در گیت هاب ما با ساخت یک اکانت عمومی مرحله تست رو انجام میدیم

بعد از تاییدیه ثبت نام و مبادله کد تایید ما رسما عضو گیت هاب میشیم

برای مخزن از ما یه اسم میخواد

و ....

\*

گیت یک برنامه مدیریت ورژنه که در کارگروهها خیلی کاربرد داره، هر فرد میتونه مطمئن باشه که کارش دیده میشه کم کاری دیگران دیده میشه و بدون نگرانی از حقوق مادی و معنویش میتونه تلاش شبانه روزی کنه و یک رقابت سالم برقرار میشه

اشکالش اینه که کاری که تو یه کامپیوتر در محیط گیت دنبال میشه در روی فولدری که مخزن شده مدیریت میشه و وقتی سیستم رو عوض کنید دیگه داده‌های قبلی رو از دست دادید

؟؟ امید میره این مشکل در گیت لب و گیت هاب رفع شده باشه؟؟ یا با استفاده از فلش مموری این کار انجام شود اما ممکن است نام درایو فلش که روی هر سیستم تغییر میکند باعث مشکل کند؟؟

\*ما گیت رو از سایت اصلی دانلود کرده نصب میکنیم تا آخرین ورژن را داشته باشیم و موقع نصب تیک آپدیت اتوماتیک رو میذاریم باشه

وقتی گیت

\*برای شروع بهتر است فولدر کاری خود را در محل مورد نظر ایجاد کرده و با راست کلیک گیت بش کنیم Git Bash Hear \*

\*اینجا پایگاه ما برای کار با گیت در این پروژه ست و از آنجا که ما همیشه چندین پروژه کاری داریم هر گاه در مورد هر کدام خواستیم کار کنیم آنرا با روش بش کردن در گیت فعال میکنیم در این صورت ما چندین محیط گیت خواهیم داشت ولی در فولدرهای کاری متفاوت(بهتر است از این کار صرفنظر کرد چرا که باعث سردرگمی میشود )

\*git bash hear

\* با استفاده از این شورت کات ما فولدر مورد نظر را فولدر جاری میکنیم

استفاده از فرمانهای داس مانند mk و cd .. و تعداد دیگری از فرمانها باعث ساخت فولدر و جابجایی در گیت میشود و اگر از فولدر کاری خارج شده و دستورات گیت را اجرا کنیم باعث میشود به نتیجه مناسب نرسیم پس بودن در فولدر کاری شرط لازم برای درست کار کردن گیت است مگر اینکه میسر را نیز به دستورات اضافه کنیم

\*???? ?? ???? ??? ?????

\*\* ?? ???????? ?? ??? ??????? ?? ??????? ????? ??? ???? ? ???? ?? ???? ?? ????? ??????? ????? ????? ? ??? ????? ??? ?? ?? ?? ????? ???? ???? ????? ????? ??? ????? ??? ???? ???? ????? ?? ?? ?? ??? ?????

git خالی تمام دستورات را نشان میدهد

git init (repository اولین کاری که یک نفر خود را راننده‌ ماشین اعلام میکند اینست که پشت فرمان مینشیند در گیت برای اینکه خود را مصدی این فولدر معرفی کنیم و این دستور است

با این دستور گیت فولدر فعلی را مخزن و خود را در شاخه اصلی قلمداد میکند

حالا که خودمون رو متصدی معرفی کردیم مشخصات میدیم

Git config user.name <Name> معرفی نام متصدی این گیت است

Git config user.email <Email> معرف ایمیل متصدی است

\* استفاده مجدد از گیت اینیت git init باعث خرابکاری میشود زیرا مجددا مخزن خالی معرفی میکند و کل فعالیتهای ارزشمند روزها و ماهها و سالهای پروژه در سمت شما ریست میشود

با این دستور مرکز فرماندهی این پروژه در فولدر مخفی .git ایجاد میشود و حافظه‌ی کاری شما خواهد بود

git status Show the working tree status

اگر بخواهیم از موقعیت فایلهای خود را در گیت مطلع شویم از git status استفاده میکنیم

grow, mark and tweak your common history

فرض کنیم تعداد زیادی فایل داریم که نمیخوایم تو پروژه git باشه ولی میخوایم تو همین فولدر باشن

همه این فایلها و فولدرها در وضعیت untrack و بدون رد هستند هرچند در فولدر گیت شده باشند

ما سه حالت برای فایلهامون داریم

1-modidy در دست تعمیر(وقتی status میگیریم قرمز رنگ) و تعمیر شده(سبز رنگ)

2-stage وقتی دستور git add زده میشه به حالت آماده برای نهایی شدن قرار میگیره

3-commit به مدیریت پروژه ارسال میشه با دستور git commit …

Git init

Git config

Git status

Git add (~ Git rm )

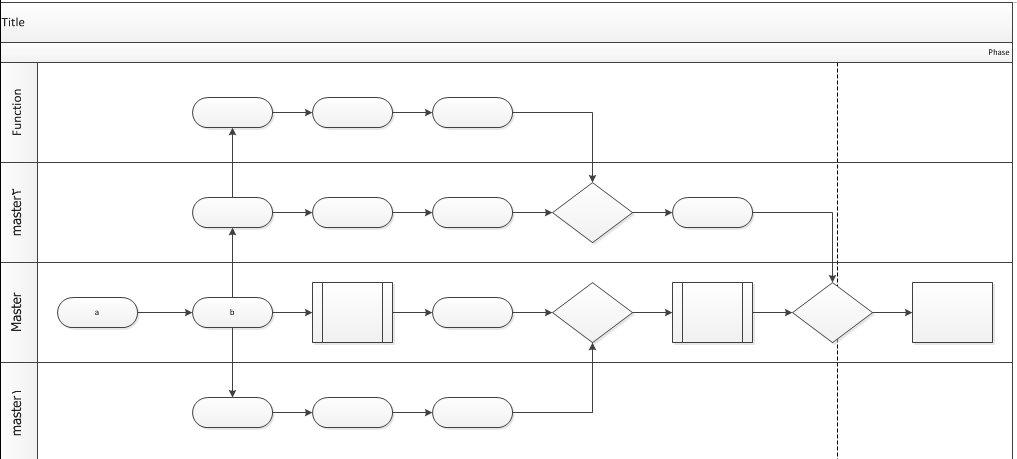
Git commit

Git diff

Git log

Git rese HEAD فایل کامیت شده رو به حالت ورک تری برمیگردونه(قبل استیج)

Git checkout فایلهای ریست شده رو بر میگردونه به قبل ترک شدن (حالت آندو)



git add . / -All / \* / <name> ???? ???? ???? ? ????? ?? ???? ??? ????? ???

usage: git

[--version]

[--help]

[-C <path>]

[-c <name>=<value>]

[--exec-path[=<path>]]

[--html-path]

[--man-path]

[--info-path]

[-p | --paginate | -P | --no-pager]

[--no-replace-objects]

[--bare]

[--git-dir=<path>]

[--work-tree=<path>]

[--namespace=<name>]

<command>

[<args>]

These are common Git

commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

sparse-checkout Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow,

mark and tweak your common history

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides.

See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.