Reference

19 февраля 2018 г. 14:08

Base

Base Schemes:
- Basic scheme of git:
- Scheme of local repository:
- Scheme of commit:
- Possible Workflows

Base Command details:
- Working with branches
- Scheme of merge:
- Reset
- Checkout
- Rebase
- Push, Fetch, Pull
- Revert, Stash

Syntax:
- Double Dot Syntax
- About ^ and ~

Commit best practices

Note What is Possible

Command reference

configuration
- Configuration
- Graphical Interfaces

remotes
- Clone/Create Repository
- Remotes

Add files
- Add files to staging area
- File movement/Deleting
- Ignoring files
- Commit
- Stash

Branch
- Branch
- Tag
- Merge
- Rebase
- Cherry-pick

Work with remotes
- Push
- Fetch
- Pull
- Revert

View
- Status
- Log
- RefLog
- Difference

Other
- Cleaning History

Scenarios

- Add all files to new repository
- Project start. Work with new github repository
- Remote repository already exist need to start work with it
- Create remote repository locally
- Local Repository exists need to add it to new github
- Local files exists need to add them to new github

- Create custom git command

Links

- Main
- Best Practices
- Other Resources

To Do

12 февраля 2018 г. 18:55

Command reference & Base & test
- [x] Rebase
- [x] Cherry-pick

Git Pro notes & read book till the end
- [x] 5
- [x] 6
- [x] 7
- [x] 8
- [x] 9
- [x] 10

- [x] git reflog

- [x] Add commit requirements
- [x] Add commit workflows

- [x] About commit range selection

- [x] Review Scenarios

Review Lectures
- [x] Lecture 1
- [ ] Lecture 2
- [ ] Lecture 3
- [ ] Lecture 4
- [ ] Lecture 5

- [x] Review all

- [ ] Add to Command Reference reset and checkout
- [ ] Finish exercises on site
- [ ] Read best practices
- [ ] Rebase --onto
- [ ] Review text file

Links

13 февраля 2018 г. 16:43

Main

https://git-scm.com/book/en/v2
https://git-scm.com/book/en/v2/Appendix-C%3A-Git-Commands-Setup-and-Config short summary about main git commands

Other Resources

https://githowto.com/ru
https://www.atlassian.com/git/tutorials
http://gitolite.com/gcs.html#%281%29

http://los-t.livejournal.com/tag/git%20guts like blog

https://learngitbranching.js.org/?NODEMO interesting site where could be practiced some common scenarios

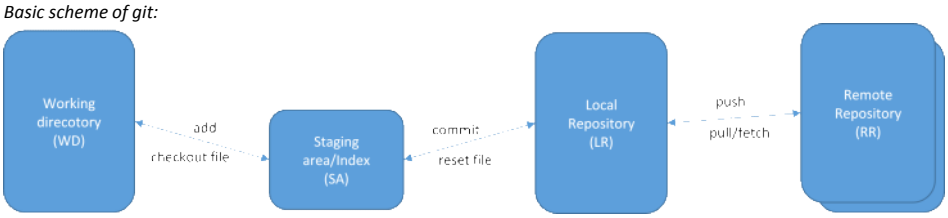
Best Practices

http://sethrobertson.github.io/GitBestPractices/
https://en.wikipedia.org/wiki/Atomic_commit#Atomic_Commit_Convention about atomic commit
https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html about commit message

Base

12 февраля 2018 г. 18:11

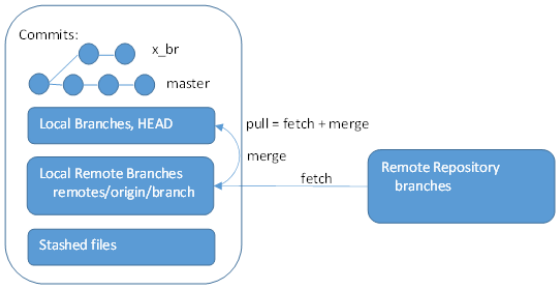
Basic scheme of git:



Working Directory > files which we can see via explorer, here it is possible to change and update them
Staging area > indicates updated files prepared for commit. It is possible to add only some files from WD there
Local Repository > Local version of changes history stored in commits
Commit > snapshot of files and directories

add > adds files to *Staging Area*. Staging the files computes a checksum for each one, store the file (blob) in the repository
commit > move files from *Staging Area* to *Local Repository*. Git checksums each subdirectory > stores that objects in git repository > create metadata & pointer to main tree, so it can recreate WD when needed
push > add files to *Remote Repository*
pull/fetch > copy data from *Remote Repository* to *Local Repository*
reset (file) > copy file from *Local Repository* to *Staging Area*
checkout (file) > copy file from *Staging Area* to *Working Directory*

Scheme of local repository:



Local Repository consist of Commits, Branches, Local Remote Branches, HEAD, Stash and Tags

Commit > saved state of tracked files
Branch > pointer to specific commit
Remote Branches > exists locally, could not be changed directly. They are copies of Remote Repository branches, in case of RR update, after fetch changes appears in the Remote Branches of local repository
HEAD > pointer to the last commit in current branch
Tag > pointer to specific commit or copy of all files, similar to branch but

Scheme of commit:

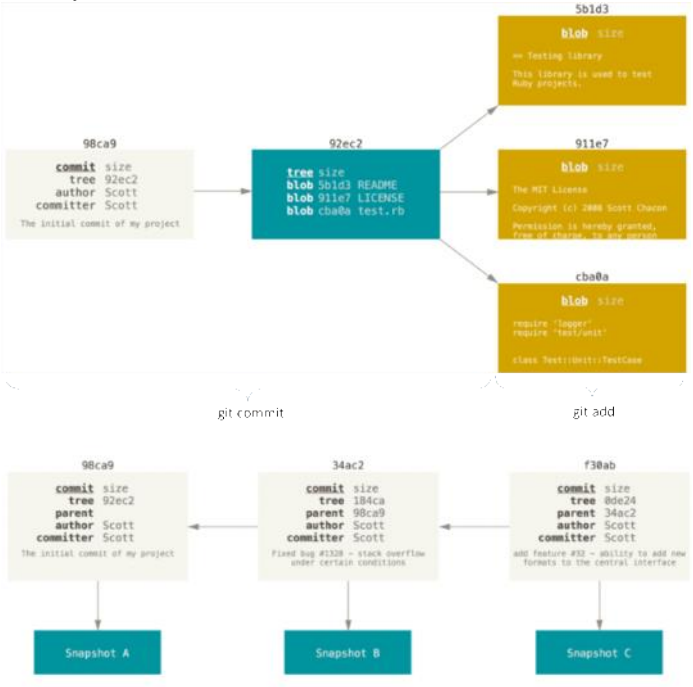


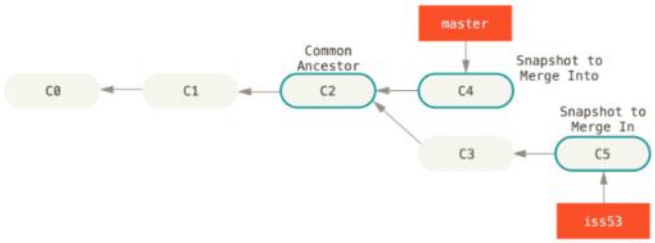
Figure 10. Commits and their parents

Commit > pointer to snapshot of content tree, authors name/email, commit message, pointers to parent commit(s)

Working with branches

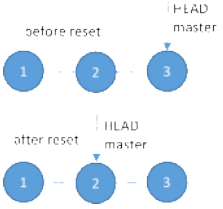
Branch > just pointer to commit (just 40char file with SHA-1), it contains own branch HEAD pointing to the last commitю A branch in Git is simply a lightweight movable pointer to one of commits.
HEAD > special pointer which show where you currently on. During new commits automatically moves. Checkout to branch move HEAD to last commit of the branch. It is possible move HEAD just to any commit
Remote-tracking branch > references to the state of remote branches. They're local references that you can't move; Git moves them for you whenever you do any network communication

Scheme of merge:



Simple three-way merge, using the two snapshots pointed to by the branch tips and the common ancestor of the two. Git determines the best common ancestor to use for its merge base
In case of conflict > fix files > git add > git commit (git status should show that merge in process if there is unsolved conflict)
git mergetool > possible to use some other tool for merging

Reset > copy entries from LR commit to index (in case of files) or set the current branch HEAD to selected pointer & optionally modify WD & Index to match commit



WD	***SA***	***LR***		
3	3	3		
3	3	2	--soft	it is possible to commit > LR become 3; if we try to checkout another branch and checkout back WD = SA = LR = 2
3	2	2	--mixed	default behaviour, LR = SA, nothing to commit if try
2	2	2	--hard	move branch HEAD pointer and align all trees (WD = SA = LR = 2)

Reset + file (to commit 2)

WD	***SA***	***LR***	
3	2	3	branch pointer do not move but SA is updated based on reset commit

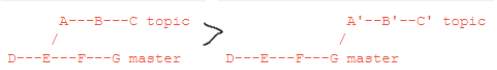
It is possible to use --patch (reset part of commit or even file) during reset
git reset HEAD <file> - to unstage changed file

Checkout > update WD to match index (in case of files), in case of commit align LR = SA = WD but do not move branch HEAD (opposite to reset)

- *git checkout <branch>* > To prepare for working on <branch>, switch to it by updating the index and the files in the working tree, and by pointing H EAD at the branch. (moves HEAD to last commit of the branch, LR contains snaphot of the commit pointed by HEAD, checkout command rewrite trees in a way LR = SA = WD)
- *git checkout <file>* > update WD from Index
- *git checkout <commit> <file>* > update Index from commit & WD from Index
- it is possible to use patch mode like in add or reset

Rebase > Reapply commits of current branch on top of indicated branch (all branches have to exist in local repo (after clone it not exists till checkout))

1) git checkout topic
2) git rebase master (update is related to topic branch)



git rebase --continue or *git rebase --abort* > in case of conflicts resolve & continue or abort

- during rebase it go till find common parent & then rebase all commits, and move current branch
- if have the same changes (the same commit) in history it will ignore them during rebase

Revert > just commit with opposite changes. Use on already shared changes

Stash > for quick changes (in WD or SA) hiding, could be few stashes, they could be applied to another branch

About ^ and ~

```
G   H   I   J
 \ /     \ /
  D   E   F
   \  |  / \
    \ | /   |
     \|/    |
      B     C
       \   /
        \ /
         A

A =      = A^0
B = A^   = A^1     = A~1
C = A^2  = A^2
D = A^^  = A^1^1   = A~2
E = B^2  = A^^2
F = B^3  = A^^3
G = A^^^ = A^1^1^1 = A~3
H = D^2  = B^^2    = A^^^2  = A~2^2
I = F^   = B^3^    = A^^3^
J = F^2  = B^3^2   = A^^3^2
```

^ - first parrent, ^^ parrent of parrent, ^2 second parent it is possible for megred commit
~ - first parrent, ~3 - parrent of parrent of parrent

Push

- push only master branch or indicated branch > git push <repository> <branch>
- to push everything > git push --all / git push origin --all
- it is not pushing tags

Fetch

- Update remotes/origin/branch on local repository
- fetch --all > fetches all branches

Pull

- as well pull only indicated branch, or it is possible to pull --all but it will merge only current branch
- or it fetches all but merge current ?

Garbage collection

if in detached mode add commits they will be collected by GC; it is necessary add branch or tag

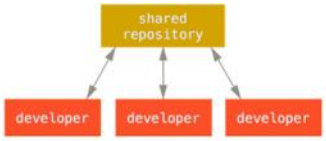
Commit best practices

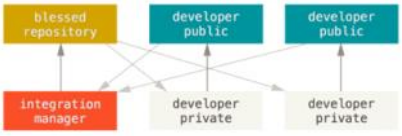
1. Коммиты должны быть атомарны. Добавляйте в ваш коммит изменения близкие по смыслу.
2. Коммиты должны содержать только работающий код.
3. Не коммитьте в нерабочий код (если только ваш код не содержит исправления).
4. Пишите подробные комментарии к коммитам. Они должны описывать проделанные изменения.
5. Не храните в репозитории то, что можно получить из исходных файлов (скомпилированные
6. классы, сгенерированные отчеты
7. Не храните в репозитории конфигурационные файлы, которые зависят от локального окружения или то, что не является неотъемлемой частью проекта (конфигурации IDE, и т.п.).
8. Не добавляйте в репозиторий большие бинарные файлы.
9. Добавляйте, удаляйте, перемещайте или переименовывайте файлы в отдельном коммите.

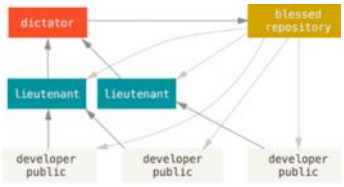
Commit comments

1. Краткий заголовок. Можно указать номер задачи в трекере.
2. После заголовка оставьте одну пустую строку.
3. Перечислите изменения в коде и как они влияют на поведение используя императив (“add test” вместо “I’ve added test” или “adding test”).
4. Укажите вашу мотивацию. Гораздо важнее знать не что было изменено, а почему те или иные изменения были сделаны.

Possible Workflows

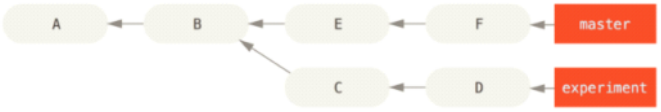






Sample of work

Double Dot Syntax



git log master..experiment > D C (all commits reachable from experiment that aren’t reachable from master)
git log experiment..master > F E
git log origin/master..HEAD > what is going to be pushed to remote
git log refA..refB ~ git log ^refA refB ~ git log refB --not refA => it can be used for more complex queries

git log master...experiment > F E D C - specifies all the commits that are reachable by either of two references but not by both of them

Command Reference

12 февраля 2018 г. 19:04

Part	Command	Param detail	Note
Configuration			Configure git before work
	git config [--global] [--replace-all] <key> <value>	--global	with global for all repositories, without for current only
		--replace-all	if some values already assigned rewrite them
		user.name <Name>	
		user.email <Email>	
		color.ui auto	different colors on command line view
		core.editor <path to editor>	set new default text editor
Examples	git config --global --replace-all core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"		Set as default editor notepad++
	git config --global user.Name "Andrey"		
Clone/Create Repository			How to create or clone repository (normal and bare)
	git init [param]		Initialized empty Git repository, in current directory
		--bare	Initialize empty **bare** repository Bare repository have to be used like remote repository, there is no working directory
	git clone [param] <path to remote repository> [clone to]	clone to	path to which repository will be cloned
		--depth n	works on remotes repositories, clones only last n commits
		--mirror	create bare repository after clone
Add files to staging area			create blobs, add files to stage area
	git add <files>		path to file or whole directory or just . (dot) to add everything under current directory
	git add [param]	--all = -A	add all files from repository which are not ignored
		--update = -u	add only updated tracked files
		--patch = -p	choose what exact changes to commit in the same file
		-e	opens current diff in editor atm use tortoise git for difference comparison
Commit			Save snapshot into repository
	git commit		move to local repository changes and save them, open editor for message
	git commit [param]	-m	indicate commit message in command line
		-a	commits all updated files, does not commit new files
		--amend	used for fixing last commit
		--amend -c HEAD	used for modifying last commit message
Stash			Save current WD and Index
	git stash		Stashes changes (WD & SA) (same as make commit and go back for 1 commit)
	git stash apply [stash no]		apply stash changes (but do not remove stash), could be indicated just stash number
	git stash pop		apply stash changes and remove stash
	git stash list		list available stashes
	git stash save --patch		select parts for stash
	git stash drop <stash name>		delete stash
	git stash clear		remove all saved stashes
	git stash branch [<stash name>]		create a branch from an existed stash
Ignoring files			Files which should not be tracked.
			Just add file pattern (file.x or file.* or *.csv or path\) to .gitignore in any directory, rules from the file applies to subdirectories *.[oa] > Git to ignore any files ending in ".o" or ".a" !lib.a - will be tracked even if *.a ignored
Undo uncommited changes	git reset HEAD -- "file or path"		reset here move file from repository to index unstage the change
File movement/Deleting			The way of moving/deleting files using git
	git mv "File" "to"		after this commit of changes should be done it is possible to move from file system but it looks like delete file and add file **Note:** to rename file it should be moved git mv f_orig f_renamed
	git rm -- "file"		stages file removal
	git rm -r "dir"	-r --cached	-r = recursevely --cached to remove from stage but keep on working tree Move back deleted directory (before commit) > git reset HEAD -- 'dir' -> git checkout -- 'dir'
Branch			
	git branch <new branch name> [starting point]		Create new branch on start point, if ommited created on HEAD
	git checkout -b <new branch> [<starting point>]		Create new branch and checkout to it
	git branch		View all local name of branches
		-a	view all branches (with remotes)
		-r	view remote branches
		--merged	view all branches merged into current
		--no-merged	
		--contains <commit id>	view all branches containing commit id
	git branch -f <branch> <where>		Reset <branchname> to <startpoint>. (Force branch movement)
	git branch -d <local name>		Delete branch locally
	git push origin :the_remote_branch		Delete branch remotely (like pushing blank local branch to origin)
	git branch -u origin/name		Setup branch <name> to track branch origin/name. Before command execution have to be on branch <name>
	git checkout -b sf origin/serverfix		Create and checkout to branch sf and setup for track origin/serverfix remote branch
Tag			
	git tag		list of all tags
	git tag v1.0		create tag at the last commit of current branch
	git tag v1.0 commit_sha1	-d -a	Tag for specific commit Delete tag Annotate tag, used for release, contain creation date, name, message etc; Other tags are lightweight just for temporary use
Merge			Merges two branches into one, create merge commit for this

	1) checkout to branch <name> 2) git merge [param] <branch>		Merge into branch <name>
		--no-commit	Merge changes, but don't commit
		--no-ff	Force the creation of a merge commit (used in case of fasf forward commit to create separate)
		-m	"my message" to replace default merge message
	Merge conflict		In case of merge conflict > fix it > git add <fixed> > git commit <fixed>
			Merge tools: git config --global merge.tool "gvimdiff" (setup for default merge tool) git config --global mergetool.p4merge.cmd\'p4merge.exe \"$BASE\" \"$LOCAL\" \"$REMOTE\" \"$MERGED\"' option tries to revert back to your state before you ran the merge Whitespace related conflicts
		--abort -Xignore-all-space / -Xignore-space-change	
Remotes			Remote repositories
	git remote add <name> <url>		Add a new remote repository, later could commit by name
	git remote rm <name>		Remove a remote
	git remote		list remote repositories
	git remote show		List of remote rrepositories
	git remote show origin		Details about origin remote repository
Push			Push commits to remote repository
	git push		Push the local tracking branch to origin
	git push <remote name> <branch name>		Push changes from a specific branch to a specific remote repository
	git push <remote name> <local branch>:<remote branch>		It is possible to push to some other remote branch
		--force	**! Dangerous**. Force a remote branch to accept a push
	git push origin :beta		Delete the remote branch called beta.
Push Tags			
	git push --tags origin		Push all tags to the origin.
	git push origin v1.0		Push tag v1.0 to the origin
	git fetch --tags origin		Fetch remote tags and update local tags. Note: it will rewrite local tags with the same name
Fetch			Update local repository branches in remotes/origin/branches based on remote repository
	git fetch <remote name>		fetch all branches from remote repository, if repository cloned - automatically adds origin name as name of remote repository
	git fetch origin master:remotes/origin/master		To fetch master from origin to your local copy of the origin/master remote branch
	git fetch --multiple remote1 remote2		Fetch changes from multiple remote repositories
	git fetch --all		Fetch changes from all remote repositories
Pull			Fetch and merge
	git pull [remote repo name] [remote branch]		
	git pull origin <remote branch>:<local branch>		Pull from one branch in another
	git pull --rebase origin master		instead of merge use rebase
Rebase			
	git rebase <branch onto>		Reapply commits on top of another base tip
	git rebase -i <branch onto>		Allows to change sequence, commit message etc. it could be limited if indicate HEAD~n
	git rebase --onto		
Cherry-pick			Given one or more existing commits, apply the change each one introduces, recording a new commit for each.
	git cherry-pick <commit1> <commit2> …		copy commits C1, C2, … to HEAD
Revert			Used to record some new commits to reverse the effect of some earlier commits
	git revert [param] <commit id>		Create revert commit for <commit id>. It is possible to use range of commits
		--no-commit	Revert a commit (changes WD & SA), but don't commit the change (or use -n) if need to revert few commits by only one revert commit
		--continue	Continue the operation in progress using the information in .git/sequencer. Can be used to continue after resolving conflicts in a failed cherry-pick or revert
		--abort	Cancel the operation and return to the pre-sequence state
		--quit	Forget about the current operation in progress. Can be used to clear the sequencer state after a failed cherry-pick or revert
	git revert -m 1 HEAD		For merge commit. The -m 1 flag indicates which parent is the "mainline" and should be kept
Status			
	git status	-s	Could be - untracked files, modified files, deleted etc if git add "a.txt" then modify a.txt > git status show file as staged to commit & modified -s > short status
Log			
	git log [param]		View a reverse chronological list of all commits
		--oneline	View the log with one shortened commit ID and subject
		-N	View the last N commits
		--since="1 week"	View the commits in the last week. You can use many kinds of times with --since or --after
		--author="some user"	View the log entries by a single committer
		--graph	Show branch graph
		--pretty=format	It is possible to define manually log format
	git log --abbrev-commit --pretty=oneline	--abbrev-commit	the output will use shorter values of SHA-1 but keep them unique
RefLog			a log of where your HEAD and branch references have been for the last few months
	git reflog		
Difference			
	git diff		View the differences between the current working tree and the staging area
	git diff --staged		View the differences between the staged changes and repository
	git diff Commit ID		View the differences between the working tree and a commit in the repository
Graphical Interfaces			
	gitk	--all	To view graphical version of history
	git-gui		Graphical version of commit
Cleaning History			
	git filter-branch --tree-filter 'rm -f file' HEAD		Removes specified file from each commit

			It is possible to pass script into ''

Note What is Possible

13 февраля 2018 г. 16:06

Note That Possible

Pathch flag git reset, add, checkout > could have --patch flag
about reset & checkout

Applying **patches from email** > git apply or git am

git **archive** > create and save to zip just some repository snapshot

git **grep** > allows you to easily search through any committed tree or the working directory for a string or regular expression

git log **-S** ZLIB_BUF_MAX --oneline > find out when the ZLIB_BUF_MAX constant was originally introduced

git log **-L** :git_deflate_bound:zlib.c > it will show you the history of a function or line of code in your codebase

To modify commits far back need try to use rebase interactively, during rebase it is possible to modify commits. As well during interactive rebase it is possible to squash or split commits

filter-branch > need to rewrite a larger number of commits in some scriptable way – for instance, changing your email address globally or removing a file from every commit
git filter-branch --tree-filter 'rm -f passwords.txt' HEAD > removes passwords.txt from each commit
It also could use scripts inside '...'; --subdirectory-filter; --commit-filter

Ours/theirs preferences > use during the merge to define in case of conflict which file to prefer
git merge -Xours <branch>

Rerere > reuse recorded resolution > it allows you to ask Git to remember how you've resolved a hunk conflict so that the next time it sees the same conflict, Git can resolve it for you automatically.

git blame > It shows you what commit was the last to modify each line of any file.

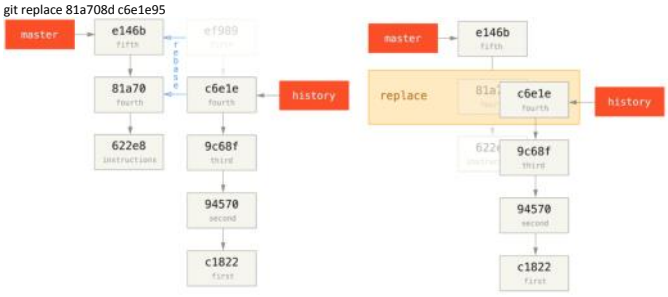
Binary search > The **bisect** command does a binary search through your commit history to help you identify as quickly as possible which commit introduced an issue.

- First you run git bisect start to get things going
- Then you use git bisect bad to tell the system that the current commit you're on is broken
- Then, you must tell bisect when the last known good state was, using git bisect good <good_commit>
- When you're finished, you should run git bisect reset

Submodules allow you to keep a Git repository as a subdirectory of another Git repository. This lets you clone another repository into your project and keep your commits separate.

Bundling > Git is capable of "bundling" its data into a single file. This data could be transferred and restored latter.
git bundle create repo.bundle HEAD master > Now you have a file named repo.bundle that has all the data needed to re-create the repository's master branch.
git clone repo.bundle to use this repository

The replace command lets you specify an object in Git and say "every time you refer to this object, pretend it's a different object". This is most commonly useful for replacing one commit in your history with another one without having to rebuild the entire history with, say, git filter-branch.
git replace 81a708d c6e1e95



Credential storage

- The default is not to cache at all. Every connection will prompt you for your username and password.
- The "cache" mode keeps credentials in memory for a certain period of time. None of the passwords are ever stored on disk, and they are purged from the cache after 15 minutes.
- The "store" mode saves the credentials to a plain-text file on disk, and they never expire. This means that until you change your password for the Git host, you won't ever have to type in your credentials again. The downside of this approach is that your passwords are stored in cleartext in a plain file in your home directory.
- If you're using a Mac, Git comes with an "osxkeychain" mode, which caches credentials in the secure keychain that's attached to your system account. This method stores the credentials on disk, and they never expire, but they're encrypted with the same system that stores HTTPS certificates and Safari auto-fills.
- If you're using Windows, you can install a helper called "Git Credential Manager for Windows." This is similar to the "osxkeychain" helper described above, but uses the Windows Credential Store to control sensitive information. It can be found at https://github.com/Microsoft/Git-Credential-Manager-for-Windows.

Signing Tags and Commits
gpg --gen-key - key generation
git config --global user.signingkey 0A46826A - config key for signing
git tag -s v1.5 - signing tag
git tag -v <tag-name> - verifying a signed tag

Git as Client > it is possible to use git as client during using another version control system. Normally exists 'bridges' which aligns systems work.
About migration to git from anther VCS.

Git configuration

comit.template > Git will use that file as the default initial message when you commit
core.autocrlf > line ending issues (converts LF endings into CRLF)

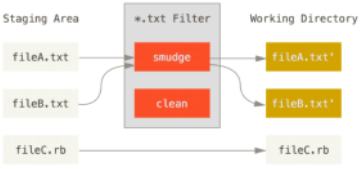
External Merge and Diff Tools
P4Merge setup in .gitconfig

```
[merge]
  tool = extMerge
[mergetool "extMerge"]
  cmd = extMerge "$BASE" "$LOCAL" "$REMOTE" "$MERGED"
  trustExitCode = false
[diff]
  external = extDiff
```

Path-specific settings are called **Git attributes** and are set either in a .gitattributes file in one of your directories (normally the root of your project) or in the .git/info/attributes file if you don't want the attributes file committed with your project.
*.pbxproj binary > To tell Git to treat all pbxproj files as binary data, add the following line to your .gitattributes file

It is possible to indicate **how to diff binary** files e.g. ***.docx diff=word** (requires addition tools setup)

You can **inject** text into a file when it's checked out and **remove** it again before it's added to a commit.



Git Hooks > custom scripts when certain important actions occur. (.git/hooks, client side and server side)

Git internals
Plumbing command > low level work
Porcelain command > more user-friendly

At the core of Git is a simple key-value data store (c content-addressable filesystem). Blob (file), tree (points to trees or files), commit (point to trees + meta information), tag objects (points to commits). Object storage…

The initial format in which Git saves objects on disk is called a "loose" object format. However, occasionally Git packs up several of these objects into a single binary file called a "packfile" in order to save space and be more efficient. (if too many objects or git gc or git push)

Refspec
git remote add origin <repo>. It creates file .git/config >
url = https://github.com/schacon/simplegit-progit
fetch = +refs/heads/*:refs/remotes/origin/*
<src>:<dst>, where <src> is the pattern for references on the remote side and <dst> is where those references will be tracked locally. The + tells Git to update the reference even if it isn't a fast-forward.
Similary push = refs/heads/master:refs/heads/qa/master

Garbage Collection
git gc --auto > do nothing, only after 7,000 loose objects or more than 50 packfiles for Git to fire up a real gc command

Data recovery >
git **reflog** (Git silently records what your HEAD is every time you change it.)
git **fsck --full** > checks your database for integrity, it shows you all objects that aren't pointed to by another object.

git filter-branch > to rewrite history

Git environment variables

Git on server protocols

- Local (filesystem)
- HTTP (just login/password)
- Secure Shell (SSH) (require ssh key)
- Git

Scenarios

13 февраля 2018 г. 16:11

Add all files to new repository

- go to directory with files
- git init
- git add .
- git commit -m "Initial commit"

Project start. Work with new github repository

- Create repository on github
- Go to blank project folder
- git clone "github repository"

Remote repository already exist need to start work with it

- Select required folder
- git clone "path to remote repository"
  or
- git clone "path to remote repository" "local folder"

Create remote repository locally

- Select required folder
- git init --bare

Local Repository exists need to add it to new github repository

- Create blank repository on github
- git remote add <remote name> <remote path>
- git push <remote name> master

Local files exists need to add them to new github repository

- Go to project folder
- git init
- git add .
- git commit -m "Initial commit"
- git remote add origin <path to remote>
- git push origin master
- git branch -u origin/master master

? Diff & Merge & Conflict resolution graphic tool

Create custom git command

Make *git status* to call with *git st*:

1. Open 'C:\Program Files\Git\mingw64\libexec\git-core\'
2. Create blank file without extension git-st (git-'command name')
3. First line of file #!/bin/sh
4. Script body > git status

File renaming: 'git mv <source file> <destination file>' > git mv $1 $2