

# Git

Arayeh Afshari

# Git

is a version control system (vcs)

version control is a system that records changes to a file or set of file over time so that you can recall specific versions later

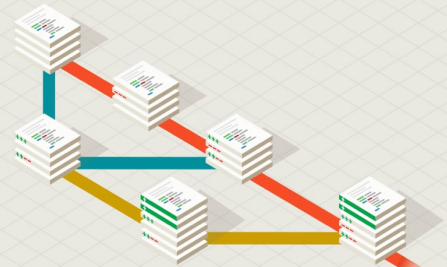
the first step in CI is have a version control system

## Central Version Control Systems

## Distributed Version Control Systems

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.



<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

## Git installation

yum install [http://opensource.wandisco.com/centos/7/git/x86\\_64/wandisco-git-release-7-2.noarch.rpm](http://opensource.wandisco.com/centos/7/git/x86_64/wandisco-git-release-7-2.noarch.rpm)

or

yum install git

yum git version

بر اساس توزیع لینوکس خود فایل مناسب را نصب کنید

## Initial Git Setup

در قدم اول باید خود را به گیت معرفی کنیم تا بفهمد ما چه کسی هستیم

git config —global user.name Arye

git config —global user.email A@H

git config —list

above information saved in user's home directory(.gitconfig)

git config --global color.ui auto

—system در سطح سیستم  
git config —global در سطح یوزر  
—local در سطح دایرکتوری

# Create Repo

working Directory

git init mypro

.git folder created

Bare Repository

git init -bare project1

don't have .git folder

(used for sharing with others)

the setting of the repository (url of the remote, username, email)

```
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
```

```
+ GitJenkins git init mypro
Initialized empty Git repository in /home/ArYe/GitJenkins/mypro/.git/
+ GitJenkins tree mypro/.git
mypro/.git
├── branches
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
├── refs
│   ├── heads
│   └── tags
└── 9 directories, 13 files
```

it is a pointer to current location (refs/heads/master)

there are script files that run when we want

contain files you don't want git to deal.

unlike the .gitignore file, it won't be shared

```
+ objects git:(master) tree .
5e
├── 82eeff6f9586f7b16a07b9827cfa0ebca423f0
├── c31c12802b79dece18caf85f3779ca180c188
├── 84
├── 5d9876b6f1d2f87c639a719848888ff5d7258
├── 8b
├── 137891791fe96927ad78e64b0aad7bdeed08bdc
├── b1
├── a82986b7841b8bdb6052fddcd60b0a0ae9bc3a
├── fd
├── eb2a216d0b4739eda4857464512a680512f980
├── info
└── pack
```

در صورت commit کردن اطلاعات به صورت object در این مسیر ذخیره می شود این مدل ذخیره سازی برای سرچ سریع تر گیت ایجاد شده است (در صورت تغییرات و commit کردن سه ویرون ایجاد تغییرات commit کردن دو object ایجاد می گردد)

git cat-file -p 5e82eeff6f9586f7b16a07b9827cfa0ebca423f0

به مدل ذخیره سازی توجه کنید دو حرف اول به عنوان فولدر و بقیه آن به صورت فایل در زیر فولدر ذخیره می شود و برای نمایش محتوا از دستور فوق استفاده می کنیم

## Sample Command:

git status

git add ( filename or . )

git commit -m "message"

git commit

git config —global core.editor(change default editor)

آیا یک فایل می تواند در دو state قرار بگیرد؟؟

اگر فایلی را ادیت کنیم stage کنیم (git add) و قبل از commit دوباره آن را ادیت کنیم.  
بنابراین قبل از commit بایستی آن را به گیت add کرد

```
* myrepo git:(master) x git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file2
nothing added to commit but untracked files present (use "git add" to track)
→ myrepo git:(master) x git add .
→ myrepo git:(master) x vim file2
→ myrepo git:(master) x git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   file2
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   file2
#
→ myrepo git:(master) x git add .
→ myrepo git:(master) x git commit -m "edit file 2"
[master 5d1ld0c] edit file 2
1 file changed, 1 insertion(+)
create mode 100644 file2
```

چطور خطوط مشخصی از فایل را در داخل commit بیاوریم؟

git add (interactively choose hunks)

git add file2 —patch

در اینجا مشکل بالا هم به وجود می آید و بایستی قبل از commit کردن add file را اجرا کرد

?

e

```
# ignore all .a files
*.a
# but do track lib.a, even though you're ignore .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TOD
/TOD
# ignore all files in any directory named build
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

فایل gitignore.

(نیاز به add شدن و commit شدن دارد)

همه تیم می توانند آن را ببینند و در صورت clone کردن و push کردن برای همه اعمال می شود

اگر developer نخواهد gitignore در مجرای خود داشته باشد بایستی آن را در info/exclude قرار دهد

برای فایل های modify شده از دستور زیر استفاده می کنیم تا در مرحله add و commit با هم انجام شود

**git commit -am "fixbug124"**

اگر فایل test.a را یکبار add و commit کرده باشیم و سپس آن را در داخل gitignore قرار دهیم آن فایل نادیده گرفته نمی شود چون گیت به آن چسبیده و تغییرات آن را دنبال می کنی برای اینکه فایل نادیده گرفته شود بایستی آن untrace شود

**git rm —cached filename**

با دستور مقابل می توان فایل را untracked کرد

اگر فایلی که در داخل ورژن های قدیمی commit شده باشد و بخواهید آن را در ورژن های پیشین ignore کنید بهتر است این کار را نکنید چون خلاف ذات git است در صورت انجام این کار بهتر است حتما از repository بکاپ داشته باشید

**git rm file**

برای حذف فایل و تغییر نام از دستورات مقابل استفاده می کنیم (استفاد از فرامین لینوکس نیازمند یک مرحله add کردن می باشد)

**git mv file2 f2**

و بهتر است از دستورات خود git استفاده شود)

**git log -3**

سه لاگ آخر را نشان می دهد

**git log -p -2**

دو لاگ آخر را با جزئیات نشان می دهد

**git diff file1**

اگر بخواهیم فایل modified شده را با آخرین commit خود مقایسه کنیم از git diff استفاده می کنیم  
اما اگر فایل stage شده باشد از git diff — stage استفاده می کنیم

**git diff — stage file1**

**git commit —amend**

اگر یک commit انجام داده باشیم و بینیم تغییری را فراموش کرده باشیم که به این commit اعمال کنیم یا حتی اسم commit را اشتباه زده باشیم از دستور مقابل استفاده می کنیم توجه داشته باشید commitid تغییر خواهد کرد  
واضح است که اگر تغییری در فایل انجام داده باشیم بایستی آن را در stage کنیم (git add)

**git reset HEAD file1**

برای unstage کردن فایل از دستور مقابل استفاده می کنیم

**git checkout file1**

برای unmodified کردن یک فایل و برگشت به آخرین commit آن از دستور مقابل استفاده می کنیم

# Remote Repository

```
git clone http://.../test
/srv/git/test
cd test
git remote -v
```

اگر دایرکتوری پروژه خود را از هر مسیری جز **local** خود بیاوریم به آن **Remote Repository** گوئیم  
و گیت برای ساده بودن آنرا **origin** می نامد

با دستور **git remote -v** بررسی می کنیم **remote repository** های ما چقدر تا هست **origin** از کجا آمده است  
این اطلاعات در فایل **.git/config**

آیا می توانیم چنین **remote repository** داشته باشیم و به آنها وصل شویم و چه کاربردی دارد؟؟  
بله

```
git remote add remote2 http://git.ArYe.com/git/myrepo2
git remote rename remote2 remotel
vim .git/config
git remote remove remotel
```

تا الان **command** هایی را زدیم که **local** ای بوده اند و برای اینکه تغییرات **local** خود را بر روی سرور یا **remote repository** قرار دهیم بایستی از دستور **push** استفاده کنیم

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

اطلاعات **local** ما روی چه **remote** و چه شاخه ای قرار گیرد. به صورت **default** مقادیر **origin** و **master** برای آن **set** شده است  
اسم **default branch** ما **master** است و شاخه اصلی پروژه ما است

```
git push
```

```
git fetch
git merge
```

```
git pull
```

برای دریافت اطلاعات از **remote repository** دو راه وجود دارد **fetch** یا **pull**  
در حالت **fetch** اطلاعات در **working directory** ما دانلود می شود اما نشان داده نمی شود و می توان با **diff** مقایسه انجام داد و بعد از اعتبار سنجی تغییرات آن را به پروژه با دستور **merge** اضافه کرد  
اگر نخواهیم مستقیم اطلاعات **update** شوند از دستور **pull** استفاده می کنیم

**git fetch + git merge = git pull**

before git push

همیشه قبل از **git push** یک **git pull** بزنید تا به اروز نخوریم  
**git pull --> git push**

ممکن است **Developer** دیگری  
تغییرات را در **Remote Repository** انجام  
داده باشد و ما از آن بی خبر باشیم

از **pull** **git** استفاده می کنیم  
تا از آن تغییرات با خبر شویم

حال **git push** می کنیم

git remote show origin

ما را به Remote Repository متصل می کند و اطلاعاتی را به ما می دهد که نشان می دهد آیا اطلاعات ما با Bare Repository روی سرور گیت up to date هست یا نه

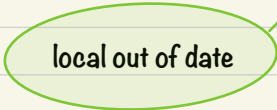
اگر ما از اطلاعات روی سرور عقب تر باشیم

به ما local out of date نشان داده می شود و برای sync کردن بایستی از git pull استفاده کنیم

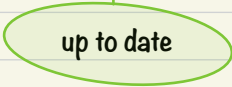
اگر ما از اطلاعات روی سرور جلو تر باشیم

به ما fast-forwardable نشان داده می شود و برای sync کردن بایستی از git push استفاده کنیم

git remote show origin



git pull



git push

Tagging

وقتی commit ای ثبت می کنیم یک کاتر اکتر آیزی ؟؟ تالی برای ما ثبت می شود معمولا برای آنکه از آن commit نسخه ای reales کنیم بر روی نسخه ای که می خواهیم release کنیم tag می زنیم

چون commit برای ما مهم است و میخواهیم بعد از آن یک reales برهیم تا بتوانیم سازه تر یا آنی صحیت کنیم

در این مدل ذخیره می شوند .git/refs/tags

tag

lightweight tags

git tag v2.3

بر روی آکفون commit اسال می شود

git tag v2.3 CommitID

Annotated tags

more meta data (name email date ...)

git tag -a tagname -m "tagmessage"

git tag

همه tag ها را نشان می دهد

git show v2.3

اطلاعات مربوط به این tag را نشان می دهد

git log —pretty=oneline

git log —oneline

`git push origin <tag_name>`

تا به حال tag ها روی سیستم local ایجاد شده بود و با `git push` خالی بر روی remote آپلود نمی شوند و بایستی نام tag آن را بنویسیم

`git push origin --tags`

همه tag های local را روی remote آپلود می کند

git remote show origin  
اطلاعات tag را برمی خیزد

## How to delete a tag?

اول از روی سیستم local حذف و سپس به remote آن را push می کنیم

`git tag -d v2.3`

deleted from working directory

`git push origin :refs/tags/v2.3`

deleted from remote repo

سمت چپ: خالی است

`git pull --prune origin refs/tags/*:refs/tags/*`

اگر روی remote حذف شده باشد و روی local ما باشد برای حذف آن به این صورت عمل می کنیم

`git ls remote --tag origin (or url)`

تگ های روی remote repo را به ما نشان می دهد

Brife

## How to delete a tag?

delete from working directory

delete from remote repo

`git tag -d v2.3`

`git push origin :refs/tags/v2.3`

## How to sync deleted tags from remote repo to my working directory?

`git pull --prune origin refs/tags/*:refs/tags/*`



# Branching

تا اینجا بدون آگاهی از branch هر تغییری اعمال و commit می کردیم در branch master اعمال می شد  
معمولا ما در branch master عملیات commit انجام نمی دهیم و در branch develop کار توسعه و develop پروژه را انجام می دهیم و در آن می کنیم

## git branch testing



## git branch

لیست branch ها در local را نشان می دهد

## git branch -a

لیست همه branch ها (remote) را نشان می دهد

git checkout filename unmodify changes(return to last commit)

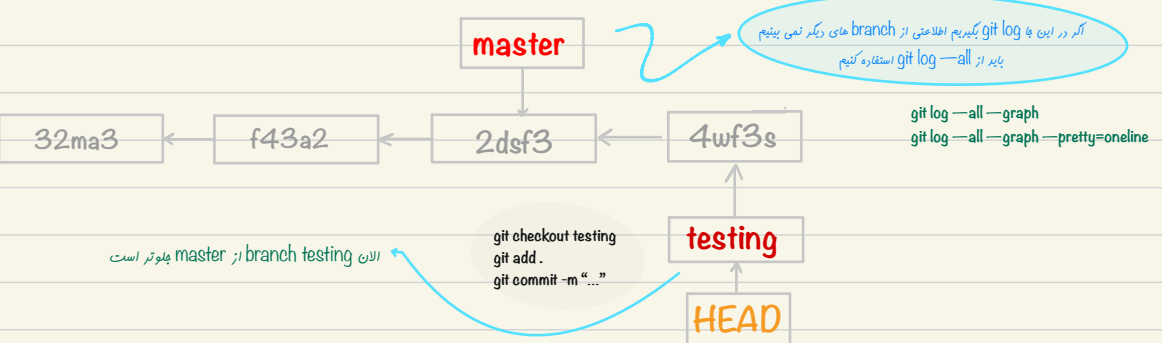
git checkout branchname branch change

## git branch -v

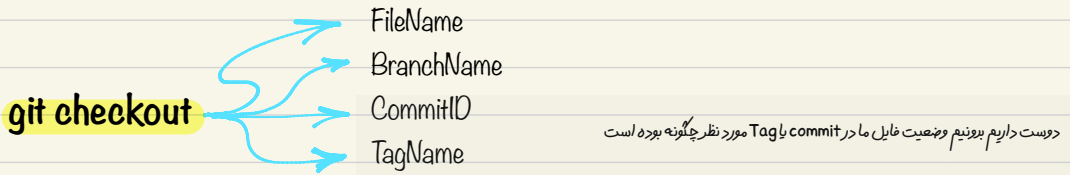
```
*master 2dsf3fa [gone] add f2
testing 2dsf3fa add f2
```

commit ID

commit comand



## Detached HEAD state



مونتن در detached HEAD خوب نیست !!

برای حل این موضوع بایستی یک branch جدید ایجاد کنیم تا بتوانیم تغییرات را داشته باشیم

به عبارت دیگر نباید به یک commitid رفته و از آنجا فایل ها را تغییر دهیم و commit کنیم

چون فایل ها گم می شود !!

`git checkout -b NewbranchName`   or   `git branch BNew`  
`git checkout Bnew`

## Merging

زمانی که کار ما در branch تمام شد بایستی آن را با branch مورد نظر merge کنیم





## Conflict Merging

If you changed the same part of the same file differently in the two branches you're merging, git won't be able to merge them cleanly and a conflict will happen.

1  
#master  
vim fl  
git commit -am "edit fl on master branch"

2  
git checkout B1  
vim fl  
git commit -am "edit fl on B1 branch"

3  
git checkout master  
git merge B1

```
Arye@local:git merge B1
Auto-merging f1
CONFLICT (content) : merge conflict in f1
Automatic merge failed; fix conflicts then commit the result
```

4  
vim fl

```
<<<<<<HEAD
add this code line in Master branch
=====
add this code line on B1 branch
>>>>>>B1
```

فقط اول و آخر پاک می کنیم  
و کد بالا و پایین مساوی را بررسی و یکی را انتخاب می کنیم

4  
git status

```
You have unmerged path
(fix conflict and run "git commit")
(use "git merge --abort" to abort the merge)

unmerged paths:
(use "git add <file>..." to mark resolution)
    both modified: f1
```

می توان با اجرای این دستور بی خیال تغییرات و merge شد

5  
git add .  
git commit -m "solve conflict"

## More Command

git checkout master  
git branch —merged

B1  
\*master

از زاویه دید master می توان branch هایی که merge شدن را دید

git branch —no-merged

B2

از زاویه دید master می توان branch هایی که merge نشدن را دید

git branch (-d or -D) B2

با این دستور branch B2 را حذف می کنیم

دستورات. بالا روی سیستم local انجام می شد برای حالت remote داستان چگونه؟؟

git push origin B5(BranchName)

—>>> push the branch to the remote repository

git push origin :B5 or git push origin —delete B5 —>>> delete the branch from the remote repository

git remote show origin —>> show information branches and synchronization to local

توجه: اگر از Remote Repo ما clone کنیم تمام branchها هم دانلود می شوند اما برای نمایش باید **git branch -a** زده شود

اگر branch روی remote حذف باشد چگونه آن را از روی local حذف کنیم؟؟

we have master & B4 branches on remote repo

#Developer1



git checkout master

git branch -D B4

git push origin :B4

از روی سیستم local نسخه دهته اول و remote اطلاعات branch B4 را حذف کردیم

#Developer2



git remote show origin —>

git remote prune origin

git branch -vv —>

```
.....
remote branches:
  master      tracked
  refs/remotes/origin/B4  stale (use 'git remote prune' to remove)
local branches:
  B4          merge with remote B4
  master      merge with remote master
```

```
B4 e2ret33 [origin/B4: gone] add filex
*master r32ew3e [origin/master] solve conflict
```

چون دستور git remote prune origin اجرا شد  
در branch B4 remote پیدا نشد گone خورده شد

الان نسخه دهته دوم نی داریم روی remote repo  
چه branchهایی وجود دارد یا ندارد  
و میخوانیم branchهای اضافی را از روی سیستم خود حذف کنیم

git branch -vv | grep 'origin/.\*: gone' | awk '{print \$1}' | xargs git branch -D

ما یک branch روی local داریم اما روی remote repo نیست  
این ما میخوانیم detect کنیم  
و branchهای اضافی را در remote local مان حذف نمایم

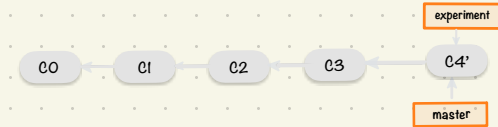
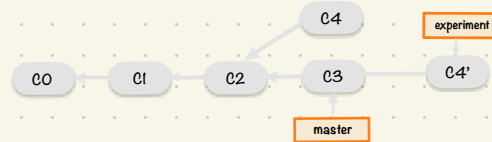
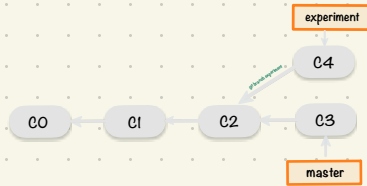
ما با انجام config زیر دیگر نیازی نداریم که git remote prune origin انجام دهیم و با هر بار که ما یک git push or fetch خود به صورت اتومات prune هم انجام می دهد و branchهای اضافی ما lable مربوط به gone را می خورند و مشخص می شوند

git config fetch.prune true

# Rebasing

Take all the changes that were committed on one branch and replay them on a different branch

روش دیگر ادغام اطلاعات branch هاست که با diff گرفتن از branch آن را به branch اصلی یعنی master متصل می کند و یک تاریخچه خطی به ما میدهد



1 git checkout experiment  
git rebase master

2 git checkout master  
git merge experiment

3 git branch -d experiment  
git log --all --graph --online

برخلاف حالت merge کردن اول روی شاخه ای که می خواهیم merge کنیم می رویم سپس روی شاخه اصلی rebase می کنیم

اگر فایل مشترکی را در اینجا edit کرده باشیم با مشکل conflict مواجه می شویم

```
Arye@local:git status
interactive rebase in progress; onto 5ce23b6
....
You are currently rebasing branch 'master' on '5ce23b6'
(fix conflicts and then run "git rebase --continue")
(use "git rebase --skip" to skip this patch)
(use "git rebase --abort" to check out the origin branch)
....
both modified :f1
```

vim f1 --> select and edit code  
git add .  
git rebase --continue

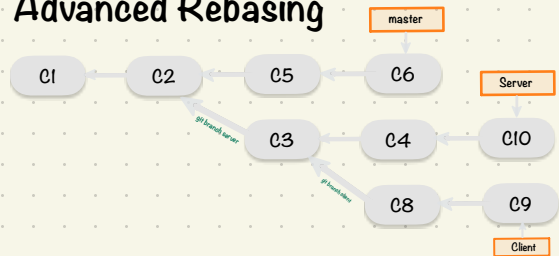
```
<<<<<<HEAD
add this code line in Master branch
=====
add this code line on experiment branch
>>>>>>experiment
```

# Advanced Rebasing

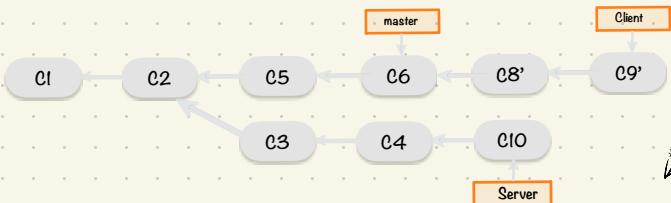
اگر بخواهیم اطلاعات یک branch را برداریم به شاخه اصلی اضافه کنیم چگونه عمل می کنیم؟؟

در اینجا یک branch سرور ایجاد کردیم و روی آن یک branch کلاینت.

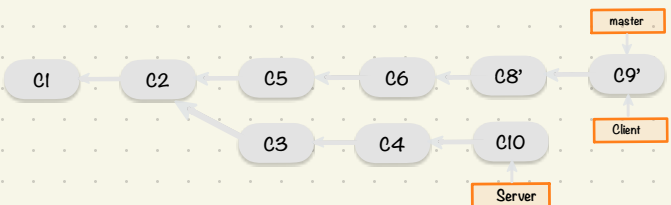
کار branch client زودتر تمام می شود چگونه اول branch client را به شاخه اصلی merge کنیم؟؟



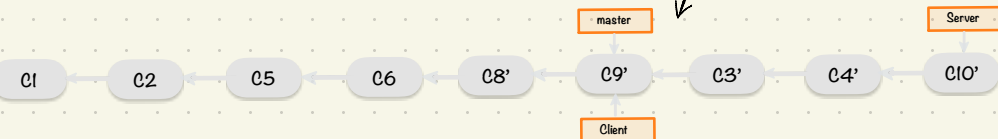
تفسیرات را ببینید <— git log —all —graph —online



**git checkout master**  
**git merge client**



**git rebase master server**  
**git checkout master**  
**git merge server**



**git branch -d server**

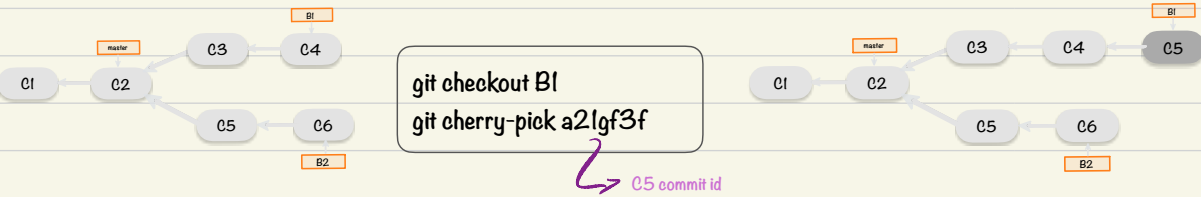
**git branch -d client**



# Cherry pick

cherry picking is the act picking a commit from a branch and applying it to another

اگر commit یی را اشتباه داخل یک branch دیگر انجام دهیم و در ادامه آن چند commit را اشتباه پشت آن انجام دهیم حال بایستی این اطلاعات را برداشته و در branch درست جایگذاری کنیم این کار چگونه انجام می شود؟؟



# Stashing

اگر ما در حال فعالیت در داخل branch خود باشیم و یک موضوع critical به وجود بیاید و بایستی آن کار را انجام دهیم فایل های خود که هنوز در حال modify یا stage هستند را نیمه کاره رها کنیم و به branch دیگر برویم فایل های خود را چه کنیم؟؟ (از commit نمی خواهیم استفاده کنیم)

```
graph LR
    C0 --> C1
    C1 --> C2
    C1 --> C3
    C2 --> C4
    C3 --> C5
    C4 --> C5
    C5 --> C6
    C6 --> C7
    C7 --> C8
    C8 --> C9
    C9 --> C10
    C10 --> C11
    C11 --> C12
    C12 --> C13
    C13 --> C14
    C14 --> C15
    C15 --> C16
    C16 --> C17
    C17 --> C18
    C18 --> C19
    C19 --> C20
    C20 --> C21
    C21 --> C22
    C22 --> C23
    C23 --> C24
    C24 --> C25
    C25 --> C26
    C26 --> C27
    C27 --> C28
    C28 --> C29
    C29 --> C30
    C30 --> C31
    C31 --> C32
    C32 --> C33
    C33 --> C34
    C34 --> C35
    C35 --> C36
    C36 --> C37
    C37 --> C38
    C38 --> C39
    C39 --> C40
    C40 --> C41
    C41 --> C42
    C42 --> C43
    C43 --> C44
    C44 --> C45
    C45 --> C46
    C46 --> C47
    C47 --> C48
    C48 --> C49
    C49 --> C50
    C50 --> C51
    C51 --> C52
    C52 --> C53
    C53 --> C54
    C54 --> C55
    C55 --> C56
    C56 --> C57
    C57 --> C58
    C58 --> C59
    C59 --> C60
    C60 --> C61
    C61 --> C62
    C62 --> C63
    C63 --> C64
    C64 --> C65
    C65 --> C66
    C66 --> C67
    C67 --> C68
    C68 --> C69
    C69 --> C70
    C70 --> C71
    C71 --> C72
    C72 --> C73
    C73 --> C74
    C74 --> C75
    C75 --> C76
    C76 --> C77
    C77 --> C78
    C78 --> C79
    C79 --> C80
    C80 --> C81
    C81 --> C82
    C82 --> C83
    C83 --> C84
    C84 --> C85
    C85 --> C86
    C86 --> C87
    C87 --> C88
    C88 --> C89
    C89 --> C90
    C90 --> C91
    C91 --> C92
    C92 --> C93
    C93 --> C94
    C94 --> C95
    C95 --> C96
    C96 --> C97
    C97 --> C98
    C98 --> C99
    C99 --> C100
```

**git stash**

بالین دستور اطلاعات در stash ذخیره می شود

```
Arye@local:git log --all --graph --online
* 05e3w5r (HEAD -> B1) add f3
* 3er54ch add f2
* 43fd32s (master) add f1
* a4b37m1 add f0
```

```
Arye@local:git status
on branch B1
change to be committed:
  modified: f1
change not staged for commit:
  modified: f2
```

```
Arye@local:git checkout master
error: your local changes to the following
files would be overwritten by checkout:f2
please commit your change or stash them
before you switch branches Aborting
```

```
Arye@local:git stash
saved working directory and index state WIP on B1:
05e3w5r add f3
```

```
Arye@local:git status
on branch B1
nothing to commit,working tree clean
```

```
Arye@local:git checkout master
switched to brach 'master'
```

```
Arye@local:git stash list
stash@{0}: WIP on B1: 05e3w5r add f3

Arye@local:git checkout B1
switched to branch 'B1'

Arye@localhost:git stash apply stash{0}
Changes not staged for commit:
...
  modified: f1
  modified: f2
```

دو فایل به حالت modified شده برویگردد برای برگشت دقیق سورسج های آن را باید استفاده کرد

وقتی از دستور `stash apply stash{0}` استفاده می کنیم اطلاعات در stash باقی می ماند و باید دستور `stash drop stash{0}` را اجرا کنیم

می توان از `git stash pop` استفاده کرد که اطلاعات مستقیم از `stash list` حذف شود

چون وضعیت فایل ها مشخص نشده به ما اجازه تغییر branch دارد  
نیی شود که با `git stash` وضعیت فعلی ذخیره می شود (مشابه stack)

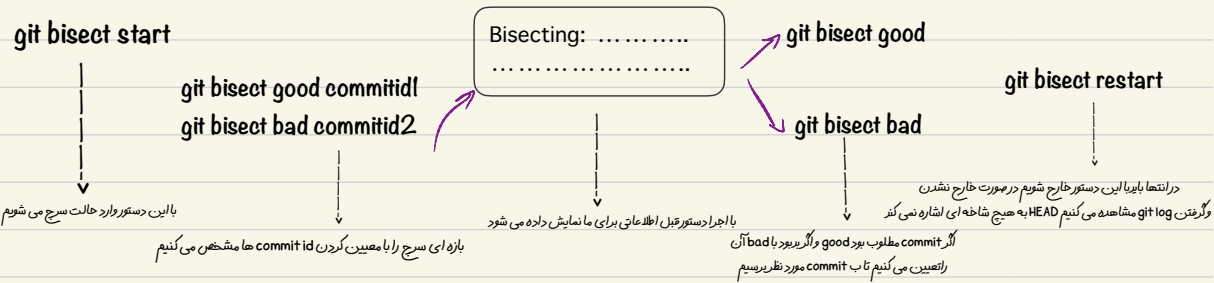
اگر اطلاعات stash را در branch دیگری `unstash` کنیم ممکن است به `conflict` برخورد کنیم که مانند گذشته به حل آن می پردازیم

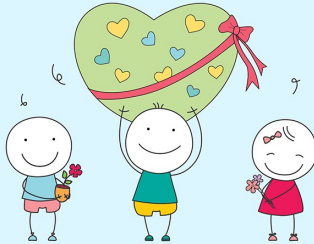



# Git bisect

اگر بخواهیم داخل commit های گذشته سرچ کنیم

ما به دنبال یک پارامتر می گردیم که بفهمیم این پارامتر در کدام commit تغییر یافته است روش چیست؟؟



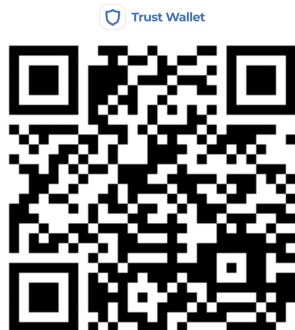




موسسه خیریه حمایت از  
کودکان مبتلا به سرطان  
**محک**

شماره کارت  
۵۸۹۲-۱۰۷-۴۴۰۷-۵۰۰۳

شماره حساب  
۱۲۰۴-۸۰۱-۵۵۵۵-۴



Trust Wallet

BTC

bc1q82uvvgmccs2c6xzc2ls47jwrnaewnmrd2a5nng

bc1q82uvvgmccs2c6xzc2ls47jwrnaewnmrd2a5nng

۶۲۱۹-۸۶۱۹-۱۰۵۶-۵۱۶۶

حمایت از من

# Git over http

چون گیت خود مکانیزمی برای authentication ندارد آن را پشت یک apache بالا میاریم و اینجا آگاهی یوزر و پسورد و دسترسی به فایل ها را برای ما برمی میکند

1 sudo vim /etc/httpd/conf.d/git.conf

```
<VirtualHost *:80>
```

```
ServerAdmin arye@aryenet.net
```

```
ServerName git.aryenet.net
```

```
SetEnv GIT_PROJECT_ROOT /srv/git
```

```
SetEnv GIT_HTTP_EXPORT_ALL
```

```
ScriptAlias /git/ /usr/libexec/git-core/git-http-backend/
```

```
<LocationMatch "^/git.*$">
```

```
Require all denied
```

```
AuthType Basic
```

```
AuthName "Restricted Content"
```

```
AuthUserFile /opt/pass
```

```
Require valid-user
```

```
</LocationMatch>
```

```
RewriteCond %{REQUEST_URI} /git-receive-pack$
```

```
RewriteRule ^/git - [E=AUTHREQUIRED:yes]
```

```
</VirtualHost>
```

همه repository های گیت در این مسیر قرار دارند

First it is necessary to understand that there are 2 components to git-over-http: git and apache. These two are connected through a script with the name of git-http-backend

This command applies to all repositories

برای اینکه برای هر repository جداگانه یوزر و دسترسی تعیین کنیم باید tag های LocationMatch جدا تعریف کنیم

If we want to define separate access for each repository,

we must define similar blocks

```
<LocationMatch "^/git/rep1/.*$">
```

```
.....
```

```
</LocationMatch>
```

چون filebase authentication هستیم

فقط افراد valid-user به مسیر root پروژه دسترسی دارند که اطلاعات آنها در /opt/pass باشد

2 sudo systemctl start https

```
cd /opt
```

file name

```
sudo htpasswd -c pass Arye
```

```
sudo htpasswd pass user1
```

برای اضافه کردن یوزر به فایل

```
cat /opt/pass
```

```
Arye.$apr1$2KSZ1rj8$KcSbrh5hwXZnpXe0ij9p8/
```

```
user1.$apr1$hmMivmlw$g$FmV3uv5.ZaPl.VfFNIs0
```

```
mkdir srv/git
```

```
Chown -R apache:apache /srv/git
```

## Create Bare Repository

```
git init --bare myrepo
```

```
cd myrepo
```

```
sudo cp hooks/post-update.sample /srv/git/myrepo/hooks/post-update
```

```
sudo git config http.receivepack true
```

چون گیت پشت apache قرار گرفته و سرورس میله اگر این در config را داخل repository که ساختیم نزدیک این repo روی پروتکل http نمی توانرند receive و send انجام دهیم

```
sudo git config http.uploadpack true
```

```
sudo git update-server-info
```

```
sudo chown -R apache:apache /srv/git (Redhat)
```

```
sudo chown -R www-data:www-data /srv/git (deb)
```

redhat base  
package https  
user apache

debian base  
package apache2  
user www-data

### Run commands Git over http(page 5)

```
git clone <url>
```

```
git clone http://127.0.0.1/git/myrepo
```