   ---o---o---o---A'---B' master (me)
#       \
#        A---B master (others)
```

```
# Others merge origin/master
(master)
$ git merge origin/master
$ git push

#              hotfix
#   ---o---o---o---A'---B'
#       \                \  origin/master
#        A---B------------x master (me, others)

# Merge is better compared to rebase

# I merge master
(master)
$ git merge hotfix
$ git push

#              hotfix
#   ---o---o---o
#       \       \
#        A---B---x master (me, others)
#                origin/master

# Others can fast-forward master now
```

gerrit



```
## git review


# this will push to branch rm/1234-topic
(rm/1234-topic)
$ git review

# push to gerrit for code review
git review:
    branch = $1 ?? currentBranch()
    if <branch> == null:
        throw "gerrit requires branch"
    if !gerrit-exists-branch <branch>:
        gerrit-create-branch master:<branch>
    git push origin head:refs/for/<branch>
```

```
# rebase and gerrit


# merge master in to topic
(topic)
$ git fetch
$ git merge origin/master
# resolve conflicts if any
# push
$ git review

# Problems with merge:
# - Logic in merge commits with conflicts

# rebase topic on master
(topic)
$ git fetch
$ git rebase origin/master
# resolve conflicts if any
# push
$ git review
```

```
## gerrit and push -f


# no force push rights - why?
# to avoid bypassing reviews
$ git push -f
( ! [remote rejected])
```

git log --name-status|--name-only

show file names and statuses

```
$ git log -1 --name-status
commit 714a14ef23acb056e3ad0b03c2fd515e3c308603 (HEAD -> dev, github/dev)
...
M   README.md
A   rm.Extensions/BitSet.cs
M   rm.Extensions/Properties/AssemblyInfo.cs
M   rm.Extensions/rm.Extensions.csproj
A   rm.ExtensionsTest/BitSetTest.cs
M   rm.ExtensionsTest/rm.ExtensionsTest.csproj
```

show file names

```
$ git difftool head~ --name-only
```

```
## grep, log grep, ls-files


# greps in working directory only
$ git grep "woohoo"
$ git grep -i "woohoo"         # ignore case
$ git grep "woohoo" -- *.gtt   # in .gtt files only

# greps in entire repo (slow)
$ git log -G "woohoo"

# greps in log messages only
$ git log --grep "woohoo"

# greps in tip of all branches (not just current)
$ git branch -a \
    | cut -c3- | xargs git grep "woohoo"
```

git add -i|-p

```
$ git status
(changes)

$ git add -i

# *** Commands ***
#   1: status       2: update       3: revert       4: add untracked
#   5: patch        6: diff         7: quit         8: help
# What now>

(some work to add/remove/patch to index)

$ git commit

# --patch for reset, checkout also
$ git add -p
$ git reset -p
$ git checkout -p
```

```
## rev-list


# get total commit count
$ git rev-list --all --count

# get total commit count for master
$ git rev-list master --count

#   ---o---o---o master
#            \
#             o---o topic

# get left-right commit count
$ git rev-list topic...master --left-right --count
2   1

# get left-right commit
$ git rev-list topic...master --left-right

# Where did I branched from?
# alias: git bfrom
$ git bfrom master@{u} next@{u}
```

```
## template dir and git hooks


# template for when you init a repo
# also works for existing repo
# .gitconfig
[init]
    templateDir = ~/.git-templates

# copy contents of template dir to .git/ dir
$ git init

# to apply git hooks to all repos
[core]
    hooksPath = ~/.git-hooks

# to skip the commit hooks
$ git commit -n
```

```
## piping


# There are more things in git and piping, Horatio,
# than are dreamt of in your philosophy.
#                    - echo "WS" \
#                        | sed 's/WS/R/' \
#                        | awk '{print $0"M"}'
#

# text tools that support piping
sed streaming editor
awk text processing tool

# files with different statuses
$ git status -s | grep "^UU"
$ git status -s | grep "^ M"

# o is shortcut for my text editor, scite

# open conflicted files
$ git status -s | grep "^UU" | cut -c3- | xargs o

# open files with BitSet in name
$ git ls-files *BitSet* | xargs o
```

```
# alias hidden: show hidden files
$ git ls-files -v | grep '^h' | cut -c3-
$ git hidden

# hide and unhide are aliases
$ git hide
$ git unhide

# hide modified files and then unhide them
$ git status -s | grep "^ M" | xargs git hide
$ git hidden | xargs git unhide

# open files with space in them
$ git ls-files "<sproc>"
    | sed 's/^/"/' | sed 's/$/"/'
    | xargs o

$ git ls-files *<sproc>*
    | awk '{print "\""$0"\""}'
    | xargs o

$ git ls-files -z *<sproc>*
    | xargs -0 o

# 'wrap' wraps all paths in quotes to handle spaces
$ git ls-files "<sproc>" | wrap | xargs o

# 'wrap' is piping-friendly
$ o ~/bin/wrap

# open files with "next"
$ git grep "next" | cut -d':' -f1 | wrap | xargs o
```

```
# delete merged branches except few
$ git branch --merged
    | egrep -v "(\*|master|pu|maint|next)"
    | egrep "cr/"
    | xargs git branch -d

# delete oldest 3 cr/* branches except current
$ git branch --sort=committerdate
    | egrep -v "\*"
    | egrep "cr/"
    | head -3
    | xargs git b -d

# top 2 items
$ ... | head -2
# last 2 items
$ ... | tail -2
# 2nd line item
$ ... | sed -n '2 p'
```

```
## advanced aliases


$ git coe <partial branch>

# multiple branches
$ git coe e
fatal: multiple branches matched
  master
  dev

# fuzzy checkout to branch
$ git coe z
Switched to branch 'temp/lazy'

# custom commands
$ git l
l = git log --pretty=two

# pretty formats
two
one

# aliases running as shell scripts run from /
l = !"git lg2"
# this will still run from /
$ cd to/some/dir && git l .
```

```
## reflog


# <commit> (refs) HEAD@{<i>} summary of operation
#     where <commit> is after the operation

# ref log
$ git reflog -3

# 87dce5f (HEAD -> dev, master) HEAD@{0}: reset: moving to master
# 0733641 HEAD@{1}: rebase -i (finish): returning to refs/heads/dev
# 0733641 HEAD@{2}: rebase -i (pick): Adding BitSet implementation.

$ git reset --hard HEAD@{1}
```

bisect

```
$ git bisect start
# head is bad
$ git bisect bad
# some commit in past is good
$ git bisect good <commit>
(bisecting)
$ git bisect good|bad
(bisecting)
(08b9bd4 is the first bad commit)
# exit bisect state
$ git bisect reset
```

```
## shortlog


#
# The ~~owls~~ stats are not what they seem.
#

# shortlog with summary, sort, email
$ git shortlog -sne

# .mailmap
- consolidate name/emails

# What to do?
- place at root OR
  outside of repo with mailmap.file path in config
- log.mailmap true
```