Lecture Summary

Keywords:

- preface this book aims quick reference git commands introduction git popular version control software refer git handbook book published late 2024 l g l flavio copes 2024 rights reserved downloaded flaviocopes com part book may reproduced distributed transmitted form any means including photocopying recording electronic mechanical methods without prior written permission publisher information book educational informational purposes not intended legal financial professional advice author publisher make no representations accuracy completeness suitability validity information in this book liable errors omissions delays information any losses injuries damages arising use this book provided free charge newsletter subscribers flavio copes for personal use redistribution resale commercial use book portions strictly prohibited without prior written permission author wish share portion book please provide proper attribution crediting flavio copes including link flaviocopes com preface welcome git cheat sheet extensive guide crafted empower novice and seasoned developers knowledge needed effectively utilize git popular version control system software industry cheat sheet designed go-to resource whether managing solo project collaborating within large team by providing clear explanations practical examples aims demystify git's complexities and transform intuitive actionable insights throughout guide explore wide array git commands concepts that form backbone software version control fundamental operations like initializing repositories committing changes advanced techniques branching merging rebasing cheat sheet covers also delve specialized topics like squashing commits bisecting debug handling submodules implementing subtrees ensuring well-prepared tackle challenge arises your development process progress learn maintain data integrity manage multiple working trees and resolve merge conflicts efficiently section structured provide step-by-step guidance empowering apply learn immediately projects end of journey deeper understanding git also confidence to use streamline workflow enhance collaboration peers to fully benefit cheat

sheet recommended readers possess foundational knowledge command-line operations general programming principles familiarity

with using terminal command prompt significantly aid understanding applying the examples provided additionally basic grasp version

control concepts will enhance ability navigate guide effectively bicg c n sin section delve fundamental git commands serve

building blocks efficiently managing navigating git repositories git distributed version control system essential tracking changes

codebase collaborating other developers maintaining integrity project history understanding basic commands crucial anyone looking

leverage full power git development workflow we'll explore variety commands cover key aspects git usage initializing new

repositories committing changes branching merging command explained with short sentence describes purpose practical examples

illustrate it can effectively used real-world scenarios whether setting new project or working existing codebase commands help keep

work organized and maintain seamless workflow git help the git help command prints git help information provides quick reference

to git's basic usage commonly used git commands command useful when need quick reminder git's functionality want explore

available commands also use git help <command> display help information specific git command example git help git prints git

help specifically git command itself these help commands valuable resources beginners experienced users to quickly access

information git's features usage git version the git version command displays version git installed system this command useful

verifying version git currently using be important compatibility certain features troubleshooting issues git init the command git init used

initialize new git repository current directory command creates new subdirectory named git contains necessary metadata new

repository typically first command run starting new project want manage git running command begin tracking files making commits

new git repository git clone <repository_url> git clone <repository_url> command creates copy remote git repository on your local

machine downloads files branches commit history allowing to start working project immediately git status the git status command shows current state git repository's working directory and staging area displays information files modified added or deleted whether

changes staged next commit the w r k g i r e c r n the tag g r e a the working directory staging area fundamental concepts git play

crucial role version control process working directory environment you actively make changes files representing current state project

is essentially sandbox freely edit delete create files develop your project however changes local machine yet part version history the

staging area also known index serves intermediary space working directory repository acts checkpoint you can selectively organize

changes committed repository's history this allows prepare set changes logically related ensuring commits meaningful coherent the

commands facilitate management changes working directory staging area enable add files staging area remove them modify existing

ones giving control included next commit using commands ensure intended updates are committed making project's history clear

organized process essential for maintaining clean understandable history allows track evolution of your project precision clarity git

checkout git checkout command discards changes working directory reverting files last committed state this command useful quickly

undoing local modifications restoring working directory clean state git reset -p the git reset -p command allows interactively reset

changes working directory provides way selectively undo modifications giving fine-grained control over changes keep discard git add

<file> the git add <file> command adds specific file staging area git prepares the file inclusion next commit allowing selectively

choose changes to include version history git add -p allows interactively stage changes working directory breaking into chunks hunks

enabling review selectively add parts changes index before committing git add -i enters interactive mode adding files provides

text-based interactive menu where you select various actions perform staging individual changes updating files or viewing status git

rm <file> remove file working directory stages removal git rm --cached <file> removes specified file staging area index leaves intact

working directory effectively untracking file version control git mv <old_path> <new_path> the git mv <old_path> <new_path>

command used move rename file or directory within git repository automatically stages change making ready the next commit git

commit -m message git commit -m message command used create new commit git repository saves changes staged added
index along descriptive message message briefly explain changes made this commit wkg with brnches git branches parallel
lines development within git repository allowing work on different features bug fixes experiments independently main codebase
each branch commit history changes made one branch affect others merged helps organizing work facilitates collaboration
by enabling multiple developers work different aspects project simultaneously without interfering other's progress section we'll
introduce commands allow create switch list rename and delete branches git repository commands help manage parallel lines
of development enabling work different features bug fixes independently you'll also learn display commit histories branch
relationships well manage remote branches git branch <branch_name> create new branch git checkout <branch_name> switch
specified branch update working directory git branch list branches git branch -d <branch_name> delete branch git push --delete
<remote> <branch> delete remote branch git branch -m <old_name> <new_name> rename branch git checkout -b
<new_branch> create switch new branch named <new_branch> based current branch git switch <branch> switch working directory
specified branch git show-branch <branch> display summary commit history branch relationships selected branches showing branch
diverged git show-branch -a all same branches commits git branch -r list remote branches local repository aware of git branch -a lists
branches repository including local remote branches ones the local repository aware of git branch --merged lists branches fully
merged current branch safely deleted longer needed git branch --no-merged list branches fully merged current branch showing
branches with changes integrated yet m r g g the git merge command used combine changes one branch another branch integrates
histories branches creating new commit includes the changes sources process allows multiple lines development brought together
facilitating collaboration ensuring updates incorporated the main project merge conflicts may arise changes overlap requiring
manual resolution ensure coherent final result git merge <branch> integrate changes specified branch current branch combining

their histories git merge --no-ff <branch> merge specified branch current branch always creating new merge commit even fast-forward

merge possible git merge --squash <branch> combines changes specified branch single commit preparing the changes commit

current branch without merging branch's history allowing you to manually edit commit message git merge --abort cancels ongoing

merge process restores state working directory and index merge started git merge -s <branch> git merge —-strategy=ours

<branch> performs merge using strategy keeps current branch's changes and discards changes specified branch effectively merging

histories without integrating changes branch git merge -strategy=theirs <branch> merges specified branch current branch using

strategy which resolves conflicts favoring changes branch merged note theirs strategy built-in strategy usually requires custom

scripting used tools to handle conflict resolution notes git remotes references remote repositories versions project hosted on internet

another network enable collaboration allowing multiple users to share sync changes central repository common operations remotes

include git fetch retrieve updates git pull fetch merge changes git push upload local commits remote repository managing remotes

involves adding removing and renaming remote connections well configuring urls seamless collaboration git fetch fetch changes

remote repository merge current branch git pull fetch changes remote repository immediately merges current branch git push upload

local branch's changes remote repository git remote lists names remote repositories configured local repository git remote -v display

urls remote repositories associated local repository showing both fetch push urls git remote add <name> <url> adds new remote

repository specified name url local repository configuration git remote remove <name> git remote rm <name> deletes specified remote

repository connection local git configuration git remote rename <old_name> <new_name> changes name existing remote repository

connection local git configuration git remote set-url <name> <newurl> changes url existing remote repository connection local git

configuration git fetch <remote> retrieves latest changes specified remote repository updating local copy of remote branches without

merging local branches git pull <remote> fetches changes specified remote repository merges current branch git remote update fetches updates remotes tracked repository git push <remote> <branch> uploads specified branch local repository given

remote repository git push <remote> --delete <branch> removes specified branch remote repository git remote show

<remote> displays detailed information specified remote repository including url fetch and push configurations branches tracks git

ls-remote <repository> lists references branches tags commit ids specified remote repository command allows view branches tags

available remote repository without cloning git push origin <branch> --set-upstream pushes local branch remote repository origin

sets local branch track the remote branch future git push git pull commands default remote branch git remote add upstream

<repository> adds new remote named upstream local repository pointing specified url this is commonly used track original repository

forked origin typically refers fork git fetch upstream retrieves updates upstream remote repository updating local references to the

branches tags remote without modifying working directory merging changes git pull upstream <branch> fetches updates upstream

remote repository merges changes into your current branch often used integrate changes original repository into your local branch git

push origin <branch> uploads local branch origin remote repository making branch's commits available remote a en g c its amending

git commit allows modify recent commit typically correct or update contents message done using git commit --amend

command which opens commit default text editor changes amending particularly useful fixing small mistakes adding forgotten

changes without creating new commit resulting cleaner accurate commit history git commit --amend modify recent commit combining

staged changes git commit --amend -m new message amends commit message recent commit git commit --fixup=head creates new

commit --fixup option intended correct amend the most recent commit instead new commit marked fixup prefix commit message used

automatically fix amend specified commit an interactive rebase stash git stashing feature allows temporarily save changes

working directory yet ready committed using git stash command can set aside changes revert working directory clean state enabling

to switch branches perform tasks without losing progress later reapply the stashed changes git stash apply git stash pop allowing continue you left functionality especially useful managing work progress need to address urgent issue experiment different code

path git stash git stash save temporarily save uncommitted changes allowing switch branches perform other operations without

committing incomplete work git stash -m message git stash save message same stores changes message git stash show display

summary changes recent stash entry showing files were modified git stash list show stashed changes repository displaying numbered

list git stash pop apply recent stash immediately remove stash list git stash drop remove recent stash entry stash list without applying

working directory git stash apply reapply recently stashed changes working directory without removing them from stash list git stash

clear clear remove stashed entries permanently deleting saved changes stash git stash branch <branch> creates new branch named

commit stashing your changes applies stashed changes new branch command effectively allows continue working stashed changes

separate branch preserving original context changes tag git tagging feature allows mark specific points repository history

as important meaningful name often used releases significant milestones unlike branches tags typically immutable changes serving

permanent reference particular commit two types tags git: lightweight tags which are simple pointers commit annotated tags store

additional metadata like the tagger's name email date message tags easily created listed pushed to remote repositories deleted

providing convenient way manage reference key points project development timeline git tag <tag_name> create new tag specified

name pointing current commit typically used to mark specific points commit history like releases git tag -a <tag_name> -m

message create annotated tag specified name message includes additional metadata like tagger's name email date points current

commit git tag -d <tag_name> deletes specified tag local repository git tag -f <tag> <commit> force tag point different commit git

show <tag_name> display detailed information specified tag including commit points any associated messages annotations git

push origin <tag_name> upload specified tag remote repository making available others git push origin --tags push local tags remote

repository ensuring tags synchronized the remote git push --follow-tags push commits tags git fetch --tags retrieve tags default

remote repository updates local repository with them without affecting current branches r e v e r ting c h n g e s reverting changes git
involves undoing modifications made repository history this can accomplished several commands git revert creates new commit
negates changes specified previous commit effectively reversing its effects preserving commit history another method using git
reset which changes current head specified commit update staging area and working directory depending chosen options --soft
--mixed --hard additionally git checkout used discard changes working directory reverting files their state last commit tools provide
flexibility managing correcting changes ensuring repository remains accurate clean git checkout -- <file> discards changes specified
file working directory reverting state of the last commit effectively undoing modifications staged git revert <commit> creates new
commit undoes changes specified commit effectively reversing effects preserving history git revert -n <commit> reverts commit
commit result git reset resets current head specified state optionally updates staging area and working directory depending options
used --soft --mixed --hard git reset --soft <commit> moves head specified commit keeping index staging area working directory
unchanged changes specified commit remain staged for committing useful want undo commits keep changes ready be committed
again git reset --mixed <commit> moves head specified commit updates index staging area match that commit leaves working
directory unchanged changes specified commit are kept untracked git reset --hard <commit> moves head specified commit updates
index staging area working directory match commit discarding changes untracked files specified commit v i e w g h r l g s git history
refers record changes made repository time includes a chronological sequence commits representing snapshot repository a specific
point history allows developers track modifications understand evolution of codebase collaborate effectively providing detailed log
made changes tools like git log help navigate history offering insights the development process aiding debugging project
management git log display commits log git log --oneline displays summary commits one line each git log --graph shows graphical
representation commit history git log --stat displays file statistics along commit history git log --pretty=format:"%h" formats log

output according specified format git log --pretty=format: %h %an %ar : %s provides human-readable format logs git log

--author=<author> shows commits made specified author git log --before=<date> shows commits made specified date git log

--after=<date> git log --since=<date> shows commits made specified date git log --cherry-pick omit commits equivalent two

branches git log --follow <file> shows commits file including renames git log --show-signature displays gpg signature information

commits git shortlog summarizes git log output author git shortlog -sn summarizes git log output author commit counts git log

--simplify-by-decoration shows commits referenced tags branches git log --no-merges omits merge commits log git whatchanged lists

commit data format similar commit log git diff-tree --pretty --name-only --root <commit> shows detailed information commit tree git log

--first-parent show only commits current branch exclude merged branches diffs git diffs feature git allow see differences various

states in your repository include differences working directory staging area staging area last commit two commits or branches

displaying line-by-line changes files diffs help review modifications before committing merging applying changes thus ensuring

accuracy consistency in your codebase git diff shows differences various states repository your working directory index staging area

index last commit or between two commits displays line-by-line changes files helping review modifications committing merging git diff

--stat shows summary changes working directory index staging area helping see files modified many lines added or removed git diff

--stat <commit> view changes commit working directory git diff --stat <commit1> <commit2> provides summary changes two

commits showing files altered and extent changes them git diff --stat <branch1> <branch2> summarizes differences two branches

indicating files changed the magnitude changes git diff --name-only <commit> shows names files changed specified commit git diff

--cached shows differences staged changes index last commit helping you review included next commit git diff head shows dif

ferences working directory latest commit head allowing see changes made since last commit git diff <branch1> <branch2> shows dif

ferences tips two branches highlighting changes between the commits end branch git difftool launches diff tool compare changes git difftool <commit1> <commit2> uses difftool show differences two specified commits git difftool <branch1> <branch2> opens difftool

compare changes two branches git cherry <branch> compares commits current branch another branch shows which commits unique

branch commonly used identify commits one branch applied another branch git flow git flow branching model git provides robust

framework managing larger projects defines strict branching strategy designed around project release cycle with two primary

branches main develop supporting branches features releases and hotfixes model helps organizing work ensuring clean

manageable history facilitating collaboration clearly defining roles processes different types of development work git flow

init initializes repository git-flow branching model git flow feature start <feature> starts new feature branch git-flow git flow feature

finish <feature> finishes feature branch git-flow exploring references git references often referred refs pointers specific

commits objects within a git repository include branches tags references like head which points current commit checked working

directory references used to keep track structure history repository enabling git efficiently manage and navigate different points

projects timeline provide way name refer to particular commits making easier work manipulate repository history git show-ref

--heads lists references heads branches git show-ref --tags lists references tags configuration git configuration involves setting various

options preferences control the behavior git environment include specifying username email setting default text editors creating

aliases commonly used commands and configuring global ignore files configuration settings applied different levels: global affecting

repositories system local affecting single repository and system-wide settings ensure customized consistent user

experiences streamline workflows enhance overall efficiency version control operations git config --global

username http://username/yourname sets user name global level git config --global

useremail http://useremail/your_email@example.com sets user email global level git config --global core editor <editor> sets default text

editor git config --global core excludesfile <file> sets global ignore file git config --list lists configuration settings git config --list

--show-origin lists config variables showing origins git config <key> retrieves value specified key git config --get <key> retrieves value

specified configuration key git config --unset <key> removes specified configuration key git config --global --unset <key> removes

specified configuration key globally s e c u r ity git gpg security involves using gnu privacy guard gpg sign commits tags ensuring

authenticity integrity configuring gpg key enabling automatic signing developers verify commits tags created trusted

source preventing tampering ensuring integrity repository's history practice enhances security providing cryptographic assurance

changes come legitimate contributor git config --global user.signingkey <key> configures gpg key signing commits tags git config

--global commit.gpgsign true automatically signs commits gpg s e tting lia e s git aliases custom shortcuts create simplify speed

workflow by mapping longer git commands shorter memorable names configuring aliases git settings quickly execute frequently used

commands less typing enhances productivity also reduces likelihood errors for example set alias like git st replace git status git co

replace git checkout aliases defined globally apply across repositories locally individual projects providing flexibility streamline git

operations git config --global alias.ci commit sets git ci alias git commit git config --global alias.st status sets git st alias git status git

config --global alias.co checkout sets git co alias git checkout git config --global alias.br branch sets git br alias git branch git config

--global alias.graph log --graph --all --oneline --decorate creates alias detailed graph repository history r e b git rebasing re-applies

changes top another branch's history creating cleaner linear project history practice helps integrate updates smoothly

avoiding unnecessary merge commits ensuring commit sequence straightforward making easier understand evolution project git

rebase <branch> git rebase command used re-apply commits top another base tip allows you move combine sequence commits

new base commit commonly used the basic usage git rebase rebase current branch onto specified branch git rebase --interactive

<branch> starts interactive rebase session allowing modify commits starting the current head lets reorder squash edit delete commits

providing way to clean refine commit history pushing changes shorter version git rebase -i <branch> git rebase --continue continues rebase process resolving conflicts git rebase --abort aborts rebase process returns original branch git fetch --rebase Keep linear

project history 2 integrate changes one branch another 3 update feature branch latest changes main branch fetches remote

repository rebases local changes ch e r r -pick g git cherry-picking process allows apply changes introduced specific commit one

branch another branch particularly useful want to selectively incorporate individual changes different branches without merging

entire branch using git cherry-pick command isolate integrate desired commits ensuring specific modifications included current

branch while avoiding potential conflicts unwanted changes parts branch git cherry-pick <commit> applies changes introduced

existing commit git cherry-pick --continue continues cherry-picking resolving conflicts git cherry-pick --abort aborts cherry-pick

process git cherry-pick --no-commit <commit> cherry-picks commit without automatically committing allows changes shorter version:

git cherry-pick -n <commit> 4 ch g git patching method used apply changes one repository another one branch another within

repository involves creating patch files text files representing differences commits branches patch files be applied repository using

commands like git apply git allowing changes be transferred integrated without directly merging branches patching particularly

useful for sharing specific changes updates across different codebases ensuring the intended modifications applied git apply

<patch_file> applies changes working directory patch file git apply --check checks patches applied cleanly git format-patch

<since_commit> creates patch files commits since specified commit git <patch_file> applies patches mailbox git --continue continues

applying patches resolving conflicts git --abort aborts patch application process git diff > <filepatch> creates patch file differences rela

tive tes git relative dates allow users refer specific points repository's history using human-readable date expressions instance

commands like main@{1 week ago} or @{3 days ago} enable access state branch view changes made since a certain time period

relative current date feature simplifies navigating the repository's timeline using intuitive terms like yesterday 2 weeks ago

specific dates making easier track manage evolution codebase without needing to remember exact commit hashes timestamps git show main@{1weekago} state main branch one week ago: git diff@{3daysago} see changes made last 3 days: git checkout

main@{2weeksago} check repository 2 weeks ago: git log@{1monthago} head see commits 1 month ago

now@{2024-06-01}@{yesterday}@{1week}@{2daysago} other usage examples tag git blaming feature git identifies last modification

made line file attributing changes specific commits authors done using git blame command provides detailed annotation file showing

made changes and when made tool particularly useful tracking history file understanding evolution code identifying source bugs

changes by pinpointing exact commit author responsible line developers gain insights development process facilitate better

collaboration accountability within team git blame <file> shows last modification line file git blame <file> -l <start> <end> limits blame

output specified line range git blame <file> <commit> shows blame information specified commit git blame <file> -c -c shows revisions

authors last modified line file copying detection the -c option detects lines moved copied within file using -c detects lines moved

copied within file using -c option twice -c -c makes git inspect unmodified files candidates source copy meant try to find origin copied

lines file files well git blame <file> --reverse works backwards showing last altered line specified file git blame <file>

--first-parent shows recently modified line file following first parent commit for merge changes rachiv git archiving feature allows

create archive files tar zip containing contents specific commit branch tag useful packaging snapshot repository specific point time

enabling distribute backup the repository's state without including entire git history git archive command is typically used purpose

providing convenient way export current state the project portable format git archive <format> <tree-ish> creates archive file eg tar zip

file containing contents specified tree-ish like commit branch tag given format example: git archive --format=tar head creates tar

archive current commit head git archive --format=zip v1.0 creates zip archive files v1.0 tag this command useful packaging snapshot

repository specific point in time rtckg git tracking refers process monitoring managing files repository the command git ls-files lists

files tracked git providing clear view the files currently version control hand git ls-tree displays the contents tree object specified branch showing structure files pointin repository together commands help developers understand files are included repository

organized ensuring efficient tracking and management project's codebase git ls-files lists tracked files git ls-tree <branch> lists

contents tree object index manipulation git index manipulation involves managing staging area also known index where changes

prepared committing include marking files assume unchanged temporarily ignore changes resetting markings track changes again

index manipulation commands git update-index allow control which files included next commit providing flexibility handling changes

optimizing the workflow specific tasks git update-index --assume-unchanged <file> marks file assume unchanged git update-index

--no-assume-unchanged <file> unmarks file assume unchanged squash git squashing process combining multiple commits single

commit is often done clean commit history merging changes main branch making history concise easier read squashing performed

using the interactive rebase command git rebase -i allows developers selectively merge reorder edit commits squashing commits

redundant minor changes be consolidated presenting clearer narrative development process git rebase -i head~<n> squashes

commits interactively integrity git data integrity refers mechanisms processes git employs ensure the accuracy consistency data

within repository git uses cryptographic hashes sha-1 or sha-256 uniquely identify objects commits trees blobs hashing not only

provides unique identifier object also ensures modification the object's content result different hash thus detecting corruption

tampering commands like git fsck used verify connectivity validity objects the database ensuring overall health integrity repository git

fsck verifies connectivity validity objects database git fsck --unreachable finds objects repository reachable reference git prune remove

unreachable objects git prune garbage collection process git garbage collection maintenance process cleans optimizes repository by

removing unnecessary files compressing file revisions save space process triggered git gc command consolidates deletes

unreachable objects as orphaned commits unreferenced blobs ensuring repository remains efficient and performant regular garbage

collection helps manage storage effectively keeps the repository's structure organized clean up cleaning git involves removing

unnecessary files references branches are no longer needed helps keep repository organized efficient regular cleanup activities
pruning remote-tracking branches deleting untracked files and removing stale references ensure repository remains manageable
free from clutter practical actions improve performance reduce storage requirements and make easier navigate work within project git
fetch --prune removes references longer exist remote git remote prune <name> prunes stale remote-tracking branches git fetch origin
--prune cleans outdated references remote repository git clean -f removes untracked files working directory forcing deletion files
being tracked git git clean -fd removes untracked files directories working directory including files and directories tracked git git clean
-i enters interactive mode cleaning untracked files git clean -x removes ignored files working directory ub tre e git subtree mechanism
managing integrating subprojects main repository unlike submodules treat subproject separate entity repository subtrees allow
include contents another repository directly within subdirectory main repository approach simplifies workflow eliminating need
for multiple repositories enabling seamless integration merging pulling updates from subproject subtrees provide flexible convenient
way manage dependencies collaborate projects require incorporating external codebases git subtree add --prefix=<dir> <repository>
<branch> adds repository subtree git subtree merge --prefix=<dir> <branch> merges subtree git subtree pull --prefix=<dir>
<repository> <branch> pulls new changes subtree's repository e r ch g git grep powerful search command git allows users search
specific strings or patterns within files repository searches working directory the index providing quick efficient way locate
occurrences specified pattern across multiple files command particularly useful developers looking find instances of code comments
text within project enabling navigate understand large codebases ease various options flags git grep perform targeted searches
making essential tool code analysis maintenance git grep <pattern> searches string working directory index git grep -e
<pattern> searches specific pattern e ecting git bisecting powerful debugging tool helps identify specific commit that introduced bug
issue project performing binary search commit history git bisect efficiently narrows range potential problem commits the process

involves marking known good commit known bad commit then repeatedly testing intermediate commits determine whether good bad

this iterative approach quickly isolates faulty commit allowing developers pinpoint the exact change caused problem thereby

facilitating faster accurate debugging git bisect start starts bisect session git bisect bad marks current version bad git bisect good

<commit> marks specified commit good git bisect reset ends bisect session returns original branch git bisect visualize launches visual

tool assist bisecting attributes git attributes settings define git handles specific files paths within a repository attributes defined file

named gitattributes control various behaviors text encoding line-ending normalization merge strategies and diff algorithms setting

attributes ensure consistent behavior across different environments collaborators making easier manage files special

requirements complexities example mark certain files binary prevent git from attempting merge specify custom diff drivers

meaningful comparisons git check-attr <attribute> <file> shows value specific attribute given file defined gitattributes configuration

helping understand git treating file respect attributes like text encoding merge behavior diff handling checkout git checkout versatile

command git used switch different branches tags or commits within repository updating working directory index match the specified

branch commit allows view work state repository at that point additionally git checkout used create new branches restore specific files

commit even start new branch history using --orphan option this command essential navigating managing different versions

project's codebase git checkout <commit> updates working directory index match specified commit allowing view work state

repository commit leaves detached head state meaning branch git checkout -b <branch> <commit> creates new branch named

starting specified commit switches that branch allowing begin working point commit history git checkout <commit> -- <file> restores

specified file specific commit working directory replacing the current version file version commit without changing commit history

index git checkout --orphan <new_branch> creates new branch named <new_branch> commit history effectively starting a new

branch begins clean working directory index new repository reflog git reflog powerful tool records changes made tips branches the head reference git repository includes actions commits checkouts merges resets maintaining history changes reflog allows users

track recent modifications recover lost commits even part current branch history provides way navigate repository's state changes

making an invaluable resource debugging undoing mistakes git reflog displays log changes head reference branch tips including

commits checkouts merges resets allowing recover lost commits track recent changes repository's state git reflog show

<ref> displays reflog specified reference showing log changes reference including updates head branch tips along associated commit

messages and timestamps plih g u n t r c k e f iles git clean removes untracked files directories working directory default shows what

would removed without actually deleting anything perform actual cleanup you need use additional flags: f c e p u h g git push

--force force push local branch remote repository even results non-fast-forward merge overwrites remote branch local changes

be necessary rewritten history e.g rebase need update the remote branch match local branch also potentially overwrite

others' changes used caution fetch g n pulling git fetch --all retrieves updates remote repositories configured local repository

fetching changes branches tags without modifying local branches git pull --rebase fetches changes remote repository rebases local

commits top the updated remote branch rather merging keeps commit history linear and avoids unnecessary merge commits h ling

e r g e c n flicts git clean -f: removes untracked files git clean -fd: removes untracked files directories git clean -fx: removes untracked

files including ignored gitignore git clean -n: shows files would removed without actually deleting them handling merge conflicts git

essential skill collaborating projects multiple contributors merge conflicts occur changes different branches commits overlap or

contradict preventing automatic merge resolving conflicts involves reviewing manually reconciling differences ensure final code

integrates contributions parties accurately practice effectively managing merge conflicts helps maintain code integrity facilitates

smooth collaboration ensuring that everyone's changes correctly incorporated project's history git mergetool launches merge tool

help resolve conflicts arise merge rebase it opens graphical interface text-based tool configured git settings allowing you to manually resolve conflicts finalize merge git rerere rerere stands reuse recorded resolution feature helps automatically resolve conflicts future

merges rebases reusing conflict resolutions previously recorded enabled git records resolved conflicts conflicts arise apply

resolutions automatically work g t r e e s working trees git allow multiple working directories associated single repository particularly

useful working multiple branches simultaneously without the need constantly switch branches directory using working trees you can

easily manage different features bug fixes experiments isolated environments improving workflow efficiency reducing risk

conflicts git worktree add ../new-branch feature-branch creates new working tree directory named new-branch based feature-branch git

worktree list lists working trees associated current repository showing paths the branches checked to git worktree remove

<path> removes specified working tree given git worktree prune removes references nonexistent working trees cleaning working tree

list git worktree lock <path> locks specified working tree given git worktree unlock <path> unlocks specified working tree given s u b

less submodules git way include manage external repositories within own repository particularly useful reusing code across multiple

projects maintaining dependencies integrating third-party libraries using submodules keep your main repository clean modular still

ensuring necessary components are included version-controlled git submodule init initializes submodules repository command sets

configuration necessary for submodules actually clone them git submodule update clones checks submodules specified paths

typically run git submodule init git submodule add <repository> <path> adds new submodule repository specified path linking

specified repository git submodule status displays status submodules showing commit hashes whether are up-to-date modified

uninitialized git submodule foreach <command> runs specified command submodule useful performing batch operations across

submodules git submodule sync synchronizes submodule urls configuration file gitmodules file ensuring up-to-date git submodule

deinit <path> unregisters specified submodule removing configuration delete the submodule's working directory git submodule

update --remote fetches updates submodules latest commit remote repositories git submodule set-url <path> <newurl> changes url specified submodule new url git submodule absorbgitdirs absorb submodule's git directory superproject simplify structure

Summary:

git init

The command git init is used to initialize a new Git repository in the current directory.

Searching

git grep is a powerful search command in Git that allows users to search for specific strings

or patterns within the files of a repository

This process is essential for

maintaining a clean and understandable history, as it allows you to track the evolution of

your project with precision and clarity.

git grep <pattern>

Searches for a string in the working directory and the index.

Reflog

Git reflog is a powerful tool that records all changes made to the tips of branches and the

HEAD reference in a Git repository

Resources: