

```

# commands for Skeleton as seen on reel

# fkControllers
fkController ("thorax", "left_clavicle", "leftClavicleController", "2.5")
fkController ("thorax", "right_clavicle", "rightClavicleController", "2.5")
fkController ("thorax", "neck", "neckController", "3")
fkController ("neck", "head", "headController", "3")
fkController ("root", "leftHip", "leftHipController", "2")
fkController ("root", "rightHip", "rightHipController", "2")
fkController ("left_clavicle", "FKleft_arm", "FKleft_armController", "2")
fkController ("FKleft_arm", "FKleft_foreArm", "FKleft_foreArm_armController", "2")
fkController ("left_foreArm", "left_hand", "left_hand_armController", "2")
fkController ("right_clavicle", "FKright_arm", "FKright_armController", "2")
fkController ("FKright_arm", "FKright_foreArm", "FKright_foreArm_armController", "2")
fkController ("right_foreArm", "right_hand", "right_hand_armController", "2")
fkController ("right_hand", "rightIndexOne", "rightIndexOne_armController", "1")
fkController ("rightIndexOne", "rightIndexTwo", "rightIndexTwo_armController", "1")
fkController ("right_hand", "rightPinkyOne", "rightPinkyOne_armController", "1")
fkController ("rightPinkyOne", "rightPinkyTwo", "rightPinkyTwo_armController", "1")
fkController ("left_hand", "leftIndexOne", "leftIndexOne_armController", "1")
fkController ("leftIndexOne", "leftIndexTwo", "leftIndexTwo_armController", "1")
fkController ("left_hand", "leftPinkyOne", "leftPinkyOne_armController", "1")
fkController ("leftPinkyOne", "leftPinkyTwo", "leftPinkyTwo_armController", "1")

# legs
IKlegSetup ("leftUpLeg", "leftLowLeg", "leftLegAnkle", "leftLegBall", "leftLegToe", 0, 0, 65)
IKlegSetup ("rightUpLeg", "rightLowLeg", "rightLegAnkle", "rightLegBall", "rightLegToe", 0, 0, 65)

# spine
spine ("root", "waist", "thorax", "rootMaster")

# arms
arm ("left_arm", "left_foreArm", "left_hand", "IKleft_arm", "IKleft_foreArm", "IKleft_hand", "FKleft_arm", "FKleft_foreArm", "FKleft_hand", 0, 60, 0)
arm ("right_arm", "right_foreArm", "right_hand", "IKright_arm", "IKright_foreArm", "IKright_hand", "FKright_arm", "FKright_foreArm", "FKright_hand", 0, 60, 0)

# foreArm
foreArmRotate ("left_foreArm", "Left_foreArmTwistOne", "Left_foreArmTwistTwo", "left_hand")
foreArmRotate ("right_foreArm", "Right_foreArmTwistOne", "Right_foreArmTwistTwo", "right_hand")

# muscle (scale the skeleton to the proper scale in a new node before activating)
muscleJoint ("left_pecUpper", "left_pecUpperEnd", "leftPecUpperTarget")
muscleJoint ("left_pecLower", "left_pecLowerEnd", "leftPecLowerTarget")
muscleJoint ("left_latBase", "left_latEnd", "leftLatTarget")
muscleJoint ("left_trapBase", "left_trapEnd", "leftTrapTarget")
muscleJoint ("right_latBase", "right_latEnd", "rightLatTarget")
muscleJoint ("right_pecUpper", "right_pecUpperEnd", "rightPecUpperTarget")
muscleJoint ("right_pecLower", "right_pecLowerEnd", "rightPecLowerTarget")
muscleJoint ("right_trapBase", "right_trapEnd", "rightTrapTarget")
muscleJoint ("left_hamstring", "left_hamstringEnd", "leftHamTarget")
muscleJoint ("right_hamstring", "right_hamstringEnd", "rightHamTarget")
muscleJoint ("leftBicepBase", "leftBicepEnd", "leftBicepLOC")
muscleJoint ("rightBicepBase", "rightBicepEnd", "rightBicepLOC")
muscleJoint ("achillesLeftBase", "achillesLeftEnd", "achillesTargetLeftLOC")
muscleJoint ("achillesRightBase", "achillesRightEnd", "achillesTargetRightLOC")

# shoulders
shoulderSetupStart ("left_clavicle", "left_arm", "left_foreArm", "left_armTwistOne", "left_armTwistTwo", 0, 0, 90)
shoulderSetupStart ("right_clavicle", "right_arm", "right_foreArmAlterShoulderTarget", "right_armTwistOne", "right_armTwistTwo", 0, 0, 90)

```

#fk controllers and automatic parenting to make a clean hierarchy

import maya.cmds as cmds

def fkController (jointParent, jointChild, controller, controllerRadius):

```
    if not cmds.objExists("groupMasterNode"):
        cmds.group (em=True, n="groupMasterNode")
    else:
        print("groupMasterNode exists")
    if not cmds.objExists("orientConstraintNode"):
        cmds.group (em=True, n="orientConstraintNode")
        cmds.parent ("orientConstraintNode", "groupMasterNode")
    else:
        print("orientConstraintNode exists")

    group = ["group"]
    control = ["control"]

    # point constraint controller to joint to be controlled
    pointConstDel = [(cmds.pointConstraint((jointChild), (cont[0][0:1])))]
    cmds.delete (pointConstDel[0])
    orientConstDel = [(cmds.orientConstraint ((jointChild), (cont[0][0:1])))]
    cmds.delete (orientConstDel[0])

    # group master node
    groupMasterControl = [(cmds.group (em=True, n= controller + (group[0])))]
    pointConstraintGroupMasterControl = [(cmds.pointConstraint((jointParent), (groupMasterControl[0])))]
    cmds.delete (pointConstraintGroupMasterControl[0])
    parentConstraintGroupMasterControl = [(cmds.parentConstraint((jointParent), (groupMasterControl[0])))]

    # node that will be parent to controller
    groupControllerNode = [(cmds.group (em=True, n=(group[0] + control[0] + controller)))]
    pointConstraintGroupMasterControl = [(cmds.pointConstraint((jointChild), (groupControllerNode[0])))]
    orientConstraintGroupMasterControl = [(cmds.orientConstraint((jointChild), (groupControllerNode[0])))]
    cmds.delete (pointConstraintGroupMasterControl[0])
    cmds.delete (orientConstraintGroupMasterControl[0])

    # parenting controller node to master controller node
    cmds.parent (groupControllerNode[0], groupMasterControl[0])
    cmds.parent (cont[0], groupControllerNode[0])
    orientConstraintControl = [(cmds.orientConstraint (cont[0], jointChild))]

    # parenting of groups to groupMasterNode
    cmds.parent (groupMasterControl[0], "groupMasterNode")
    cmds.parent (orientConstraintControl[0], "orientConstraintNode")
```

```

# beginnings ok ik setup, ideally add more attributes or better controls
# these controls are simply groups in hierarchy
# pole vector also created

import maya.cmds as cmds
def IKlegSetup (UpLeg, LowLeg, LegAnkle, LegBall, LegToe, legPVx, legPVy, legPVz):

# strings
displayHandle = [".displayHandle"]
selectionHandleY = [".selectHandleY"]
scalePivot = [".scalePivot"]
rotatePivot = [".rotatePivot"]
rotate = [".rotate"]
marker = ["marker"]
LOC = ["locator"]
poleVector = ["pV"]
legIKhandle = ["IKhandle"]
groupLeg = ["LegGroup"]
tempLoc = ["tempLoc"]
translateLeg = [".translate"]
foot = ["Foot"]
master = ["Master"]

# creation of Pole Vector controls

cmds.setAttr ((UpLeg + rotate[0]), legPVx, legPVy, legPVz)
poleVectorLocator = [cmds.spaceLocator (n = (poleVector[0] + UpLeg))]
deletedPointConstraint = [cmds.pointConstraint(LowLeg, poleVectorLocator[0])]
cmds.delete (deletedPointConstraint[0])
cmds.setAttr ((UpLeg + rotate[0]), 0, 0, 0)

# ik handles Setup

legIKhandleUpLeg = [cmds.ikHandle (n = UpLeg + legIKhandle[0], sj = UpLeg, ee = LegAnkle)]
legIKhandleBallLeg = [cmds.ikHandle (n = LegBall + legIKhandle[0], sj = LegAnkle, ee = LegBall)]
legIKhandleToeLeg = [cmds.ikHandle (n = LegToe + legIKhandle[0], sj = LegBall, ee = LegToe)]

# temporary Locators and Constraints

poleVectorLocatorLegTemp = ((cmds.spaceLocator (n = (tempLoc[0] + UpLeg))))
deletedPointConstraintAnkle = [cmds.pointConstraint(LegAnkle, poleVectorLocatorLegTemp[0])]
cmds.delete (deletedPointConstraintAnkle[0])

poleVectorLocatorBallTemp = ((cmds.spaceLocator (n = (tempLoc[0] + LegBall))))
deletedPointConstraintBall = [cmds.pointConstraint(LegBall, poleVectorLocatorBallTemp[0])]
cmds.delete (deletedPointConstraintBall[0])

poleVectorLocatorToeTemp = ((cmds.spaceLocator (n = (tempLoc[0] + LegToe))))
deletedPointConstraintToe = [cmds.pointConstraint(LegToe, poleVectorLocatorToeTemp[0])]
cmds.delete (deletedPointConstraintToe[0])

#group IK Handles

# group for IKHandles for Ankle
groupLegIKhandle = [cmds.group (em = True, n=(groupLeg[0] + UpLeg + legIKhandle[0]))]
deletedPointConstraintgroupLegIKhandle = [cmds.pointConstraint(poleVectorLocatorLegTemp[0], groupLegIKhandle[0])]
cmds.delete (deletedPointConstraintgroupLegIKhandle[0])
cmds.delete (poleVectorLocatorLegTemp[0])
cmds.parent (legIKhandleUpLeg[0][0:1], groupLegIKhandle[0])

# group for IKHandles for Ball
groupLegGroupBallIKhandle = [cmds.group (em = True, n=(groupLeg[0] + LegBall + legIKhandle[0]))]
deletedPointConstraintgroupLegIKhandleBall = [cmds.pointConstraint(poleVectorLocatorBallTemp[0], groupLegGroupBallIKhandle[0])]
cmds.delete (deletedPointConstraintgroupLegIKhandleBall[0])
cmds.parent (legIKhandleBallLeg[0][0:1], groupLegGroupBallIKhandle[0])

# group for IKHandles for Toe
groupLegGroupToeIKhandle = [cmds.group (em = True, n=(groupLeg[0] + LegToe + legIKhandle[0]))]
deletedPointConstraintgroupLegIKhandleToe = [cmds.pointConstraint(poleVectorLocatorBallTemp[0], groupLegGroupToeIKhandle[0])]
cmds.delete (deletedPointConstraintgroupLegIKhandleToe[0])
cmds.parent (legIKhandleToeLeg[0][0:1], groupLegGroupToeIKhandle[0])
cmds.delete (poleVectorLocatorBallTemp[0])

# parenting and Foot Control
cmds.parent (groupLegIKhandle[0], groupLegGroupBallIKhandle[0])
groupFoot = [cmds.group (em = True, n=(UpLeg + groupLeg[0] + foot[0]))]
deletedPointConstraintgroupLegFoot = [cmds.pointConstraint(poleVectorLocatorToeTemp[0], groupFoot[0])]
cmds.delete (deletedPointConstraintgroupLegFoot [0])
cmds.parent (groupLegGroupBallIKhandle[0], groupFoot[0])
cmds.parent (groupLegGroupToeIKhandle[0], groupFoot[0])
cmds.delete (poleVectorLocatorToeTemp[0])

# display handles
cmds.setAttr ((groupFoot[0] + displayHandle[0]), 1)
cmds.setAttr ((groupFoot[0] + selectionHandleY[0]), -3)
cmds.setAttr ((groupLegGroupBallIKhandle[0] + displayHandle[0]), 1)
cmds.setAttr ((groupLegGroupBallIKhandle[0] + selectionHandleY[0]), -2)
cmds.setAttr ((groupLegIKhandle[0] + displayHandle[0]), 1)
cmds.setAttr ((groupLegIKhandle[0] + selectionHandleY[0]), -2)

# master leg Controls
groupFootMaster = [cmds.group (em = True, n=(UpLeg+ master[0]))]
deletedPointConstraintgroupLegFootMaster = [cmds.pointConstraint(LegToe, groupFootMaster[0])]
cmds.delete (deletedPointConstraintgroupLegFootMaster[0])
cmds.parent (groupFoot[0], groupFootMaster[0])

# pole vector constraint
groupPoleVector = [cmds.group (em = True, n=(UpLeg + poleVector[0] + foot[0]))]
deletedPointConstraintPoleVectorGroup = [cmds.pointConstraint(poleVectorLocator[0], groupPoleVector[0])]
cmds.delete (deletedPointConstraintPoleVectorGroup[0])
cmds.parent (poleVectorLocator[0], groupPoleVector[0])
cmds.parent (groupPoleVector[0], groupFootMaster[0])
cmds.poleVectorConstraint( poleVectorLocator[0], legIKhandleUpLeg[0][0:1] )

if not cmds.objExists("groupMasterNode"):
cmds.group (em=True, n="groupMasterNode")
else:
print("groupMasterNode exists")

```

```
cmds.parent (groupFootMaster[0], "groupMasterNode")
```

```

# spine setup. point orient constrain base setup on spine
# moveable and rotatable hips and shoulders

import maya.cmds as cmds

def spine (root, waist, thorax, rootMaster):

    #strings
    controller = ["Controller"]

    # root control (nonMaster)
    rootControl = [cmds.circle (n = (root + controller[0]), r=4)]
    deleteSpinePointConstraint = [cmds.pointConstraint (root, rootControl[0])]
    cmds.delete (deleteSpinePointConstraint[0])

    # waist control
    waistControl = [cmds.circle (n = (waist + controller[0]), r=4)]
    deleteWaistPointConstraint = [cmds.pointConstraint (waist, waistControl[0])]
    cmds.delete (deleteWaistPointConstraint[0])

    # thorax control
    thoraxControl = [cmds.circle (n = (thorax + controller[0]), r=4)]
    deleteThoraxPointConstraint = [cmds.pointConstraint (thorax, thoraxControl[0])]
    cmds.delete (deleteThoraxPointConstraint[0])

    # conditions
    if not cmds.objExists("groupMasterNode"):
        cmds.group (em=True, n="groupMasterNode")
    else:
        print("groupMasterNode exists")

    if not cmds.objExists("spineConstraints"):
        cmds.group (em=True, n="spineConstraints")
        cmds.parent ("spineConstraints", "groupMasterNode")
    else:
        print("spineConstraints exists")

    #constraints
    pointConstraintRoot = [cmds.pointConstraint (rootControl[0], root)]
    orientConstraintRoot = [cmds.orientConstraint (rootControl[0][0:1], root, mo = True)]

    pointConstraintWaist = [cmds.pointConstraint (waistControl[0], waist)]
    orientConstraintWaist = [cmds.orientConstraint (waistControl[0][0:1], waist, mo = True)]

    pointConstraintThorax = [cmds.pointConstraint (thoraxControl[0], thorax)]
    orientConstraintThorax = [cmds.orientConstraint (thoraxControl[0][0:1], thorax, mo = True)]

    #constraint parenting
    cmds.parent (pointConstraintRoot[0], "spineConstraints")
    cmds.parent (orientConstraintRoot[0], "spineConstraints")
    cmds.parent (pointConstraintWaist[0], "spineConstraints")
    cmds.parent (orientConstraintWaist[0], "spineConstraints")
    cmds.parent (pointConstraintThorax[0], "spineConstraints")
    cmds.parent (orientConstraintThorax[0], "spineConstraints")

    #control parenting
    cmds.parent (rootControl[0], rootMaster)
    cmds.parent (waistControl[0], rootMaster)
    cmds.parent (thoraxControl[0], waistControl[0])

    # freeze Transforms

    cmds.makeIdentity (rootControl[0], apply=True, translate=True, rotate=True)
    cmds.makeIdentity (waistControl[0], apply=True, translate=True, rotate=True)
    cmds.makeIdentity (thoraxControl[0], apply=True, translate=True, rotate=True)

```

```

# fk ik setup with node for blend
# base pole vector setup

import maya.cmds as cmds
def arm (shoulder, elbow, hand, IKshoulder, IKelbow, IKhand, FKshouldler, FKelbow, FKhand, armPVx, armPVy, armPVz):

    poleVectorArm = ["pV"]
    armRotate = [".rotate"]
    groupArm = ["group"]
    ikArm = ["ikArm"]
    handArm = ["Hand"]
    W0 = ["W0"]
    W1 = ["W1"]
    period = ["."]
    blend = [".blend"]

    orientConstraintShoulder = [cmds.orientConstraint (IKshoulder, FKshouldler, shoulder)]
    orientConstraintElbow = [cmds.orientConstraint (IKelbow, FKelbow, elbow)]

    # conditions
    if not cmds.objExists("groupMasterNode"):
        cmds.group (em=True, n="groupMasterNode")
    else:
        print("groupMasterNode exists")

    if not cmds.objExists("armConstraints"):
        cmds.group (em=True, n="armConstraints")
        cmds.parent ("armConstraints", "groupMasterNode")
    else:
        print("armConstraints exists")

    # parenting of constraints
    cmds.parent (orientConstraintShoulder[0], "armConstraints")
    cmds.parent (orientConstraintElbow[0], "armConstraints")

    #poleVectorSetup
    cmds.setAttr ((IKshoulder + armRotate[0]), armPVx, armPVy, armPVz)
    poleVectorControl = [cmds.spaceLocator (n = poleVectorArm[0] + IKshoulder)]
    pointConstraintpoleVectorControlDelete = [cmds.pointConstraint (IKelbow, poleVectorControl[0])]
    cmds.delete (pointConstraintpoleVectorControlDelete[0])
    grouppoleVectorControl = [cmds.group (em=True, n = (groupArm[0] + poleVectorArm[0] + IKshoulder))]
    groupPointConstraintpoleVectorControlDelete = [cmds.pointConstraint (poleVectorControl[0], grouppoleVectorControl[0])]
    cmds.delete (groupPointConstraintpoleVectorControlDelete[0])
    cmds.parent (poleVectorControl[0], grouppoleVectorControl[0])
    cmds.parent (grouppoleVectorControl[0], "groupMasterNode")
    cmds.setAttr ((IKshoulder + armRotate[0]), 0, 0, 0)

    # ikHandle
    ikHandleArm = [cmds.ikHandle (sj=IKshoulder, ee=IKhand)]
    GroupikHandleArm = [cmds.group (em=True, n = (groupArm[0] + IKshoulder))]
    cmds.parent (ikHandleArm[0][0:1], GroupikHandleArm[0])
    cmds.parent (GroupikHandleArm[0], "groupMasterNode")

    #handControl
    handControl = [cmds.spaceLocator (n = hand + IKshoulder)]
    pointConstrainthandControlDelete = [cmds.pointConstraint (hand, handControl[0])]
    cmds.delete (pointConstrainthandControlDelete[0])
    grouppolehandControl = [cmds.group (em=True, n = (groupArm[0] + IKhand))]
    grouppolehandControlDelete = [cmds.pointConstraint (handControl[0], grouppolehandControl[0])]
    cmds.delete (grouppolehandControlDelete[0])
    cmds.parent (handControl[0], grouppolehandControl[0])
    cmds.parent (grouppolehandControl[0], "groupMasterNode")
    cmds.parent (ikHandleArm[0][0:1], handControl[0])
    cmds.select (handControl[0])
    cmds.addAttr (shortName='blend', longName='blend', defaultValue=1.0, minValue=0, maxValue=1, k = True)

    #polevectorConstraint
    cmds.poleVectorConstraint( poleVectorControl[0], ikHandleArm[0][0:1])

    #isolation of constraint attributes shoulder
    orientSelectShoulder = [cmds.select (orientConstraintShoulder[0])]
    orientSelectShoulderArray = [cmds.listAttr (orientConstraintShoulder[0], s=True, r=True, w=True, c=True, k=True)]
    orientSelectShoulderArrayW0 = [(orientConstraintShoulder[0]+orientSelectShoulderArray[0][6:7])]
    orientSelectShoulderArrayW1 = [(orientConstraintShoulder[0]+orientSelectShoulderArray[0][7:8])]
    orientSelectShoulderArrayW0Isolated = [{"%s" % (orientSelectShoulderArrayW0[0][0])}]
    orientSelectShoulderArrayW1Isolated = [{"%s" % (orientSelectShoulderArrayW1[0][0])}]
    orientSelectShoulderArrayW0IsolatedName = [(orientSelectShoulderArrayW0Isolated[0] + period[0] + IKshoulder + W0[0])]
    orientSelectShoulderArrayW1IsolatedName = [(orientSelectShoulderArrayW1Isolated[0] + period[0] + FKshouldler + W1[0])]
    cmds.setAttr ((orientSelectShoulderArrayW0IsolatedName[0]), 1)
    cmds.setAttr ((orientSelectShoulderArrayW1IsolatedName[0]), 0)

    #isolation of constraint attributes elbow
    orientSelectElbow = [cmds.select (orientConstraintElbow[0])]
    orientSelectElbowArray = [cmds.listAttr (orientConstraintElbow[0], s=True, r=True, w=True, c=True, k=True)]
    orientSelectElbowArrayW0 = [(orientConstraintElbow[0]+orientSelectElbowArray[0][6:7])]
    orientSelectElbowArrayW1 = [(orientConstraintElbow[0]+orientSelectElbowArray[0][7:8])]
    orientSelectElbowArrayW0Isolated = [{"%s" % (orientSelectElbowArrayW0[0][0])}]
    orientSelectElbowArrayW1Isolated = [{"%s" % (orientSelectElbowArrayW1[0][0])}]
    orientSelectElbowArrayW0IsolatedName = [(orientSelectElbowArrayW0Isolated[0] + period[0] + IKelbow + W0[0])]
    orientSelectElbowArrayW1IsolatedName = [(orientSelectElbowArrayW1Isolated[0] + period[0] + FKelbow + W1[0])]
    cmds.setAttr ((orientSelectElbowArrayW0IsolatedName[0]), 1)
    cmds.setAttr ((orientSelectElbowArrayW1IsolatedName[0]), 0)

    #isolation of constraint attributes hand
    selectHandBlend = [(handControl[0])]
    selectHandBlendArray = [cmds.listAttr (handControl[0], s=True, r=True, w=True, c=True, k=True)]
    selectHandBlendArrayMaster = [handControl[0] + selectHandBlendArray[0][10:11]]
    selectHandBlendArrayMasterIsolated = [{"%s" % (selectHandBlendArrayMaster[0][0])}]
    selectHandBlendArrayMasterIsolatedBlend = [selectHandBlendArrayMasterIsolated[0]+ blend[0]]

    #setDrivenKeyframe for hand blend
    cmds.setAttr (selectHandBlendArrayMasterIsolatedBlend[0], 1)
    cmds.setAttr ((orientSelectShoulderArrayW0IsolatedName[0]), 1)

```

```
cmds.setAttr ((orientSelectShoulderArrayW1IsolatedName[0]), 0)
cmds.setAttr ((orientSelectElbowArrayW0IsolatedName[0]), 1)
cmds.setAttr ((orientSelectElbowArrayW1IsolatedName[0]), 0)

cmds.setDrivenKeyframe( orientSelectElbowArrayW0IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
cmds.setDrivenKeyframe( orientSelectElbowArrayW1IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
cmds.setDrivenKeyframe( orientSelectShoulderArrayW0IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
cmds.setDrivenKeyframe( orientSelectShoulderArrayW1IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )

cmds.setAttr (selectHandBlendArrayMasterIsolatedBlend[0], 0)
cmds.setAttr ((orientSelectShoulderArrayW0IsolatedName[0]), 0)
cmds.setAttr ((orientSelectShoulderArrayW1IsolatedName[0]), 1)
cmds.setAttr ((orientSelectElbowArrayW0IsolatedName[0]), 0)
cmds.setAttr ((orientSelectElbowArrayW1IsolatedName[0]), 1)

cmds.setDrivenKeyframe( orientSelectElbowArrayW0IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
cmds.setDrivenKeyframe( orientSelectElbowArrayW1IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
cmds.setDrivenKeyframe( orientSelectShoulderArrayW0IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
cmds.setDrivenKeyframe( orientSelectShoulderArrayW1IsolatedName[0], cd=selectHandBlendArrayMasterIsolatedBlend[0] )
```

```

# scale rott node to desired size prior to activating joints
# joints scaling off of IK and utility nodes to bulge mesh
# better suited for realtime simulation, although can work for render as well

import maya.cmds as cmds

def muscleJoint (startJoint, endJoint, targetLOC):

    startJointLOC = ["aLOC"]
    endJointLOC = ["bLOC"]
    translate = [".translate"]
    distance = [".distance"]
    LOC = ["LOC"]
    divide = ["divide"]
    input1X = [".input1X"]
    input2X = [".input2X"]
    distanceName = ["DistanceNode"]
    mDN = ["MultiplyDivideNode"]
    attrOperation = [".operation"]
    ikHandle = ["IKhandle"]
    outputX = [".outputX"]
    scaleX = [".scaleX"]
    group = ["Group"]

    # locator world Space Set Up for Distance
    distanceLocBase = (cmds.spaceLocator (n= (startJoint + LOC [0])))
    pointConstraintDistanceLocBase = [cmds.pointConstraint (startJoint,distanceLocBase[0])]
    distanceLocEnd = (cmds.spaceLocator (n= (endJoint + LOC [0])))
    pointConstraintDistanceLocEnd = [cmds.pointConstraint (targetLOC, distanceLocEnd[0])]

    # distance float points
    startJointDistance = (cmds.getAttr (distanceLocBase[0]+translate[0]))
    endJointDistance = (cmds.getAttr (distanceLocEnd[0]+translate[0]))

    #distanceNode
    distanceNode = [cmds.distanceDimension (sp = startJointDistance[0], ep = endJointDistance[0])]
    distanceDim = [cmds.rename ("distanceDimension1", startJoint + distanceName[0])]

    # setup of multiplyDivideNode
    multiplyDivide = [cmds.createNode ("multiplyDivide", name = (startJoint + divide[0]))]
    cmds.setAttr ((multiplyDivide[0] + attrOperation[0]), 2)
    cmds.connectAttr ((distanceDim[0] + distance[0]), (multiplyDivide[0] + input1X[0]))
    dimension = [cmds.getAttr (distanceDim[0] + distance[0])]
    cmds.setAttr ((multiplyDivide[0] + input2X[0]), (dimension[0]))

    # ik handle setup
    stretchyIK = [cmds.ikHandle ( n = (startJoint + ikHandle[0]), sj = startJoint, ee = endJoint)]

    # point constrain ikHandle to "Target Locator"
    ikHandleTargetLocatorPointConstraint = [cmds.pointConstraint (targetLOC,stretchyIK[0][0:1])]

    # send multiplyDivide node.attribute x output to start joint scale
    cmds.connectAttr ((multiplyDivide[0] + outputX[0]),(startJoint + scaleX[0]))

    # group for measuring Locators
    groupLocators = [cmds.group (em=True, n= (distanceLocBase[0] + distanceLocEnd[0] + group[0]))]
    cmds.parent (distanceLocBase[0], groupLocators[0])
    cmds.parent (distanceLocEnd[0], groupLocators[0])

    # group for dimensionNode
    groupDimensionNode = [cmds.group (em=True, n= (distanceDim[0] + group[0]))]
    cmds.parent (distanceDim[0], groupDimensionNode[0])

    # group for ikHandle
    groupStretchyIK = [cmds.group (em=True, n= (startJoint + (ikHandle[0]) + group[0]))]
    cmds.parent (stretchyIK[0][0:1], groupStretchyIK[0])

    # masternode
    if not cmds.objExists("groupMasterNode"):
        cmds.group (em=True, n="groupMasterNode")
    else:
        print("groupMasterNode exists")

    # visibility
    if not cmds.objExists("visibility"):
        cmds.group (em=True, n="visibility")
        cmds.parent ("visibility", "groupMasterNode")
    else:
        print("visibility exists")

    cmds.parent (groupLocators[0], "groupMasterNode")
    cmds.parent (groupDimensionNode[0], "visibility")
    cmds.parent (groupStretchyIK[0], "groupMasterNode")

```

```

# twist nodes activated to make gradual forearm twist
# should work with all wrist angles with no flipping

import maya.cmds as cmds
def forearmRotate (foreArm, forearmTwistOne, forearmTwistTwo, Wrist):

    # masterController
    master = ["Master"]
    forearmMasterLOC = [cmds.spaceLocator (n= foreArm + master[0])]
    forearmMasterLOC_PC_delete = [cmds.pointConstraint (foreArm, forearmMasterLOC[0])]
    cmds.delete (foreArmMasterLOC_PC_delete[0])
    forearmMasterLOC_OC_delete = [cmds.orientConstraint (foreArm, forearmMasterLOC[0])]
    cmds.delete (foreArmMasterLOC_OC_delete[0])
    forearmMasterLOC_PC = [cmds.parentConstraint (foreArm, forearmMasterLOC[0])]

    # apprenticeController
    apprentice = ["Apprentice"]
    forearmApprenticeLOC = [cmds.spaceLocator (n= foreArm + apprentice[0])]
    forearmApprenticeLOC_PC_delete = [cmds.pointConstraint (foreArm, forearmApprenticeLOC[0])]
    cmds.delete (foreArmApprenticeLOC_PC_delete[0])
    forearmApprenticeLOC_OC_delete = [cmds.orientConstraint (foreArm, forearmApprenticeLOC[0])]
    cmds.delete (foreArmApprenticeLOC_OC_delete[0])
    cmds.parent (foreArmApprenticeLOC[0], forearmMasterLOC[0])

    # apprentice aim constraint
    cmds.select (Wrist, forearmApprenticeLOC[0])
    apprenticeAimConstraint = [cmds.aimConstraint (mo=True, weight=1, u= (0, 0, 1), worldUpType= "objectrotation", worldUpVector= (0, 0, 1), worldUpObject= Wrist)]

    #creatNodeMultiplyDivide
    multiply = ["Multiply"]
    multiplyDivideForeArm = [cmds.createNode ("multiplyDivide", name = (foreArm + multiply[0]))]

    # setattr multiply divide node, connect Attr apprentice to node to joints
    forearmInput2X = [".input2X"]
    forearmInput1X = [".input1X"]
    forearmInput2Y = [".input2Y"]
    forearmInput1Y = [".input1Y"]
    nullRotX = [".rotateX"]
    nullRotY = [".rotateY"]
    outPutX = [".outputX"]
    outPutY = [".outputY"]
    cmds.setAttr ((multiplyDivideForeArm[0] + forearmInput2X[0]), .2)
    cmds.setAttr ((multiplyDivideForeArm[0] + forearmInput2Y[0]), .6)
    cmds.connectAttr ((foreArm + apprentice[0] + nullRotX[0]), (multiplyDivideForeArm[0] + forearmInput1X[0]))
    cmds.connectAttr ((foreArm + apprentice[0] + nullRotX[0]), (multiplyDivideForeArm[0] + forearmInput1Y[0]))
    cmds.connectAttr ((multiplyDivideForeArm[0] + outPutX[0]), (foreArmTwistOne + nullRotX[0]))
    cmds.connectAttr ((multiplyDivideForeArm[0] + outPutY[0]), (foreArmTwistTwo + nullRotX[0]))

    #parenting
    if not cmds.objExists("groupMasterNode"):
        cmds.group (em=True, n="groupMasterNode")
    else:
        print("groupMasterNode exists")

    if not cmds.objExists("foreArmGroup"):
        cmds.group (em=True, n="foreArmGroup")
        cmds.parent ("foreArmGroup", "groupMasterNode")
    else:
        print("orientConstraintNode exists")

    cmds.parent (foreArmMasterLOC[0], "foreArmGroup")

```

```

# target creation setup to work for my base skeleton
# must have a child joint or node in opposite direction of forearm to act as alternate target
# this script fakes a quaternion and will allow 180 degrees of rotation with no flipping (gradual deformation)

import maya.cmds as cmds
def shoulderSetupStart (clavicle, armUpShoulder, aimTarget, shoulderTwistOne, shoulderTwistTwo, shouldUpVectorX, shouldUpVectorY, shouldUpVectorZ):

    # arm master group
    master = ["master"]
    masterGroup = [cmds.group (em = True, n = (master[0] + armUpShoulder))]
    pointConstDel = [cmds.pointConstraint (armUpShoulder, masterGroup[0])]
    orientConstDel = [cmds.orientConstraint (armUpShoulder, masterGroup[0])]
    cmds.delete (pointConstDel[0])
    cmds.delete (orientConstDel[0])
    parentConst = [cmds.parentConstraint (clavicle, masterGroup[0])]

    # arm apprentice group
    apprentice = ["apprentice"]
    apprenticeGroup = [cmds.group (em = True, n = (apprentice[0] + armUpShoulder))]
    pointConstApprenticeDel = [cmds.pointConstraint (armUpShoulder, apprenticeGroup[0])]
    orientConstApprenticeDel = [cmds.orientConstraint (armUpShoulder, apprenticeGroup[0])]
    cmds.delete (pointConstApprenticeDel[0])
    cmds.delete (orientConstApprenticeDel[0])

    # arm twist group
    twist = ["twist"]
    twistGroup = [cmds.group (em = True, n = (twist[0] + armUpShoulder))]
    pointConsttwistDel = [cmds.pointConstraint (armUpShoulder, twistGroup[0])]
    orientConsttwistDel = [cmds.orientConstraint (armUpShoulder, twistGroup[0])]
    cmds.delete (pointConsttwistDel[0])
    cmds.delete (orientConsttwistDel[0])

    # masterTwistgroup
    twistTarget = ["twistTarget"]
    twistTargetGroup = [cmds.group (em = True, n = (twistTarget[0] + armUpShoulder))]
    pointConsttwistTargetDel = [cmds.pointConstraint (armUpShoulder, twistTargetGroup[0])]
    orientConsttwistTargetDel = [cmds.orientConstraint (armUpShoulder, twistTargetGroup[0])]
    cmds.delete (pointConsttwistTargetDel[0])
    cmds.delete (orientConsttwistTargetDel[0])

    # grouping of nulls
    cmds.parent (apprenticeGroup[0], masterGroup[0])
    cmds.parent (twistGroup[0], apprenticeGroup[0])
    cmds.parent (twistTargetGroup[0], twistGroup[0])

    # orientConstraint twist joint
    orientConsttwist = [cmds.orientConstraint (armUpShoulder, twistGroup[0])]

    #upVectorCreation and Location
    rotate = [".rotate"]
    upVector = ["upVector"]
    cmds.setAttr ((armUpShoulder + rotate[0]), shouldUpVectorX, shouldUpVectorY, shouldUpVectorZ)
    upVectorGroup = [cmds.group (em = True, n = (upVector[0] + armUpShoulder))]
    pointConstupVectorGroupDel = [cmds.pointConstraint (aimTarget, upVectorGroup[0])]
    cmds.delete (pointConstupVectorGroupDel[0])
    cmds.parent (upVectorGroup[0], masterGroup[0])
    cmds.setAttr ((armUpShoulder + rotate[0]), 0,0,0)

    #creatNodeMultiplyDivide
    multiply = ["MultiplyShoulder"]
    multiplyDivideShoulder = [cmds.createNode ("multiplyDivide", name = (armUpShoulder + multiply[0]))]

    # setAttr multiply divide node, connect Attr apprentice to node to joints
    shoulderInput2X = [".input2X"]
    shoulderInput1X = [".input1X"]
    shoulderInput2Y = [".input2Y"]
    shoulderInput1Y = [".input1Y"]
    shoulderNullRotX = [".rotateX"]
    shoulderNullRotY = [".rotateY"]
    shoulderOutPutX = [".outputX"]
    shoulderOutPutY = [".outputY"]
    cmds.setAttr ((multiplyDivideShoulder[0] + shoulderInput2X[0]), .6)
    cmds.setAttr ((multiplyDivideShoulder[0] + shoulderInput2Y[0]), .35)
    cmds.connectAttr ((twistTargetGroup[0] + shoulderNullRotX[0]), (multiplyDivideShoulder[0] + shoulderInput1X[0]))
    cmds.connectAttr ((twistTargetGroup[0] + shoulderNullRotY[0]), (multiplyDivideShoulder[0] + shoulderInput1Y[0]))
    cmds.connectAttr ((armUpShoulder + multiply[0] + shoulderOutPutX[0]), (shoulderTwistOne + shoulderNullRotX[0]))
    cmds.connectAttr ((armUpShoulder + multiply[0] + shoulderOutPutY[0]), (shoulderTwistTwo + shoulderNullRotX[0]))

    # aimConstraint for target will get to later
    cmds.select (aimTarget, twistTargetGroup[0])
    apprenticeAimConstraint = [cmds.aimConstraint (mo=True, weight=1, u= (0, 0, 1), worldUpType= "object", worldUpVector= (0, 0, 1), worldUpObject= upVectorGroup[0])]

    #parenting
    if not cmds.objExists("groupMasterNode"):
        cmds.group (em=True, n="groupMasterNode")
    else:
        print("groupMasterNode exists")

    if not cmds.objExists("shoulderArmGroup"):
        cmds.group (em=True, n="shoulderArmGroup")
        cmds.parent ("shoulderArmGroup", "groupMasterNode")

    cmds.parent (masterGroup[0], "shoulderArmGroup")

```

```
# script to reattribute multiply divide nodes if scaled
multiplyDivideSelected = (cmds.ls (sl=True))
len(multiplyDivideSelected)

for i in range(0, len(multiplyDivideSelected), 1):
    input1Xchange = (.input1X)
    input2Xchange = (.input2X)
    xTotalOne = (multiplyDivideSelected[i] + input1Xchange)
    xTotalTwo = [multiplyDivideSelected[i] + input2Xchange]
    xTotalNumber = [cmds.getAttr (xTotalOne)]
    cmds.setAttr ((xTotalTwo[0]), (xTotalNumber[0]))
```