Git is divided into ***porcelain commands***, which are the high level commands frequently, used by GIT users. Also, GIT features ***plumbing commands***, which are *low level commands* that are often combined in the background to perform *high level actions. Below you will find a* ***shortcut/workflow overview*** *that briefly explains the high yield* porcelain commands and displays examples of usage for quick reference.

git setup: (chain of commands for starting up)

#Create GIT Repo
git init
#Check the version of GIT
git version
#Define GIT user name
git config --global user.name “John Doe”
#Define GIT user email
git config --global user.email “john.doe@site.com”
#View gitconfig
cat ~/.gitconfig
#Create ignore file
touch .gitignore
#Review Defined user info
git config -l
#Making Shortcuts for checkout
#setup alias “co” for checkout
git config --global alias.co checkout
#setup git to ignore white space for reviewing diffs
git config --global apply.whitespace nowarn
#setup coloring for easier view in the command line
git config --global color.interactive auto
#setup garbage collection to occur automatically
git config --global gc.auto
#change editor that git will target
git config --global core.editor 'emacs'
#Customize GITK
vi ~/.gitk
***Mac User consider changing fonts to these defaults:**
set mainfont {Monaco 12}
set textfont {Monaco 12}
set uifont {"Monaco Bold" 12}
***Defaults okay for windows, but feel free to update anything for any platform**

git house cleaning: (chain of commands)

#Check for corrupt or unreachable files
git fsck
#Clean corrupt or unreachable files
git prune
#Perform garbage collection (takes loose objects and packs them, optimizes packed files)
git gc

git log: (view commit info)

#View logs of commits
git log
#View log details of content changes throughout each commit
git log -p
#Quick view of changes with stats
git log --stat
#View range of commits from the last commit (see logs 1 back and 5 back in the examples below)
git log -1
git log -5
#View a specific log for a revision (log from revision id# SHA-1: eef767594990f5823a19de12c1cf78f444af7542, only need about 4-7 chars for a unique match)
git log eef7675
#See commits log on one line per commit (options: oneline, short, medium, full, fuller, email, or raw)
git log --pretty=oneline
#View commits based on time:
#All log since 1 day ago, followed by 2 days ago
git log --since="1 day ago"
git log --since="2 days ago"
#All logs between March 6th 2010 and March 14th 2010
git log --before="2010-03-14" --after="2010-03-06"
#Search logs for a particular author
git log --author=“John”

#See changes from one particular branch (in this case a branch named feature1)
git log feature1
git log master

git stash: (put staged changes into a safe place for a clean workspace)

#list stash (Stash is zero based in count, latest stash is 0)
git stash list
#Remove a stack at a ceartin index (*in this case 1)
git stash drop stash@{1}
#Apply a ceartin stash for a specific index (*in this case 1)
git stash apply stash@{1}
#Apply last stash and remove the stash from the stash list (*in this case 0)
git stash pop
#Clear stashes
git stash clear
#Save a stash
git stash save “stashed for hotfix in another branch”

git k: (REPO browser from command line)

#Run gitk
gitk -a
#Run gitk in the background
gitk -a &

git gui: (Commit Manager from command line)

#Run git gui
git gui
#Run git gui in the background
git gui &

git add: (Setting files up for staging aka: marking files as commit worthy)

#Add all non-staged files in this directory and all it's child directories
git add .
#Add a specific file
git add *relative_path/file_name.extension i.e. git add ../index.php*
#Add specific files
git add file1 file2 file3 i.e. *git add index.php banner.jpg footer.jpg sitemanager.xml*
#Add files interactively (command line system to work with what should be staged)
git add -i

git commit: (marking revisions)

#Create a commit (will open default text editor to input commit message)
git commit
#Commit with a comment (will not require text editor for commit message)
git commit -m "comment"
#Commit with add and comment in one line (will add files to staging and commit, newly created files must use git add)
git commit -m -a "comment"
#commit with diff as part of the commit message (opens default text editor for commit message, will include diff in message to start)
git commit -v -a

git checkout:

#Create branch and switch your local workspace to that branch
git checkout featureA
#Create branch and switch your local workspace to that branch
git checkout -b "featureA"
#Reverts README.txt to last commit
git checkout README.txt
#Revert all work in the local work space to the last commit
git checkout -f

git branch:

#List all of the current branches (asterisks will be to the left of the branch you are currently in)
git branch
#Create new branch named feature1
git branch feature1
#Create branch and switch your local workspace to that branch
git checkout -b “featureA”
#Delete a branch
git branch -d new
#Delete an un-merged branch
git branch -D branch1

git tag:

#List all of the current tags
git tag
#List tags that are like this (find tags that are 1. something)
git tag -l v1.*
#Create an annotated tag (commit info for tag)
git tag -a v1.4 -m 'version 1.4'
#Create a light weight tag (no commit info for tag)
git tag v1.4
#Look at a specific tag
git show v1.4

git merge:

#Checkout the branch you want to merge code into and perform the following (merges code from experiment branch)
git merge experiment
**see git mergetool*

git diff:

#Run this command after changing an existing file or files and it will display the differences
git diff
#Changes since the last commit (staged and unstaged)
git diff HEAD
#How my file has changed since a tag
git diff v1.0 -- README.txt
#See what a merge would do before performing a merge
git diff master featurebranch

git rebase:

#Re-order or squash commit set from head to two commits back
`git rebase -i HEAD~3`
#Update branch with the code from another branch, in this case the current branch is being brought up to speed with changes in master
#Also, please note this rewrites history and removes evidence of a fork in the history
`git rebase master`

git reset:

#Remove file from staging once it has been added
git reset HEAD new.txt
#Use if a merge has gone bad to restore the state of a branch before the merge
git reset --hard ORIG_HEAD
#Undo last commit but keep changes in the staging area, good for when you forget to add something to the commit
git reset --soft HEAD^
#Undo last commit and remove the changes from staging, good for when you want to get rid of all uncommitted changes
git reset --hard HEAD^

git revert:

#Revert to the most recent commit and make a commit for the revert
git revert HEAD

git cherry-pick:

#Perform a selective merge, by pulling a specific commit into another branch
#(in this case we use the sha1 from a commit in another branch to cherry pick it into a current branch)
git cherry-pick 4e38874cfda636ef0135fc7833398716184346eb

git archive:

#Create a zip or a tar of your workspace without the git db, in this case we make an archive of the master branch in a master_archive zip and then tar file.
git archive master > master_archive.zip
git archive master > master_archive.tar

git clone:

#Get a GIT REPO
git clone git://domain.com/git

git svn:

#Replay all GIT commits to SVN
git svn clone http://svn/project/trunk
#Replay all GIT commits to SVN
git svn dcommit
#Update your project with the latest from SVN
git svn rebase

git reflog:

#Recover history that has been reset, with this you can get the SHA1's to checkout, cherry-pick or merge lost changes
#Commits, Tags, Blobs and Tree's are immutable so even if changes appear lost, they are not.
git reflog

git whatchanged:

#See what has changed in the file named hello.txt
git whatchanged hello.txt

git mergetool:

#allows you to resolve issues in the default merge tool you have setup (opendiff, kdiff3, etc.)
git mergetool

git fetch:

#This runs a fetch getting the content from a remote and updating your local REPO
git fetch <remoteURL_here>l

git pull:

#This runs a fetch getting the content from a remote and updating your local REPO. It then merges the changes automatically
git pull <remoteURL_here>

git push:

#allows you to resolve issues in the default merge tool you have setup (opendiff, kdiff3, etc.)
git push <remoteURL_here>l

hooks example:

```
#!/bin/sh
#
# An example hook script that is called after a successful
# commit is made.
#
# To enable this hook, rename this file to "post-commit".

echo "====================RYSNC executed.===================="
rsync -v -a /Users/UserName/Desktop/dev/git-bin/sites/site_name/web
/Users/UserName/Desktop/sites/site_name/
```

.gitconfig:

```
[user]
        name = Lamar Hines
        email = y2kdev@gmail.com
[color]
        ui = auto
        interactive = auto
[apply]
        whitespace = nowarn
[alias]
    ci  = commit -a -m
    st  = status
    co  = checkout
    cob = checkout -b
        cl  = clone
        ll  = log
    lm = log --pretty=\"format:%ad %h (%an): %s\" --date=short
    ls = log --pretty=oneline
```

```
    l  = log
br  = branch
    a  = add .
    d  = diff
    m  = merge
    mt = mergetool
    rlc = reset --soft HEAD^
    i   = init
    v   = version
    rb  = rebase -i
```